KAPITEL 2

Der Data Engineering Lifecycle

Unser zentrales Anliegen ist es, Sie zu ermutigen, Data Engineering nicht nur als eine Sammlung von spezifischen Datentechnologien zu betrachten. Die Datenlandschaft erlebt eine regelrechte Explosion neuer Datentechnologien und -praktiken mit einem immer höheren Grad an Abstraktion und Benutzerfreundlichkeit. Aufgrund der zunehmenden technischen Abstraktion werden Data Engineers mehr und mehr zu *Data Lifecycle Engineers*, die nach den *Prinzipien* des Data-Lifecycle-Managements denken und arbeiten.

In diesem Kapitel befassen wir uns mit dem *Data Engineering Lifecycle*, dem zentralen Thema dieses Buchs. Der Data Engineering Lifecycle ist unser Konzept zur Beschreibung des Data Engineering von der »Wiege bis zur Bahre«. Außerdem stellen wir die bedeutenden Faktoren des Data Engineering Lifecycle vor, die eine wichtige Grundlage für alle Aktivitäten im Data Engineering bilden.

Was ist der Data Engineering Lifecycle?

Der Data Engineering Lifecycle umfasst mehrere Arbeitsgänge, in denen aus Rohdaten ein nutzbringendes Endprodukt entsteht, das von Analysten, Data Scientists, ML-Engineers und anderen genutzt werden kann. In diesem Kapitel werden die wichtigsten Phasen dieses Lebenszyklus vorgestellt, wobei der Schwerpunkt auf den Schlüsselkonzepten der einzelnen Phasen liegt und Details für spätere Kapitel aufgespart werden.

Wir unterteilen den Data Engineering Lifecycle in fünf Phasen (Abbildung 2-1, oben):

- Generierung
- Speicherung
- Ingestion
- Transformation
- Bereitstellung

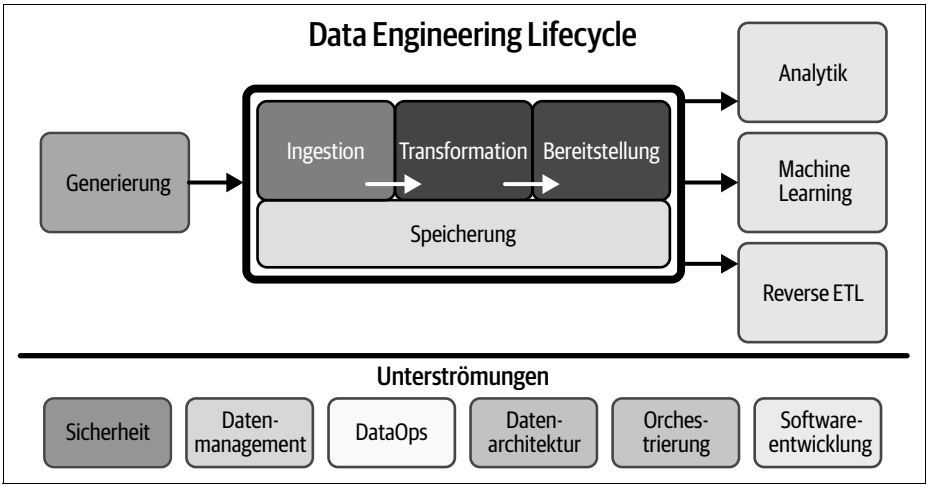

Abbildung 2-1: Phasen und bedeutende Faktoren des Data Engineering Lifecycle

In der ersten Phase rufen wir Daten aus Quellsystemen ab und speichern sie. Im nächsten Schritt wandeln wir die Daten um und widmen uns dann unserem eigentlichen Ziel, der Bereitstellung der Daten für Analysten, Data Scientists, ML-Engineers und andere. In der Praxis erfolgt die Speicherung der Daten während des gesamten Lebenszyklus, da die Daten vom Anfang bis zum Ende übertragen werden – daher zeigt das Diagramm die »Phase« der Speicherung als Grundstein, der die anderen Phasen untermauert.

Prinzipiell geraten die mittleren Phasen – Speicherung, Ingestion, Transformation – ein wenig durcheinander. Und das ist auch in Ordnung. Obwohl wir die einzelnen Teile des Lebenszyklus unterscheiden, handelt es sich nicht immer um einen geordneten, gleichmäßigen Ablauf. Verschiedene Phasen des Lebenszyklus können sich wiederholen, in anderer Reihenfolge auftreten, sich überschneiden oder anderweitig miteinander verwoben sein.

Das Fundament bilden die *Unterströmungen* (siehe Abbildung 2-1 unten), die sich durch mehrere Abschnitte des Data Engineering Lifecycle ziehen: Sicherheit, Datenmanagement, DataOps, Datenarchitektur, Orchestrierung und Softwareentwicklung. Kein einziger Teil des Data Engineering Lifecycle funktioniert ordnungsgemäß ohne diese Unterströmungen.

Datenlebenszyklus versus Data Engineering Lifecycle

Möglicherweise fragen Sie sich, worin der Unterschied zwischen dem Datenlebenszyklus insgesamt und dem Data Engineering Lifecycle besteht. Der Unterschied zwischen den beiden ist sehr fein: Der Data Engineering Lifecycle ist eine Teilmenge des Datenlebenszyklus (siehe Abbildung 2-2). Während sich der Datenlebenszyklus auf den gesamten Lebenszyklus der Daten bezieht, konzentriert sich der Data Engineering Lifecycle auf die Phasen, die ein Data Engineer kontrolliert.

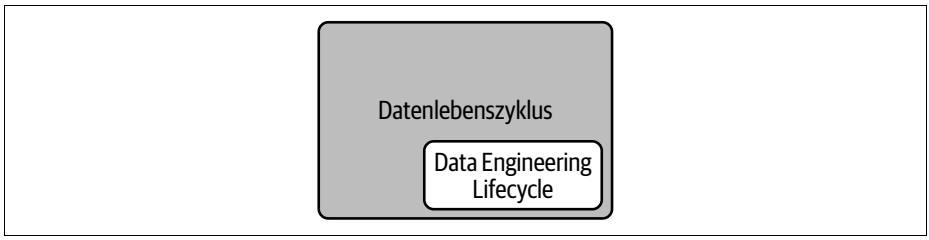

Abbildung 2-2: Der Data Engineering Lifecycle ist eine Teilmenge des Datenlebenszyklus.

Generierung: Quellsysteme

Ein *Quellsystem* ist der Herkunftsort der Daten, die im Data Engineering Lifecycle verwendet werden. Ein Quellsystem kann beispielsweise ein IoT-Gerät, eine Anwendungswarteschlange oder eine transaktionale Datenbank sein. Ein Data Engineer bezieht Daten aus einem Quellsystem, besitzt oder kontrolliert jedoch in der Regel nicht das Quellsystem selbst. Der Data Engineer muss die Funktionsweise der Quellsysteme, die von diesen erzeugten Daten, die Häufigkeit, mit der sich die Daten verändern, und die Varietät der generierten Daten verstehen.

Darüber hinaus müssen Data Engineers mit den Eigentümern der Quellsysteme im Dialog bleiben, wenn es um Änderungen geht, die Pipelines und Analysen beeinträchtigen könnten. Beispielsweise könnte der Programmcode die Struktur der Daten in einem Feld ändern, oder das Team könnte beschließen, das Backend auf eine völlig neue Datenbanktechnologie zu migrieren.

Eine große Herausforderung im Data Engineering ist die überwältigende Anzahl von Datenquellensystemen, mit denen Data Engineers arbeiten und die sie verstehen müssen. Um das zu veranschaulichen, betrachten wir zwei gängige Quellsysteme, von denen eines sehr traditionell (eine Anwendungsdatenbank) und das andere ein neueres Beispiel (ein IoT-Schwarm) ist.

Abbildung 2-3 zeigt ein klassisches Quellsystem mit mehreren Anwendungsservern, die von einer Datenbank unterstützt werden. Dieses Quellsystem wurde in den 1980ern mit dem überwältigenden Erfolg der relationalen Datenbankmanagementsysteme (RDBMS) populär. Die Kombination aus Anwendung und Datenbank ist auch heute noch beliebt, wobei es zahlreiche neue Ansätze für die Softwareentwicklung gibt.

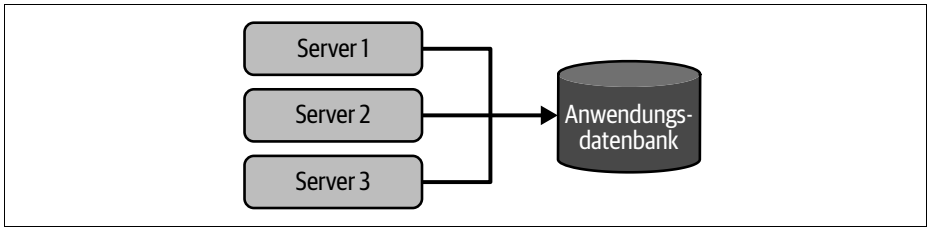

Abbildung 2-3: Beispiel für ein Quellsystem: eine Anwendungsdatenbank

Beispielsweise bestehen heutige Anwendungen oft aus vielen kleinen Dienst-Datenbank-Paaren mit Microservices statt aus einer einzigen riesigen Datenbank.

Betrachten wir ein weiteres Beispiel für ein Quellsystem. Abbildung 2-4 zeigt einen IoT-Schwarm: Eine Vielzahl von Geräten (Kreise) sendet Nachrichten (Rechtecke) an ein zentrales System. Mit der zunehmenden Verbreitung von IoT-Geräten wie Sensoren, Smart Devices und vielem mehr wird das IoT-Quellsystem immer gebräuchlicher.

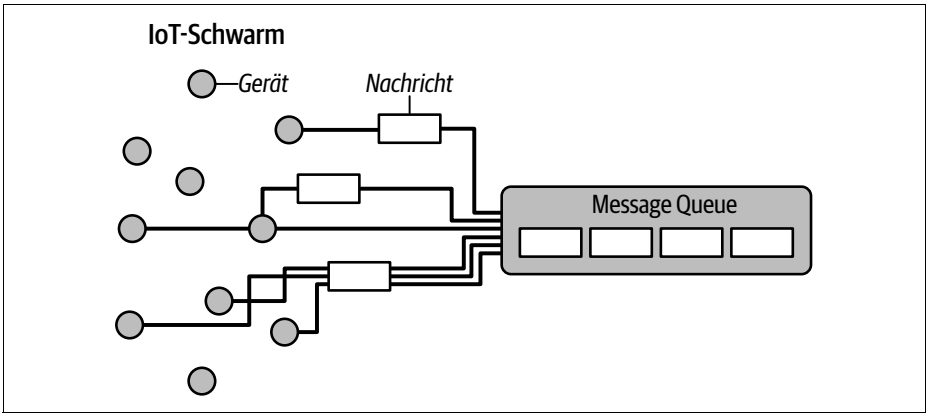

Abbildung 2-4: Beispiel für ein Quellsystem: ein IoT-Schwarm und eine Message Queue (Nachrichtenwarteschlange)

Bewertung von Quellsystemen: Wichtige technische Überlegungen

Bei der Beurteilung von Quellsystemen gibt es viele Dinge zu beachten, z.B. die Handhabung von Dateningestion, -zustand und -generierung durch das System. Im Folgenden finden Sie eine kleine Auswahl an Ausgangsfragen zur Bewertung von Quellsystemen, die Data Engineers berücksichtigen müssen:

- Was sind die wesentlichen Merkmale der Datenquelle? Handelt es sich um eine Anwendung? Ist es ein Schwarm von IoT-Geräten?
- Wie werden die Daten im Quellsystem aufbewahrt? Werden sie langfristig vorgehalten, oder liegen sie zeitlich begrenzt vor und werden schnell wieder gelöscht?
- Mit welcher Geschwindigkeit werden Daten erzeugt? Wie viele Ereignisse pro Sekunde? Wie viele Gigabytes pro Stunde?
- Wie konform sind die Ausgabedaten? Wenn Data Engineers die Qualität der Daten prüfen, wie oft treten dann Inkonsistenzen auf – Nullwerte, wo sie nicht erwartet werden, mangelhafte Formatierung usw.?
- Wie häufig treten Fehler auf?
- Enthalten die Daten Duplikate?

- Werden einige Datenwerte verspätet eintreffen, unter Umständen sogar viel später als andere zeitgleich produzierte Informationen?
- Wie sieht das Schema der eingelesenen Daten aus? Müssen Data Engineers mehrere Tabellen oder sogar mehrere Systeme verbinden, um ein vollständiges Bild der Daten zu erhalten?
- Wie wird bei Änderungen des Schemas (z. B. durch Hinzufügen einer neuen Spalte) verfahren, und wie wird dies den Beteiligten mitgeteilt?
- Wie regelmäßig sollten Daten aus dem Quellsystem abgerufen werden?
- Werden bei statischen Systemen (z. B. einer Datenbank, in der Kundenkontoinformationen erfasst werden) die Daten als regelmäßige Snapshots oder als Aktualisierungen aus der *Change Data Capture* (CDC) bereitgestellt? Wie sieht die Logik für die Durchführung von Änderungen aus, und wie werden diese in der Quelldatenbank verfolgt?
- Wer/was ist der Datenanbieter, der die Daten für die nachfolgende Nutzung übermittelt?
- Wirkt sich das Lesen aus einer Datenquelle auf deren Performance aus?
- Ist das Quellsystem von übergeordneten Daten abhängig? Was sind die Merkmale dieser vorgelagerten Systeme?
- Wird die Datenqualität überprüft, um zu ermitteln, ob Daten verspätet sind oder fehlen?

Alle Quellen erzeugen Daten, die von Folgesystemen genutzt werden, darunter von Hand erstellte Tabellen, IoT-Sensoren sowie Web- und mobile Anwendungen. Jede Quelle hat ihren eigenen Umfang und ihre eigene Taktung der Datengenerierung. Ein Data Engineer sollte daher verstehen, wie die Quelle Daten generiert, einschließlich relevanter Besonderheiten und Feinheiten. Data Engineers müssen auch die Schwachstellen der Quellsysteme kennen, mit denen sie interagieren. Können beispielsweise Abfragen in einer Quelldatenbank zu Ressourcenkonflikten und Leistungsproblemen führen?

Zu den anspruchsvollsten Aspekten von Quelldaten zählt das Schema. Das *Schema* legt die hierarchische Organisation der Daten fest. Folgerichtig kann man sich die Daten vom gesamten Quellsystem bis hin zu den einzelnen Tabellen und der Struktur der jeweiligen Felder vorstellen. Das Schema der von den Quellsystemen gelieferten Daten wird auf verschiedene Weise festgelegt. Zwei beliebte Optionen sind schemafreie und feste Schemata.

Schemafrei bedeutet nicht, dass es kein Schema gibt. Vielmehr bedeutet es, dass die jeweilige Anwendung das Schema definiert, während die Daten geschrieben werden, sei es in eine Nachrichtenwarteschlange (Message Queue), ein Flat File, einen Blob oder eine Dokumentendatenbank wie MongoDB. Ein traditionelleres auf relationalen Datenbanken basierendes Modell verwendet ein *festes Schema*, das in der Datenbank vorgegeben ist und an das sich die Anwendungen halten müssen.

Beide Modelle bergen Herausforderungen für Data Engineers. Schemata ändern sich im Laufe der Zeit; tatsächlich wird die Fortentwicklung von Schemata im Rahmen des agilen Ansatzes der Softwareentwicklung befürwortet. Ein Hauptbestandteil der Arbeit des Data Engineers besteht darin, Rohdaten in das Schema des Quellsystems zu übertragen und diese in brauchbare Informationen für Analysen umzuwandeln. Diese Aufgabe wird umso anspruchsvoller, je mehr sich das Quellschema weiterentwickelt.

In Kapitel 5 gehen wir ausführlicher auf Quellsysteme ein; Schemata und Datenmodellierung werden in den Kapiteln 6 und 8 behandelt.

Speicherung

Sie müssen Ihre Daten sichern. Die Wahl einer geeigneten Speicherlösung ist der entscheidende Faktor für den Erfolg im Datenlebenszyklus und aus verschiedenen Gründen auch eine der kompliziertesten Phasen des Lebenszyklus. Erstens kommen bei Datenarchitekturen in der Cloud oft *mehrere* Speicherlösungen zum Einsatz. Zweitens fungieren nur wenige Speichersysteme als reine Datenspeicher, vielmehr unterstützen viele von ihnen komplexe Transformationsabfragen; selbst Objektspeichersysteme können leistungsstarke Abfragefunktionen unterstützen – z. B. Amazon S3 Select (*https://oreil.ly/XzcKh*). Drittens ist die Datensicherung zwar eine Phase des Data Engineering Lifecycle, sie betrifft aber häufig auch andere Phasen wie Ingestion, Transformation und Bereitstellung der Daten.

Die Datensicherung erstreckt sich über den gesamten Data Engineering Lifecycle und kommt oft an mehreren Stellen in einer Datenpipeline vor, wobei sich Speichersysteme mit Quellsystemen, Ingestion, Transformation und Bereitstellung überschneiden. Die Art und Weise, wie Daten gespeichert werden, wirkt sich in vielerlei Hinsicht darauf aus, wie sie in allen Phasen des Data Engineering Lifecycle verwendet werden. Cloud Data Warehouses können beispielsweise Daten speichern, in Pipelines verarbeiten und für Analysten bereitstellen. Streaming-Frameworks wie Apache Kafka und Pulsar können gleichzeitig als Ingestion-, Speicher- und Abfragesysteme für Informationen fungieren, wobei die Objektspeicher eine Standardschicht für die Datenübertragung darstellen.

Bewertung von Speichersystemen: wichtige technische Überlegungen

Im Folgenden finden Sie einige wichtige Fragen, die Sie sich bei der Wahl eines Speichersystems für ein Data Warehouse, ein Data Lakehouse, eine Datenbank oder einen Objektspeicher stellen sollten:

- Ist das Speichersystem mit den erforderlichen Schreib- und Lesegeschwindigkeiten der Infrastruktur kompatibel?
- Führt die Datenspeicherung zu Engpässen bei nachgelagerten Prozessen?
- Ist Ihnen klar, wie das Speichersystem funktioniert? Nutzen Sie es optimal, oder begehen Sie Fehler? Wenden Sie z. B. in einem Objektspeichersystem eine

hohe Rate von zufälligen Zugriffsaktualisierungen an? (Dies ist ein schlechtes Vorgehen und führt zu erheblichen Leistungseinbußen).

- Kann dieses Speichersystem die künftig zu erwartende Expansion bewältigen? Sie sollten alle Kapazitätsgrenzen des Speichersystems berücksichtigen: gesamter verfügbarer Speicherplatz, Lesegeschwindigkeit, Schreibleistung usw.
- Können nachgelagerte Anwender und Prozesse die Daten im Rahmen des erforderlichen *Service Level Agreements* (SLA) abrufen?
- Werden auch Metadaten über die weitere Entwicklung des Schemas, den Datenfluss, die Datenherkunft usw. erfasst? Metadaten haben einen erheblichen Einfluss auf den Nutzen von Daten. Sie sind eine Investition für die Zukunft, da sie Auffindbarkeit von Informationen und das institutionelle Wissen erheblich verbessern und künftige Projekte und strukturelle Änderungen optimieren.
- Handelt es sich um eine reine Speicherlösung (Objektspeicher), oder unterstützt das System komplexe Abfragemuster (z. B. ein Cloud Data Warehouse)?
- Arbeitet das Speichersystem schemaunabhängig (Objektspeicher)? Mit einem flexiblen Schema (Cassandra)? Oder einem erzwungenen Schema (einem Cloud Data Warehouse)?
- Wie werden die Stammdaten, die Datenqualität der Golden Records und die Datenherkunft für Data Governance überwacht? (Mehr zu diesen Themen erfahren Sie in »Datenmanagement« auf Seite 82.)
- Wie gehen Sie mit der Einhaltung von gesetzlichen Vorgaben und der Datenhoheit um? Dürfen Sie zum Beispiel Ihre Daten an bestimmten geografischen Standorten aufbewahren, an anderen aber nicht?

Die Zugriffshäufigkeit auf Daten verstehen

Nicht auf alle Daten wird auf die gleiche Weise zugegriffen. Die Zugriffsmuster hängen stark von den gespeicherten und abgefragten Daten ab. Dies bringt den Begriff der *Temperatur* von Daten ins Spiel. Die Häufigkeit des Datenzugriffs bestimmt die Temperatur Ihrer Daten.

Daten, auf die am häufigsten zugegriffen wird, nennt man *heiße Daten*. Diese Daten werden in der Regel viele Male pro Tag, unter Umständen sogar mehrmals pro Sekunde abgerufen, z. B. in Systemen, die Benutzeranfragen bedienen. Diese Daten sollten so gespeichert werden, dass sie schnell abgerufen werden können, wobei sich »schnell« auf den jeweiligen Anwendungsfall bezieht. Auf *lauwarme Daten* wird seltener zugegriffen – etwa einmal pro Woche oder Monat.

Kalte Daten werden nur selten abgefragt und eignen sich für die Speicherung in einem Archivierungssystem. Diese Daten werden häufig zu Zwecken der Einhaltung von Vorschriften oder im Fall eines katastrophalen Ausfalls eines anderen Systems aufbewahrt. Früher wurden kalte Daten auf Bändern gespeichert und zu entfernten Archivierungseinrichtungen transportiert. In Cloud-Umgebungen bieten Anbieter spezialisierte Speicherschichten mit sehr günstigen monatlichen Speicherkosten, aber hohen Preisen für den Datenabruf an.

Die Wahl eines Speichersystems

Welches Speichersystem empfiehlt sich? Dies hängt von Ihren Anwendungsfällen, dem Datenvolumen, der Häufigkeit der Erfassung, dem Format und der Größe der zu erfassenden Daten ab – im Wesentlichen also von den in der vorangegangenen Liste aufgeführten Überlegungen. Es gibt keine allgemeingültige Empfehlung. Jedes Speichersystem hat ihre Nachteile. Es gibt unzählige Varianten von Speichertechnologien, und man ist leicht überfordert, wenn es darum geht, die beste Option für die eigene Datenarchitektur zu finden.

Kapitel 6 befasst sich ausführlicher mit bewährten Verfahren und Ansätzen für die Datensicherung sowie mit den Übergängen zwischen der Datensicherung und anderen Phasen des Lebenszyklus.

Ingestion

Nachdem Sie die Datenquelle, die Merkmale des von Ihnen verwendeten Quellsystems und die Datensicherung verstanden haben, müssen Sie die Daten sammeln. Die nächste Phase des Data Engineering Lifecycle ist die Übertragung der Daten aus den Quellsystemen.

Unserer Erfahrung nach verursachen die Quellsysteme und die Ingestion die größten Herausforderungen im Data Engineering Lifecycle. Die Quellsysteme befinden sich normalerweise außerhalb Ihrer direkten Kontrolle und können mitunter nicht mehr erreichbar sein oder liefern Daten von schlechter Qualität. Oder der Ingestionsdienst funktioniert plötzlich nicht mehr. Infolgedessen wird der Datenfluss gestoppt, oder es werden nicht genügend Daten für die Speicherung, Verarbeitung und Bereitstellung geliefert.

Unzuverlässige Quell- und Ingestionssysteme wirken sich negativ auf den gesamten Data Engineering Lifecycle aus. Sofern Sie sich mit den wichtigsten Fragen zu Quellsystemen auseinandergesetzt haben, sind Sie jedoch in guter Verfassung.

Wichtige technische Überlegungen für die Ingestionsphase

Bei der Entwicklung oder dem Aufbau eines Systems sollten Sie sich die folgenden Fragen zur Ingestion stellen:

- Was sind die Einsatzmöglichkeiten für die Daten, die ich aufnehme? Kann ich diese Daten wiederverwenden, anstatt mehrere Versionen desselben Datensatzes zu erstellen?
- Sind die Systeme, die diese Daten generieren und aufnehmen, zuverlässig, und sind die Daten verfügbar, wenn ich sie brauche?
- Wohin werden die Daten nach der Erfassung weitergeleitet?
- Wie häufig werde ich auf die Daten zugreifen?
- Wie umfangreich werden die Daten normalerweise sein?

- In welchem Format liegen die Daten vor? Können meine nachgelagerten Speicher- und Verarbeitungssysteme dieses Format verarbeiten?
- Sind die Quelldaten in einem guten Zustand, und können sie sofort weiterverarbeitet werden? Falls ja, für wie lange, und was könnte dazu führen, dass sie unbrauchbar werden?
- Müssen die Daten, die aus einer Streaming-Quelle stammen, transformiert werden, bevor sie ihr Ziel erreichen? Wäre eine In-Flight-Transformation möglich, bei der die Daten innerhalb des Streams selbst transformiert werden?

Dies sind nur einige der Punkte, über die Sie bei der Ingestion nachdenken müssen. Wir behandeln diese und weitere Fragen in Kapitel 7. Bevor wir uns der nächsten Phase widmen, wollen wir uns kurz zwei wichtigen Konzepten der Dateningestion zuwenden: Batch versus Streaming und Push versus Pull.

Batch versus Streaming

Praktisch alle Daten, mit denen wir es zu tun haben, sind von Natur aus *Streaming-Daten*. Die Daten werden fast immer direkt an ihrer Quelle erstellt und laufend aktualisiert. Die *Batch-Ingestion* ist einfach eine besondere und bequeme Art, diesen Datenstrom gebündelt zu verarbeiten, z. B. einen ganzen Tageswert von Daten in einem einzigen Batch.

Die Streaming-Ingestion ermöglicht uns, Daten kontinuierlich und in Echtzeit an nachgelagerte Systeme – andere Anwendungen, Datenbanken oder Analysesysteme – zu übermitteln. *Echtzeit* (oder *nahezu Echtzeit*) bedeutet hier, dass die Daten einem nachgelagerten System kurz nach ihrer Erzeugung zur Verfügung stehen (z. B. weniger als eine Sekunde später). Die für die Einstufung als Echtzeit erforderliche Latenzzeit variiert je nach Bereich und Anforderungen.

Batch-Daten werden entweder in einem vorbestimmten Zeitintervall oder bei Erreichen einer bestimmten Datengröße aufgenommen. Die Batch-Ingestion ist eine Einbahnstraße: Sobald die Daten in Batches aufgeteilt sind, ist die Latenz für nachgelagerte Verbraucher von Natur aus begrenzt. Aufgrund der Einschränkungen von bestehenden Systemen war die stapelweise Verarbeitung lange Zeit die Standardmethode für die Datenübertragung. Die Batch-Verarbeitung ist nach wie vor äußerst beliebt, um Daten für die Weiterverarbeitung zu erfassen, insbesondere bei Analysen und ML.

Die Trennung von Speicherung und Verarbeitung in vielen Systemen und die allgegenwärtige Verfügbarkeit von Streaming- und Event-Processing-Plattformen machen die kontinuierliche Verarbeitung von Datenströmen jedoch viel zugänglicher und beliebter. Die Wahl hängt weitgehend vom Anwendungsfall und den Erwartungen an die Aktualität der Daten ab.

Wichtige Überlegungen zur Batch- und Streaming-Ingestion

Sollten Sie sich für das Streamen der Daten entscheiden? Trotz der Attraktivität des Streamings gibt es viele Faktoren, die man berücksichtigen muss. Im Folgenden finden Sie einige Fragen, die Sie sich stellen sollten, wenn Sie beurteilen wollen, ob Streaming-Ingestion die richtige Wahl gegenüber Batch-Ingestion ist:

- Wenn ich die Daten in Echtzeit aufnehme, können die nachgelagerten Speichersysteme die Datenmengen verarbeiten?
- Benötige ich meine Daten im Millisekundentakt? Oder reicht ein Mikro-Batch-Konzept aus, bei dem die Daten z. B. im Minutentakt erfasst und aufgenommen werden?
- Was sind meine Anwendungsfälle für die Streaming-Ingestion? Welche praktischen Vorteile ergeben sich für mich aus der Implementierung von Streaming? Wenn ich Daten in Echtzeit erhalte, welche Aktionen kann ich dann mit diesen Daten durchführen, die eine Verbesserung gegenüber der Batch-Verarbeitung darstellen würden?
- Würde das Streamen mehr Zeit, Geld, Wartung, Ausfallzeiten und Alternativkosten verursachen als eine einfache Batch-Lösung?
- Sind meine Streaming-Pipeline und mein System zuverlässig und redundant, falls die Infrastruktur ausfällt?
- Welche Tools sind für meinen Anwendungsfall am besten geeignet? Sollte ich einen verwalteten Dienst nutzen (Amazon Kinesis, Google Cloud Pub/Sub, Google Cloud Dataflow) oder eigene Instanzen von Kafka, Flink, Spark, Pulsar usw. einrichten? Wenn ich mich für Letzteres entscheide, wer wird es verwalten? Was sind die Kosten und Risiken?
- Wenn ich ein ML-Modell einsetze, welche Vorteile habe ich dann mit Onlinevorhersagen und kontinuierlichem Training?
- Erhalte ich Daten von einer Live-Produktionsinstanz? Wenn ja, welche Auswirkungen hat mein Ingestionsprozess auf dieses Quellsystem?

Wie Sie sehen, scheint das Streamen von Daten eine gute Idee zu sein, aber es ist nicht immer einfach; es entstehen zusätzliche Kosten und Komplexitäten. Viele großartige Ingestion-Frameworks beherrschen sowohl Batch- als auch Mikro-Batch-Ingestion-Stile. Wir sind der Meinung, dass Batch ein hervorragender Ansatz für viele gängige Anwendungsfälle ist, z. B. für das Training von ML-Modellen und wöchentliche Berichte. Führen Sie Echtzeit-Streaming erst dann ein, wenn Sie einen Anwendungsfall gefunden haben, der die Nachteile gegenüber der Batch-Verarbeitung rechtfertigt.

Push versus Pull

Das *Push*-Modell der Datenzufuhr sieht vor, dass ein Quellsystem Daten in ein Zielsystem schreibt, sei es eine Datenbank, ein Objektspeicher oder ein Dateisys-

tem. Beim *Pull*-Modell werden die Daten aus dem Quellsystem abgerufen. Die Grenze zwischen dem Push- und dem Pull-Konzept ist oft fließend, da die Daten auf ihrem Weg durch die verschiedenen Phasen einer Datenpipeline sowohl geschoben als auch gezogen werden.

Betrachten wir zum Beispiel den ETL-Prozess (*Extrahieren, Transformieren, Laden*), der häufig in Batch-orientierten Ingestion-Workflows verwendet wird. Der *Extraktteil* (*E*) verdeutlicht, dass wir es mit einem Pull-Ingestion-Modell zu tun haben. Im herkömmlichen ETL-Prozess fragt das Ingestionssystem einen aktuellen Snapshot der Quelltabelle nach einem festen Zeitplan ab. Sie werden in diesem Buch mehr über ETL sowie auch über ELT (*Extrahieren, Laden, Transformieren*) erfahren.

Ein weiteres Beispiel ist die kontinuierliche CDC, die auf verschiedene Weise erreicht werden kann. Eine gängige Methode löst jedes Mal eine Meldung aus, wenn eine Zeile in der Quelldatenbank geändert wird. Diese Nachricht wird in eine Warteschlange *geschoben*, wo sie vom Ingestionssystem abgeholt wird. Eine andere gängige CDC-Methode verwendet Binärprotokolle, die jeden Commit an die Datenbank aufzeichnen. Die Datenbank *schiebt* die Daten in ihre Protokolle. Das Ingestionssystem liest diese Logs, interagiert aber ansonsten nicht direkt mit der Datenbank. Dadurch wird die Quelldatenbank wenig bis gar nicht zusätzlich belastet. Einige Versionen der Batch-CDC verwenden das *Pull*-Prinzip. Bei der zeitstempelbasierten CDC beispielsweise fragt ein Ingestionssystem die Quelldatenbank ab und ruft die Zeilen ab, die sich seit der letzten Aktualisierung geändert haben.

Beim Streamen werden die Daten unter Umgehung einer Backend-Datenbank direkt an einen Zielpunkt übertragen, wobei die Daten in der Regel von einer Event-Streaming-Plattform zwischengespeichert werden. Dieses Vorgehen ist nützlich bei Flotten von IoT-Sensoren, die Sensordaten aussenden. Anstatt sich auf eine Datenbank zu verlassen, um den aktuellen Zustand aufrechtzuerhalten, betrachten wir jeden aufgezeichneten Messwert als ein Ereignis. Dieses Verfahren wird auch bei Softwareanwendungen immer beliebter, da es die Echtzeitverarbeitung vereinfacht, App-Entwicklerinnen und -Entwicklern die Möglichkeit gibt, ihre Nachrichten für nachgelagerte Analysen anzupassen, und die Arbeit von Data Engineers erheblich vereinfacht.

Das optimale Vorgehen und die besten Methoden für die Ingestion werden in Kapitel 7 ausführlich behandelt. Wir wenden uns nun der Transformationsphase des Data Engineering Lifecycle zu.

Transformation

Nach der Dateningestion und -sicherung müssen Sie etwas mit den Daten tun. Der nächste Schritt im Data Engineering Lifecycle ist die *Transformation*, d.h., die Daten müssen aus ihrer ursprünglichen Form in eine für nachgelagerte Anwendungsfälle nützliche Struktur umgewandelt werden. Ohne eine ordnungsgemäße Trans-

formation sind die Daten nutzlos und liegen in einer für Berichte, Analysen oder ML unbrauchbaren Form vor. In der Regel ist die Umwandlungsphase der Zeitpunkt, an dem die Daten einen Wert für die nachgelagerte Nutzung schaffen.

Unmittelbar nach der Ingestion werden die Daten durch grundlegende Transformationen in korrekte Typen umgewandelt (z.B. Umwandlung der aufgenommenen Zeichenketten in numerische und Datumstypen), Datensätze in Standardformate gebracht und ungültige Datensätze entfernt. Spätere Transformationsstufen können das Datenschema umwandeln und eine Normalisierung vornehmen. In nachgelagerten Schritten können wir umfangreiche Aggregationen für die Berichterstattung vornehmen oder Daten für ML-Prozesse mit Features versehen.

Wichtige Überlegungen für die Transformationsphase

Bei der Planung von Datentransformationen sollten Sie Folgendes berücksichtigen:

- Wie hoch sind die Kosten und die Kapitalrendite der Umstellung? Wie hoch ist der damit verbundene Geschäftswert?
- Ist die Umwandlung so einfach und autark wie möglich?
- Welche Geschäftsregeln unterstützen die Transformationen?

Sie können Daten im Batch oder während des laufenden Streaming-Prozesses umwandeln. Wie bereits in »Ingestion« auf Seite 70 erwähnt, stammen praktisch alle Daten aus einem kontinuierlichen Strom; die Batch-Verarbeitung ist nur eine spezielle Art der Verarbeitung eines Datenstroms. Batch-Transformationen erfreuen sich zwar großer Beliebtheit, aber angesichts der wachsenden Popularität von Stream-Processing-Lösungen und der allgemeinen Zunahme von Streaming-Daten erwarten wir, dass die Nutzung von Streaming-Transformationen weiter zunehmen und die Batch-Verarbeitung in bestimmten Bereichen vielleicht bald vollständig ersetzen wird.

Zwar behandeln wir die Transformation als einen eigenständigen Bereich des Data Engineering Lifecycle, aber in der Praxis kann es viel komplizierter sein. Die Transformation ist häufig mit anderen Phasen des Lebenszyklus verwoben. Normalerweise werden die Daten in den Quellsystemen oder während des Ingestionsprozesses transformiert. Beispielsweise kann ein Quellsystem einem Datensatz einen Ereigniszeitstempel hinzufügen, bevor er an einen Ingestionsprozess weitergeleitet wird. Oder ein Datensatz wird innerhalb einer Streaming-Pipeline mit zusätzlichen Feldern und Berechnungen »angereichert«, bevor er an ein Data Warehouse gesendet wird. Transformationen sind in verschiedenen Bereichen des Lebenszyklus allgegenwärtig. Datenvorbereitung, Datenverarbeitung und Datenbereinigung – diese Transformationsaufgaben schaffen einen Mehrwert für die Endverbraucher von Daten.

Die Geschäftslogik ist eine treibende Kraft für die Datentransformation, die oft mit der Datenmodellierung verbunden ist. Daten übersetzen die Geschäftslogik in wiederverwendbare Elemente (z.B. bedeutet ein Verkauf: »Jemand hat bei mir 12 Bil-

derrahmen zu je 30 Dollar gekauft, also insgesamt für 360 Dollar.«). Die Datenmodellierung ist entscheidend, um ein klares und aktuelles Bild der Geschäftsprozesse zu erhalten. Eine einfache Ansicht der rohen Einzelhandelstransaktionen ist möglicherweise nicht nützlich, wenn nicht die Logik der Buchhaltungsregeln hinzugefügt wird, damit der CFO ein klares Bild der finanziellen Situation erhält. Sorgen Sie dafür, die Geschäftslogik standardisiert in Ihre Transformationen zu implementieren.

Die Datenfeaturisierung für ML ist ein weiterer Prozess der Datentransformation. Bei der Featurisierung geht es darum, Datenmerkmale zu extrahieren und zu verbessern, die für das Training von ML-Modellen nützlich sind. Featurisierung kann durchaus schwierig sein, da sie Fachwissen (um zu ermitteln, welche Merkmale für die Vorhersage wichtig sein könnten) mit umfassender Erfahrung in Data Science kombiniert. In diesem Buch geht es vor allem darum, dass Data Scientists, sobald sie herausgefunden haben, wie sie ihre Daten mit Merkmalen versehen können, die Featurisierungsprozesse in der Transformationsphase einer Datenpipeline automatisieren können.

Transformation ist ein umfangreiches Thema, und wir können ihm in dieser kurzen Einführung nicht gerecht werden. Kapitel 8 befasst sich eingehender mit Abfragen, Datenmodellierung sowie verschiedenen Umwandlungspraktiken und -besonderheiten.

Bereitstellung

Sie haben die letzte Phase des Data Engineering Lifecycle erreicht. Nachdem die Daten aufgenommen, gespeichert und in sinnvolle und nützliche Strukturen umgewandelt wurden, ist es nun an der Zeit, aus den Daten einen Mehrwert zu ziehen. »Wert aus Daten zu ziehen«, bedeutet für jeden Anwender etwas anderes.

Informationen haben einen *Wert*, wenn sie für praktische Zwecke verwendet werden. Daten, die nicht genutzt oder abgefragt werden, sind schlichtweg nutzlos. Prestigeprojekte sind ein großes Risiko für Unternehmen. Viele Unternehmen führten in der Big-Data-Ära solche Prestigeprojekte durch, indem sie riesige Datensätze in Data Lakes sammelten, die jedoch nie sinnvoll genutzt wurden. Die Cloud-Ära löst eine neue Welle von Prestigeprojekten aus, die auf den neuesten Data Warehouses, Objektspeichersystemen und Streaming-Technologien basieren. Datenprojekte müssen über den gesamten Lebenszyklus hinweg durchdacht sein. Was ist der wirtschaftliche Nutzen der so sorgfältig gesammelten, bereinigten und gespeicherten Daten?

Die Bereitstellung von Daten ist der wohl spannendste Teil des Data Engineering Lifecycle. Hier geschieht die Magie. Hier können ML-Engineers die fortschrittlichsten Techniken anwenden. Sehen wir uns einige der beliebtesten Nutzungsmöglichkeiten an: Datenanalyse, ML und Reverse ETL.

Datenanalyse

Die Datenanalyse ist das Herzstück der meisten Projekte. Sobald Ihre Daten gespeichert und umgewandelt sind, können Sie Berichte oder Dashboards erstellen und Ad-hoc-Analysen mit den Daten durchführen. Während der Großteil der Analytik früher BI (*Business Intelligence*) umfasste, schließt sie jetzt auch andere Facetten wie Operational Analytics (Betriebsanalyse) und Embedded Analytics (siehe Abbildung 2-5) mit ein. Wir werden kurz auf diese Varianten eingehen.

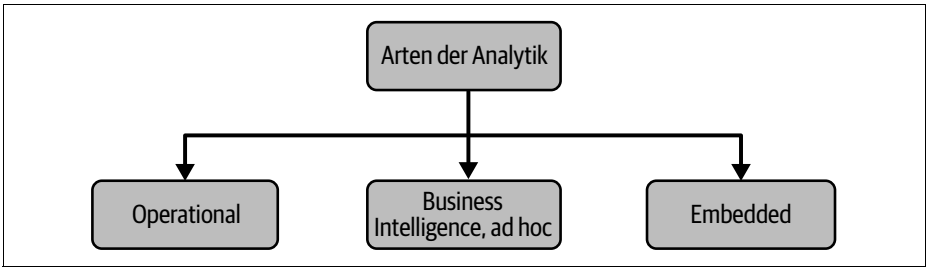

Abbildung 2-5: Verschiedene Arten der Analytik

Business Intelligence. BI fasst gesammelte Daten zusammen, um den vergangenen und aktuellen Zustand eines Unternehmens zu beschreiben. BI erfordert die Verwendung von Geschäftslogik zur Verarbeitung von Rohdaten. Beachten Sie, dass die Bereitstellung von Daten für Analysen ein weiterer Bereich ist, in dem die Phasen des Data Engineering Lifecycle durcheinandergeraten können. Wie bereits erwähnt, wird die Geschäftslogik oft in der Transformationsphase des Data Engineering Lifecycle auf die Daten angewandt, aber ein Logic-on-Read-Ansatz wird immer beliebter. Die Daten werden in einer sauberen, aber ziemlich rohen Form mit einem Minimum an nachverarbeitender Geschäftslogik gespeichert. Ein BI-System verwaltet ein Repository mit Geschäftslogik und Definitionen. Diese Geschäftslogik wird verwendet, um das Data Warehouse abzufragen, damit die Berichte und Dashboards mit den Geschäftsdefinitionen übereinstimmen.

Mit zunehmender Datenreife wird ein Unternehmen von der Ad-hoc-Datenanalyse zur Self-Service-Analyse übergehen, die den Geschäftsanwendern einen verbesserten Datenzugriff ermöglicht, ohne dass die IT-Abteilung eingreifen muss. Die Fähigkeit, Self-Service-Analysen durchzuführen, setzt voraus, dass die Daten so gut sind, dass die Mitarbeitenden im gesamten Unternehmen einfach selbst darauf zugreifen, sie nach Belieben zerlegen und sofort Erkenntnisse gewinnen können. Obwohl Self-Service-Analysen in der Theorie einfach sind, lassen sie sich in der Praxis nur schwer umsetzen. Der Hauptgrund dafür ist, dass schlechte Datenqualität, organisatorische Insellösungen und ein Mangel an angemessenen Datenkenntnissen einer umfassenden Nutzung von Analysen oft im Wege stehen.

Operational Analytics. Die Betriebsanalyse konzentriert sich auf die feinen Details der Vorgänge und fördert Maßnahmen, auf die ein Nutzer der Berichte sofort re-

agieren kann. Bei der Betriebsanalyse kann es sich um eine Live-Ansicht des Bestands oder ein Echtzeit-Dashboard des Website- oder Anwendungsstatus handeln. In diesem Fall werden die Daten in Echtzeit abgerufen, entweder direkt von einem Quellsystem oder von einer Streaming-Datenpipeline. Die Arten von Erkenntnissen in der Betriebsanalyse unterscheiden sich von der traditionellen BI, da sich Operational Analytics auf die Gegenwart konzentriert und nicht unbedingt auf historische Trends.

Embedded Analytics. Sie fragen sich vielleicht, warum wir Embedded Analytics (kundenorientierte Analyse) getrennt von BI aufgeführt haben. In der Praxis sind Analysen, die Kunden auf einer SaaS-Plattform zur Verfügung gestellt werden, mit einer Reihe separater Anforderungen und Komplikationen verbunden. Interne BI richtet sich an eine begrenzte Zielgruppe und bietet im Allgemeinen eine begrenzte Anzahl einheitlicher Ansichten. Zugriffskontrollen sind wichtig, aber nicht besonders kompliziert. Der Zugriff wird über eine Handvoll von Rollen und Zugriffsebenen verwaltet.

Bei Embedded Analytics steigt die Anzahl der Berichtsanfragen und die damit verbundene Belastung der Analysesysteme deutlich an; die Zugriffskontrolle wird wesentlich komplizierter und kritischer. Unternehmen können Analysen und Daten für Tausende oder mehr Kunden getrennt bereitstellen. Jeder Kunde muss seine Daten und nur seine Daten sehen können. Ein interner Fehler beim Datenzugriff in einem Unternehmen würde wahrscheinlich zu einer verfahrenstechnischen Überprüfung führen. Ein Datenleck zwischen Kunden würde einen massiven Vertrauensbruch bedeuten, der zu Medienaufmerksamkeit und einem erheblichen Kundenverlust führen würde. Minimieren Sie Ihren Aktionsradius in Bezug auf Datenlecks und Sicherheitslücken. Wenden Sie Sicherheitsmaßnahmen auf Kunden- oder Datenebene innerhalb Ihres Speichers und überall dort an, wo die Möglichkeit eines Datenlecks bestehen könnte.

Mehrmandantenfähigkeit

Viele aktuelle Speicher- und Analysesysteme unterstützen die Mehrmandantenfähigkeit auf unterschiedliche Weise. Data Engineers können sich dafür entscheiden, Daten für viele Kunden in gemeinsamen Tabellen zu speichern, um eine einheitliche Ansicht für interne Analysen und ML zu ermöglichen. Diese Daten werden den einzelnen Kunden über logische Ansichten mit entsprechend definierten Kontrollen und Filtern präsentiert. Es obliegt den Data Engineers, die Feinheiten der Mandantenfähigkeit in den von ihnen eingesetzten Systemen zu verstehen, um absolute Datensicherheit und -isolierung zu gewährleisten.

Machine Learning

Das Entstehen und der Erfolg von ML ist eine der spannendsten technologischen Revolutionen. Sobald Unternehmen einen hohen Grad an Datenreife erreicht haben, können sie damit beginnen, Probleme zu identifizieren, die sich für ML eignen, und eine entsprechende Lösung zu entwickeln.

Die Aufgaben des Data Engineers überschneiden sich in den Bereichen Analytik und ML erheblich, und die Grenzen zwischen Data Engineering, ML-Engineering und Analytics Engineering können fließend sein. Ein Data Engineer muss beispielsweise Spark-Cluster unterstützen, die Analysepipelines und ML-Modelltraining erleichtern. Möglicherweise muss er auch ein System bereitstellen, das Aufgaben teamübergreifend orchestriert und Metadaten- und Katalogisierungssysteme unterstützt, die den Datenverlauf und die Datenherkunft verfolgen. Die Festlegung dieser Verantwortungsbereiche und der entsprechenden Berichtsstrukturen ist eine wichtige strategische Überlegung.

Der Feature Store ist ein kürzlich entwickeltes Tool, das Data Engineering und ML-Engineering kombiniert. Feature Stores sollen den operativen Aufwand für ML-Engineers verringern, indem sie die Feature-Historie und die Feature-Versionen verwalten, die gemeinsame Nutzung von Features durch Teams unterstützen und grundlegende operative und Orchestrierungsfunktionen wie Backfilling bereitstellen. In der Praxis sind Data Engineers Teil des Kernsupportteams für Feature Stores, um ML-Engineering zu unterstützen.

Sollte ein Data Engineer mit ML vertraut sein? Es ist sicherlich hilfreich. Unabhängig von den operativen Grenzen zwischen Data Engineering, ML-Engineering, Business Analytics usw. sollten Data Engineers das operative Wissen über ihre Teams aufrechterhalten. Ein guter Data Engineer kennt die grundlegenden ML-Techniken und die damit verbundenen Datenverarbeitungsanforderungen, die Anwendungsfälle für Modelle in seinem Unternehmen und die Zuständigkeiten der verschiedenen Analyseteams im Unternehmen. Dies trägt zur Aufrechterhaltung einer effizienten Kommunikation bei und erleichtert die Zusammenarbeit. Im Idealfall entwickeln Data Engineers in Zusammenarbeit mit anderen Teams Tools, die keines der Teams allein entwickeln könnte.

Dieses Buch kann unmöglich ML im Detail behandeln. Ein wachsendes Angebot an Büchern, Videos, Artikeln und Communities steht Ihnen zur Verfügung, sofern Sie mehr erfahren möchten; einige Vorschläge finden Sie in »Weitere Quellen« auf Seite 103.

Im Folgenden werden einige Überlegungen für die Phase der Datenbereitstellung speziell für ML aufgeführt:

- Sind die Daten von ausreichender Qualität, um ein zuverlässiges Feature Engineering durchzuführen? Qualitätsanforderungen und -bewertungen werden in enger Zusammenarbeit mit den Teams, die die Daten nutzen, entwickelt.

- Sind die Daten gut zugänglich? Können Data Scientists und ML-Engineers wertvolle Daten leicht finden?
- Wo sind die technischen und organisatorischen Grenzen zwischen Data Engineering und ML-Engineering? Diese organisatorische Frage hat erhebliche Auswirkungen auf die gesamte Infrastruktur.
- Bildet der Datensatz die Realität richtig ab? Ist er unverhältnismäßig verzerrt?

ML ist zwar spannend, aber unserer Erfahrung nach stürzen sich Unternehmen oft vorschnell in die Materie. Bevor Sie eine Menge Ressourcen in ML investieren, sollten Sie sich die Zeit nehmen, eine solide Datengrundlage zu schaffen. Das bedeutet, dass Sie die besten Systeme und Architekturen für den gesamten Lebenszyklus von Data Engineering und ML einrichten müssen. In der Regel ist es am besten, wenn man erst einmal Kompetenz in der Analytik entwickelt, bevor man zu ML übergeht. Viele Unternehmen haben ihre ML-Träume aufgegeben, weil sie Initiativen ohne geeignete Grundlagen gestartet haben.

Reverse ETL

Reverse ETL ist seit Langem eine gängige Praxis im Bereich der Datenverarbeitung, die allerdings nur ungern thematisiert wird. Beim Reverse ETL werden die am Ende des Data Engineering Lifecycle gewonnenen Daten in die Quellsysteme zurückgeleitet (siehe Abbildung 2-6). In der Praxis ist dieser Ablauf von Vorteil und oft auch notwendig. Reverse ETL ermöglicht uns, Analysen, Bewertungsmodelle usw. in bestehende Systeme oder SaaS-Plattformen zurückzuspeisen.

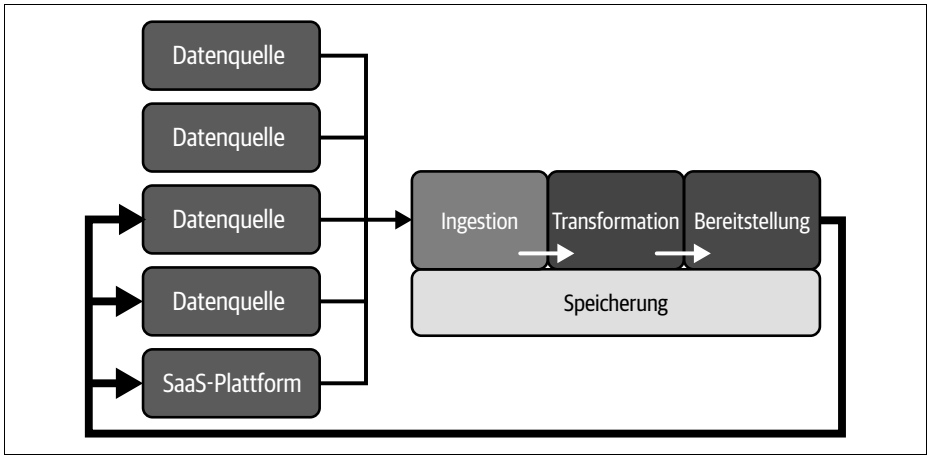

Abbildung 2-6: Reverse ETL

Marketinganalysten mussten Gebote in Microsoft Excel anhand der in ihrem Data Warehouse vorhandenen Daten berechnen und diese anschließend in Google Ads hochladen. Dieser Prozess erfolgte oft händisch und war primitiv.

Während wir an diesem Buch arbeiten, haben mehrere Anbieter das Konzept der Reverse ETL aufgegriffen und entsprechende Produkte entwickelt, z. B. Hightouch und Census. Reverse ETL befindet sich als Verfahren noch im Anfangsstadium, aber wir vermuten, dass es sich durchsetzen wird.

Reverse ETL ist besonders wichtig geworden, da Unternehmen zunehmend auf SaaS und externe Plattformen zurückgreifen. So können Unternehmen beispielsweise bestimmte Metriken aus ihrem Data Warehouse an eine Kundendatenplattform oder ein CRM-System weiterleiten. Werbeplattformen sind ein weiterer alltäglicher Anwendungsfall, wie im Beispiel von Google Ads. Es ist mit einer verstärkten Aktivität im Bereich Reverse ETL zu rechnen, sodass es zu Überschneidungen zwischen Data Engineering und ML-Engineering kommen wird.

Es bleibt abzuwarten, ob sich der Begriff *Reverse ETL* durchsetzen wird. Möglicherweise wird sich die Methode weiterentwickeln. Einige behaupten, dass wir Reverse ETL vermeiden können, indem wir Datentransformation in einem Eventstream verarbeiten und diese Ereignisse bei Bedarf an die Quellsysteme zurücksenden. Ob sich dieses Vorgehen in allen Geschäftsbereichen durchsetzen wird, steht auf einem anderen Blatt. Der springende Punkt ist, dass verarbeitete Daten auf die eine oder andere Weise an die Quellsysteme zurückgeführt werden müssen, möglichst mit der korrekten Zuordnung zum Quellsystem und dem entsprechenden Geschäftsprozess.

Die wesentlichen Unterströmungen innerhalb des Data Engineering Lifecycle

Data Engineering entwickelt sich ständig weiter. Während sich das Data Engineering in der Vergangenheit lediglich auf die technologische Ebene konzentrierte, hat sich dieser Schwerpunkt durch die fortschreitende Abstraktion und Vereinfachung von Tools und Verfahren verschoben. Data Engineering umfasst mittlerweile weit mehr als Tools und Technologien und integriert nun auch klassische Unternehmenspraktiken wie Datenmanagement und Kostenoptimierung sowie neuere Praktiken wie DataOps.

Wir bezeichnen diese als *Unterströmungen* – Sicherheit, Datenmanagement, DataOps, Datenarchitektur, Orchestrierung und Softwareentwicklung –, die jeden Abschnitt des Data Engineering Lifecycle (siehe Abbildung 2-7) unterstützen. In diesem Abschnitt geben wir einen kurzen Überblick über diese Unterströmungen und ihre Hauptbestandteile, die Sie im Laufe des Buchs noch genauer kennenlernen werden.

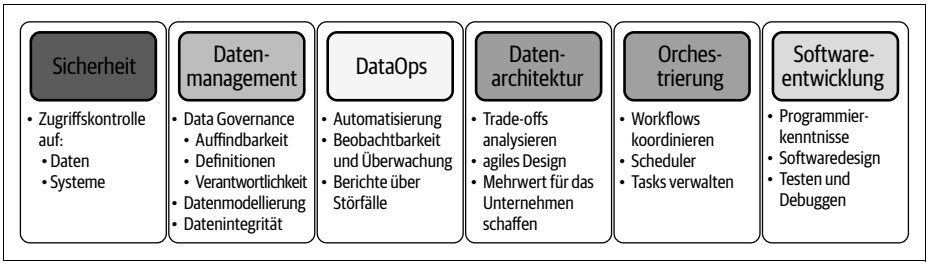

Abbildung 2-7: Die wesentlichen Unterströmungen des Data Engineering

Sicherheit

Das Thema Sicherheit muss für Data Engineers an erster Stelle stehen, und diejenigen, die es missachten, tun dies auf eigene Gefahr. Data Engineers müssen sich sowohl mit der Daten- als auch mit der Zugriffssicherheit auskennen und das *Prinzip der geringsten Privilegien* anwenden. Das Prinzip der geringsten Privilegien (*https://oreil.ly/6RGAq*) besagt, dass ein Benutzer oder ein System nur auf die Daten und Ressourcen zugreifen darf, die für die Ausführung einer bestimmten Funktion erforderlich sind. Ein häufiges Fehlverhalten, das bei Data Engineers mit wenig Erfahrung im Bereich Sicherheit zu beobachten ist, besteht darin, allen Benutzerinnen und Benutzern Administratorzugriff zu gewähren. Dieses Vorgehen ist eine vorprogrammierte Katastrophe!

Erteilen Sie den Benutzern nur den Zugriff, den diese für ihre aktuelle Arbeit benötigen, und nicht mehr. Arbeiten Sie nicht von einer Root-Shell aus, wenn Sie nur nach sichtbaren Dateien mit normalem Benutzerzugriff suchen. Verwenden Sie bei der Abfrage von Tabellen in einer Datenbank nicht die Superuser-Rolle, wenn eine untergeordnete ausreicht. Wenn wir uns das Prinzip der geringsten Privilegien aneignen, können wir ungewollte Schäden vermeiden und eine sicherheitsorientierte Denkweise beibehalten.

In jedem Unternehmen stellen die Menschen und die Organisationsstruktur die größten Sicherheitslücken dar. Wenn wir in den Medien von schweren Sicherheitsverstößen hören, stellt sich oft heraus, dass jemand im Unternehmen grundlegende Vorsichtsmaßnahmen ignoriert hat, einem Phishing-Angriff zum Opfer gefallen ist oder anderweitig unverantwortlich gehandelt hat. Die erste Schutzmaßnahme für die Datensicherheit besteht darin, eine Sicherheitskultur zu schaffen, die das gesamte Unternehmen durchdringt. Alle Personen, die Zugang zu Daten haben, müssen sich ihrer Verantwortung für den Schutz der sensiblen Daten des Unternehmens und seiner Kunden bewusst sein.

Bei der Datensicherheit geht es auch um das Timing, d. h. darum, dass genau die Personen und Systeme Zugriff auf die Daten haben, die sie benötigen, und zwar *nur so lange, wie es für die Durchführung ihrer Arbeit erforderlich ist*. Daten sollten durch Verschlüsselung, Tokenisierung, Anonymisierung, Verfremdung und Zu-

griffskontrollen vor unerwünschter Einsicht geschützt werden, sowohl während als auch nach der Übertragung.

Data Engineers müssen kompetente Sicherheitsadministratoren sein, da die Sicherheit in ihren Bereich fällt. Ein Data Engineer sollte die besten Sicherheitsverfahren für die Cloud und vor Ort kennen. Kenntnisse über Benutzer- und Identitätsmanagement (IdM), Rollen, Richtlinien, Gruppen, Netzwerksicherheit, Passwortrichtlinien und Verschlüsselung sind ein guter Anfang.

In diesem Buch werden immer wieder Bereiche hervorgehoben, in denen die Sicherheit im Data Engineering Lifecycle an erster Stelle stehen sollte. Detailliertere Einblicke in die Sicherheit erhalten Sie auch in Kapitel 10.

Datenmanagement

Vielleicht denken Sie, dass Datenmanagement sehr ... korporativ klingt. »Altmodische« Methoden des Datenmanagement finden ihren Weg in das Data Engineering und das ML-Engineering. Alte Trends werden wieder populär: Zwar gibt es Datenmanagement schon seit Jahrzehnten, aber bis vor Kurzem konnte es sich nicht im Data Engineering durchsetzen. Die Datentools werden immer einfacher, und die Komplexität, mit der Data Engineers umgehen müssen, nimmt ab. Infolgedessen steigt der Data Engineer in der Wertschöpfungskette zur nächsten Stufe der Best Practices auf. Best Practices für Daten, die früher nur großen Unternehmen vorbehalten waren – Data Governance, Master Data Management, Datenqualitätsmanagement, Metadatenmanagement –, werden nun von Unternehmen aller Größen und Reifegrade eingesetzt. Data Engineering wird, wie wir zu sagen pflegen, »unternehmerisch«. Das ist großartig!

Die *Data Management Association International* (DAMA) ist Herausgeber des *Data Management Body of Knowledge* (*DMBOK*), das wir für das maßgebliche Standardwerk für das Datenmanagement in Unternehmen halten. Darin wird Datenmanagement folgendermaßen definiert:

> Datenmanagement umfasst die Entwicklung, Ausführung und Überwachung von Plänen, Richtlinien, Programmen und Praktiken zur Bereitstellung, Kontrolle, zum Schutz und zur Steigerung des Werts von Daten und Informationsbeständen während ihres gesamten Lebenszyklus.

Wie hängt diese Definition mit Data Engineering zusammen? Data Engineers verwalten den Lebenszyklus von Daten, und das Datenmanagement umfasst eine Reihe von bewährten Verfahren, die Data Engineers zur Erfüllung dieser Aufgabe einsetzen, sowohl technisch als auch strategisch. Ohne einen Rahmen für die Handhabung von Daten sind Data Engineers lediglich Fachkräfte, die in einem Vakuum arbeiten. Data Engineers benötigen eine umfassende Sichtweise auf den Nutzen von Daten im gesamten Unternehmen, von den Quellsystemen bis zur Chefetage – und überall dazwischen.

Warum ist Datenmanagement wichtig? Datenmanagement zeigt, dass Daten für den täglichen Betrieb von entscheidender Bedeutung sind, so wie Unternehmen finanzielle Ressourcen, produzierte Waren oder Immobilien als Vermögenswerte betrachten. Das Datenmanagement bietet einheitliche Vorgaben, die jeder übernehmen kann, um sicherzustellen, dass das Unternehmen aus den Daten einen Nutzen zieht und sie angemessen behandelt.

Datenmanagement hat eine ganze Reihe von Aufgaben, unter anderem:

- Data Governance, einschließlich Auffindbarkeit und Verantwortlichkeit
- Datenmodellierung und -design
- Datenherkunft
- Speicherung und Handhabung
- Integration und Interoperabilität von Daten
- Verwaltung des Lebenszyklus von Daten
- Datensysteme für erweiterte Datenanalyse und ML
- Ethik und Datenschutz

Auch wenn dieses Buch keineswegs eine erschöpfende Quelle für das Datenmanagement ist, wollen wir doch kurz auf einige wichtige Punkte aus jedem Bereich eingehen, sofern sie einen Bezug zum Data Engineering haben.

Data Governance

Laut *Data Governance: The Definitive Guide* ist »Data Governance in erster Linie eine Datenverwaltungsfunktion, die die Qualität, Integrität, Sicherheit und Nutzbarkeit der von einer Einrichtung gesammelten Daten gewährleistet.«[1]

Wir erweitern diese Definition und fügen hinzu, dass Data Governance Menschen, Prozesse und Technologien einbezieht, um den Wert der Daten im gesamten Unternehmen zu maximieren und gleichzeitig die Daten durch angemessene Sicherheitskontrollen zu schützen. Wirksame Data Governance wird mit Sorgfalt entwickelt und von der Organisation unterstützt. Wenn Data Governance zufällig und planlos ist, können die Folgen von nicht vertrauenswürdigen Daten bis hin zu Sicherheitsverstößen und allem dazwischen reichen. Durch eine bewusste Data Governance werden die Datenkapazitäten des Unternehmens und der aus den Daten generierte Wert maximiert. Es wird auch (hoffentlich) verhindern, dass ein Unternehmen wegen fragwürdiger oder fahrlässiger Datenpraktiken in die Schlagzeilen gerät.

Stellen Sie sich ein typisches Beispiel für eine mangelhafte Datenverwaltung vor. Eine Unternehmensanalystin erhält eine Anfrage für einen Bericht, weiß aber nicht, welche Daten sie zur Beantwortung der Frage verwenden soll. Sie kann Stunden

1 Evren Eryurek et al., *Data Governance: The Definitive Guide* (Sebastopol, CA: O'Reilly, 2021), 1, *https://oreil.ly/LFT4d*.

damit verbringen, Dutzende von Tabellen in einer Transaktionsdatenbank zu durchforsten und wahllos zu entscheiden, welche Felder nützlich sein könnten. Die Analystin stellt einen »richtungsweisenden« Bericht zusammen, ist sich aber nicht ganz sicher, ob die dem Bericht zugrunde liegenden Daten korrekt und aussagekräftig sind. Auch der Empfänger des Berichts stellt die Gültigkeit der Daten infrage. Die Integrität der Analystin – und aller Daten in den Systemen des Unternehmens – wird infrage gestellt. Das Unternehmen ist über seine Leistung im Unklaren, was eine Geschäftsplanung unmöglich macht.

Data Governance ist die Grundlage für datengesteuerte Geschäftspraktiken und ein entscheidender Bestandteil des Data Engineering Lifecycle. Wenn Data Governance gut praktiziert wird, sind Menschen, Prozesse und Technologien so aufeinander abgestimmt, dass Daten als wichtige Geschäftsfaktoren behandelt werden. Wenn Datenprobleme auftreten, werden sie umgehend gelöst.

Die wichtigsten Aspekte der Data Governance sind Auffindbarkeit, Sicherheit und Verantwortlichkeit.[2] Innerhalb dieser Hauptgruppen gibt es Unterkategorien, wie Datenqualität, Metadaten und Datenschutz. Wir werden uns jede Hauptgruppe der Reihe nach ansehen.

Auffindbarkeit. In einem datengesteuerten Unternehmen müssen die Daten verfügbar und auffindbar sein. Die Endnutzer sollten schnellen und zuverlässigen Zugang zu den Daten haben, die sie für ihre Arbeit benötigen. Sie sollten wissen, woher die Daten stammen, wie sie mit anderen Daten in Beziehung stehen und was die Daten bedeuten.

Zu den Schwerpunkten der Auffindbarkeit von Daten gehören die Verwaltung von Metadaten und die Verwaltung von Stammdaten. Lassen Sie uns diese Bereiche kurz vorstellen.

Metadaten. *Metadaten* sind »Daten über Daten« und bilden die Grundlage für jeden Abschnitt des Data Engineering Lifecycle. Metadaten sind genau die Daten, die benötigt werden, um Daten auffindbar und beherrschbar zu machen.

Wir unterteilen Metadaten in zwei Kategorien: automatisch generierte und von Menschenhand erstellte. Im heutigen Data Engineering dreht sich alles um Automatisierung, aber die Erfassung von Metadaten erfolgt oft manuell und ist fehleranfällig.

Technologie kann diesen Prozess unterstützen und einen Großteil der fehleranfälligen Arbeit der manuellen Erfassung von Metadaten übernehmen. Es gibt immer mehr Datenkataloge, Systeme zur Nachverfolgung des Datenflusses und Tools zur Verwaltung von Metadaten. Diese Tools können Datenbanken nach Beziehungen durchsuchen und Datenpipelines überwachen, um zu verfolgen, woher die Daten kommen und wohin sie gehen. Ein manueller Ansatz mit geringem Detaillierungs-

2 Eryurek, *Data Governance*, 5.

grad basiert auf internen Bemühungen, bei denen verschiedene Interessengruppen die Metadatenerfassung innerhalb des Unternehmens durch Crowdsourcing unterstützen. Diese Datenmanagementtools werden in diesem Buch ausführlich behandelt, da sie einen Großteil des Data Engineering Lifecycle unterstützen.

Metadaten werden zu einem Nebenprodukt von Daten und Datenprozessen. Es bleiben jedoch wichtige Herausforderungen bestehen. Insbesondere mangelt es noch an Interoperabilität und Standards. Metadatentools sind nur so gut wie ihre Verbindungen zu Datensystemen und ihre Fähigkeit, Metadaten auszutauschen. Darüber hinaus sollten automatisierte Metadatentools den Menschen nicht völlig aus dem Verkehr ziehen.

Daten haben ein soziales Element; jede Organisation sammelt soziales Kapital und Wissen um Prozesse, Datensätze und Pipelines. Menschenorientierte Metadatensysteme konzentrieren sich auf den sozialen Aspekt von Metadaten. Dies ist etwas, das Airbnb in seinen verschiedenen Blogbeiträgen zu Datentools betont hat, insbesondere in seinem ursprünglichen Dataportal-Konzept.[3] Solche Tools sollten einen Ort bieten, an dem Dateneigentümer, Datenkonsumenten und Fachexperten offengelegt werden können. Dokumentations- und interne Wiki-Tools bilden eine wichtige Grundlage für die Verwaltung von Metadaten, aber diese Tools sollten auch in die automatische Datenkatalogisierung integriert werden. So können beispielsweise Tools zum Scannen von Daten Wiki-Seiten mit Links zu relevanten Datenobjekten erstellen.

Sobald Metadatensysteme und -prozesse vorhanden sind, können Data Engineers Metadaten sinnvoll nutzen. Metadaten bilden die Grundlage für die Entwicklung von Pipelines und die Verwaltung von Daten während des gesamten Lebenszyklus.

DMBOK unterscheidet vier Arten von Metadaten, die für Data Engineers wichtig sind:

- geschäftliche Metadaten
- technische Metadaten
- operative Metadaten
- Referenz-Metadaten

Wir werden jede dieser Kategorien kurz vorstellen.

Geschäftliche Metadaten beziehen sich auf die Art und Weise, wie Daten im Unternehmen verwendet werden, mit eingeschlossen Themen wie Geschäfts- und Datendefinitionen, Datenregeln und -logik, den/die Dateneigentümer und wie und wo die Daten verwendet werden.

Ein Data Engineer verwendet geschäftliche Metadaten, um nicht technische Fragen zum Wer, Was, Wo und Wie zu beantworten. Ein Data Engineer kann beispielsweise damit beauftragt werden, eine Datenpipeline für die Analyse des Kundenum-

3 Chris Williams et al., »Democratizing Data at Airbnb«, *The Airbnb Tech Blog*, 12. Mai 2017, *https://oreil.ly/dM332*.

satzes zu erstellen. Aber was ist ein Kunde? Ist es jemand, der in den letzten 90 Tagen eingekauft hat? Oder jemand, der zu einem beliebigen Zeitpunkt eingekauft hat? Ein Data Engineer würde die entsprechenden Daten verwenden, um in den geschäftlichen Metadaten (Datenwörterbuch oder Datenkatalog) nachzuschlagen, wie ein »Kunde« definiert ist. Geschäftliche Metadaten liefern einem Data Engineer den geeigneten Kontext und die passenden Definitionen, um Daten ordnungsgemäß zu verwenden.

Technische Metadaten beschreiben die Daten, die von Systemen während des gesamten Data Engineering Lifecycle erstellt und verwendet werden. Sie umfassen Datenmodell und -schema, die Datenabfolge, Feldzuordnungen und Pipeline-Workflows. Ein Data Engineer nutzt technische Metadaten, um verschiedene Systeme über den gesamten Data Engineering Lifecycle hinweg zu erstellen, zu verbinden und zu überwachen.

Im Folgenden werden einige gängige Arten von technischen Metadaten aufgeführt, die ein Data Engineer verwendet:

- Pipeline-Metadaten (werden oft in Orchestrierungssystemen erzeugt)
- Datenherkunft
- Schema

Die Orchestrierung ist ein zentraler Dreh- und Angelpunkt, der die Arbeitsabläufe über verschiedene Systeme hinweg koordiniert. Die in Orchestrierungssystemen erfassten *Pipeline-Metadaten* enthalten Details zum Ablaufplan, zu System- und Datenabhängigkeiten, Konfigurationen, Verbindungsdetails und vielem mehr.

Metadaten zur Datenhistorie verfolgen den Ursprung und die Änderungen von Daten und deren Abhängigkeiten im Laufe der Zeit. Während die Daten den Data Engineering Lifecycle durchlaufen, entwickeln sie sich durch Transformationen und Kombinationen mit anderen Daten weiter. Die Datenhistorie bietet einen Prüfpfad für die Entwicklung der Daten, während diese verschiedene Systeme und Arbeitsabläufe durchlaufen.

Schema-Metadaten beschreiben die Struktur der in einem System wie einer Datenbank, einem Data Warehouse, einem Data Lake oder einem Dateisystem gespeicherten Daten; sie sind eines der wichtigsten Unterscheidungsmerkmale zwischen verschiedenen Speichersystemen. Objektspeicher zum Beispiel verwalten keine Schema-Metadaten; diese müssen stattdessen in einem *Metastore* verwaltet werden. Cloud Data Warehouses hingegen verwalten Schema-Metadaten intern.

Das sind nur einige Beispiele für technische Metadaten, die ein Data Engineer kennen sollte. Diese Aufzählung ist nicht vollständig, und wir behandeln im Laufe des Buchs weitere Beispiele für technische Metadaten.

Operative Metadaten geben Aufschluss über die Betriebsergebnisse verschiedener Systeme und umfassen Statistiken über Prozesse, Job-IDs, Laufzeitlogs, in einem Prozess verwendete Daten und Fehlerprotokolle. Ein Data Engineer verwendet operative Metadaten, um festzustellen, ob ein Prozess erfolgreich war oder fehlgeschlagen ist und welche Daten an dem Prozess beteiligt waren.

Orchestrierungssysteme können einen begrenzten Überblick über operative Metadaten geben, aber Letztere sind in der Regel immer noch über viele Systeme verstreut. Der Bedarf an qualitativ besseren operativen Metadaten und einer besseren Metadatenverwaltung ist eine wichtige Triebkraft für Orchestrierungs- und Metadatenverwaltungssysteme der nächsten Generation.

Referenz-Metadaten sind Daten, die zur Bestimmung von anderen Daten verwendet werden. Sie werden auch als *Nachschlagewerk* bezeichnet. Typische Beispiele für Referenzdaten sind interne und geografische Codes, Maßeinheiten und interne Kalenderstandards. Beachten Sie, dass ein Großteil der Referenzdaten vollständig intern verwaltet wird, aber Elemente wie geografische Codes können von externen Standardreferenzen stammen. Referenzdaten sind im Wesentlichen ein Standard für die Interpretation anderer Daten; wenn sie sich also ändern, geschieht dies langsam und im Laufe der Zeit.

Verantwortlichkeit. *Verantwortlichkeit* bedeutet, dass eine Person mit der Verwaltung eines Teils der Daten betraut wird. Die verantwortliche Person koordiniert dann die Governance-Aktivitäten der anderen Beteiligten. Die Verwaltung der Datenqualität ist schwierig, wenn niemand für die fraglichen Daten verantwortlich ist.

Beachten Sie, dass die für die Daten verantwortlichen Personen nicht zwangsläufig Data Engineers sein müssen. Die verantwortliche Person kann ein Softwareentwickler oder eine Produktmanagerin sein oder eine andere Funktion innehaben. Darüber hinaus verfügt die verantwortliche Person in der Regel nicht über alle erforderlichen Mittel zur Aufrechterhaltung der Datenqualität. Stattdessen stimmt sie sich mit allen Personen ab, die mit den Daten zu tun haben, einschließlich der Data Engineers.

Die Verantwortlichkeit für Daten kann auf verschiedenen Ebenen liegen; die Verantwortlichkeit kann sich auf der Ebene einer Tabelle oder eines Protokollstroms befinden, sie kann aber auch so feinkörnig sein wie ein einzelnes Feld, das in vielen Tabellen vorkommt. Eine Person kann für die Verwaltung einer Kunden-ID in mehreren Systemen verantwortlich sein. Für die Verwaltung von Unternehmensdaten ist eine Datendomäne die Menge aller möglichen Werte, die für einen bestimmten Feldtyp – wie in diesem Beispiel die ID – auftreten können. Dies mag übermäßig bürokratisch und penibel erscheinen, kann aber einen erheblichen Einfluss auf die Datenqualität haben.

Datenqualität.

> Can I trust this data?
>
> *– Jeder in der Branche*

Datenqualität ist die Optimierung von Daten hinsichtlich des angestrebten Zustands und kreist um die Frage: »Erhalte ich, was ich erwarte?« Die Daten sollten mit den Erwartungen in den geschäftlichen Metadaten übereinstimmen. Stimmen die Daten mit der vom Unternehmen vereinbarten Definition überein?

Ein Data Engineer gewährleistet die Datenqualität während des gesamten Data Engineering Lifecycle. Dazu gehört die Durchführung von Qualitätsprüfungen und die Sicherstellung der Konformität der Daten mit den Erwartungen an das Schema, die Vollständigkeit der Daten und der Präzision.

Laut *Data Governance: The Definitive Guide* wird die Datenqualität anhand von drei Merkmalen definiert:[4]

Richtigkeit
: Sind die erfassten Daten sachlich richtig? Gibt es doppelte Werte? Sind die numerischen Werte korrekt?

Vollständigkeit
: Sind die Datensätze vollständig? Enthalten alle erforderlichen Felder gültige Werte?

Aktualität
: Sind die Daten zeitnah verfügbar?

Jedes dieser Merkmale ist recht differenziert. Wie gehen wir zum Beispiel mit Bots und Web Scrapers um, wenn wir mit Web-Ereignisdaten arbeiten? Wollen wir die Customer Journey analysieren, brauchen wir einen Prozess, der es uns ermöglicht, menschlichen von maschinell erzeugtem Zugriff zu trennen. Alle von Bots erzeugten Ereignisse, die fälschlicherweise als *menschlich* klassifiziert werden, stellen ein Problem für die Datengenauigkeit dar und umgekehrt.

Es ergibt sich eine Reihe interessanter Aspekte in Bezug auf Vollständigkeit und Aktualität. In dem von Google veröffentlichten Artikel, in dem das Dataflow-Modell vorgestellt wird, beschreiben die Autoren das Beispiel einer Offlinevideoplattform, die Werbung anzeigt.[5] Bei bestehender Verbindung lädt die Plattform Videos und Werbeanzeigen herunter, ermöglicht dem Nutzer, diese offline anzusehen, und lädt dann die Daten über die Ansicht der Anzeige hoch, sobald wieder eine Verbindung besteht. Diese Daten können spät eintreffen, lange nachdem die Werbung angesehen wurde. Wie wickelt die Plattform die Abrechnung der Werbung ab?

Grundsätzlich kann dieses Problem nicht mit rein technischen Mitteln gelöst werden. Vielmehr müssen die Entwickler ihre Vorgaben für verspätet eintreffende Daten festlegen und diese einheitlich durchsetzen, möglicherweise mithilfe diverser Hilfsmittel.

4 Eryurek, *Data Governance*, 113.

5 Tyler Akidau et al., »The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing«, *Proceedings of the VLDB Endowment* 8 (2015): 1792–1803, *https://oreil.ly/Z6XYy*.

Management von Stammdaten

Stammdaten sind Daten über Geschäftseinheiten wie Belegschaft, Kunden, Produkte und Standorte. Da Unternehmen durch organisches Wachstum und Übernahmen größer und komplexer werden und mit anderen Unternehmen zusammenarbeiten, wird es immer schwieriger, ein konsistentes Bild von Einheiten und Identitäten zu erhalten.

Unter *Master Data Management* (MDM, Stammdatenmanagement) versteht man die Erstellung konsistenter Entitätsdefinitionen, die als *Golden Records* bekannt sind. Golden Records harmonisieren Entitätsdaten innerhalb eines Unternehmens und mit seinen Partnern. MDM ist ein Geschäftsprozess, der durch die Entwicklung und Bereitstellung von Technologiewerkzeugen erleichtert wird. Ein MDM-Team könnte beispielsweise ein Standardformat für Adressen festlegen und dann mit Data Engineers zusammenarbeiten, um eine API zu entwickeln, die konsistente Adressen zurückgibt, sowie ein System, das Adressdaten verwendet, um Kundendatensätze über Unternehmensbereiche hinweg abzugleichen.

MDM erstreckt sich über den gesamten Datenzyklus bis hin zu den operativen Datenbanken. Es kann direkt in den Zuständigkeitsbereich von Data Engineering fallen, ist aber oft die Aufgabe eines speziellen Teams, das unternehmensweit arbeitet. Auch wenn sie nicht für MDM zuständig sind, müssen Data Engineers stets damit vertraut sein, da sie möglicherweise an MDM-Initiativen mitarbeiten.

Die Datenqualität liegt im Grenzbereich zwischen menschlichen und technischen Problemen. Data Engineers benötigen robuste Prozesse, um verwertbares menschliches Feedback zur Datenqualität zu sammeln und technische Werkzeuge zu nutzen, um Qualitätsprobleme präventiv zu erkennen, bevor die Anwender sie überhaupt bemerken. Wir behandeln diese Erfassungsprozesse in den entsprechenden Kapiteln dieses Buchs.

Datenmodellierung und -design

Um mithilfe von Business Analytics und Data Science aus Daten Geschäftswissen abzuleiten, müssen die Daten in einer verwertbaren Form vorliegen. Der Prozess der Transformation von Daten in eine verwertbare Form wird als *Datenmodellierung und -design* bezeichnet. Wenngleich wir bei Datenmodellierung traditionell an ein Problem für Datenbankadministratoren (DBAs) und ETL-Entwickler denken, kann Datenmodellierung fast überall in einem Unternehmen stattfinden. Firmware-Engineers entwickeln das Datenformat eines Datensatzes für ein IoT-Gerät, oder Webanwendungsentwicklerinnen entwerfen die JSON-Antwort auf einen API-Aufruf oder ein MySQL-Tabellenschema – all dies sind Beispiele für Datenmodellierung und -design.

Die Datenmodellierung ist aufgrund der Vielzahl neuer Datenquellen und Anwendungsfälle deutlich komplexer geworden. Beispielsweise funktioniert eine strenge

Normalisierung nicht gut mit Ereignisdaten. Dank neuartiger Datentools wird die Flexibilität von Datenmodellen erhöht, während die logische Trennung von Kennzahlen, Dimensionen, Attributen und Hierarchien erhalten bleibt. Cloud Data Warehouses unterstützen die Aufnahme enormer Mengen an denormalisierten und semistrukturierten Daten, während gleichzeitig gängige Datenmodellierungsmuster wie Kimball, Inmon und Data Vault unterstützt werden. Datenverarbeitungsframeworks wie Spark können ein ganzes Spektrum von Daten aufnehmen, von flach strukturierten relationalen Datensätzen bis hin zu unstrukturiertem Text. Diese Datenmodellierungs- und -umwandlungsmuster werden in Kapitel 8 ausführlicher behandelt.

Angesichts der großen Vielfalt an Daten, mit denen Data Engineers umgehen müssen, ist die Versuchung groß, die Hände in den Schoß zu legen und die Datenmodellierung aufzugeben. Das jedoch hätte so fatale Folgen wie das WORN-Zugriffsmuster (*Write Once, Read Never*) oder der *Datensumpf*. Data Engineers müssen die Best Practices der Modellierung verstehen und die Flexibilität entwickeln, um die geeignete Ebene und Art der Modellierung für die Datenquelle und den Anwendungsfall auszuwählen.

Datenherkunft

Woran erkennt man, welches System die Daten im Laufe ihres Lebenszyklus beeinflusst hat oder woraus die Daten bestehen, während sie weitergegeben und umgewandelt werden? Datenherkunft bezeichnet die Erfassung eines Audit-Trails über den Lebenszyklus von Daten, wobei sowohl die Systeme, die die Daten verarbeiten, als auch die zugrunde liegenden Daten verfolgt werden.

Die Datenhistorie hilft bei der Fehlerverfolgung, der Rechenschaftspflicht und der Fehlersuche bei Daten und den Systemen, die sie verarbeiten. Sie hat den offensichtlichen Vorteil, dass sie einen Audit-Trail für den Lebenszyklus der Daten bietet und bei der Einhaltung von Vorschriften hilft. Wenn beispielsweise ein Benutzer seine Daten aus Ihren Systemen löschen lassen möchte, können Sie mithilfe der Datenherkunft feststellen, wo diese Daten gespeichert sind und welche Abhängigkeiten sie haben.

Datenherkunft gibt es in größeren Unternehmen mit strengen Compliance-Vorgaben schon seit Langem. Mit der zunehmenden Verbreitung des Datenmanagements in kleineren Unternehmen wird dieses Instrument jedoch immer häufiger eingesetzt. Wir stellen außerdem fest, dass Andy Petrellas Konzept des Data Observability Driven Development (DODD) (*https://oreil.ly/3f4WS*) eng mit der Datenherkunft verwandt ist. Bei DODD werden die Daten entlang ihres gesamten Verlaufs beobachtet. Dieser Prozess wird bei der Entwicklung, beim Testen und schließlich bei der Produktion angewandt, um Qualität und Konformität mit den Anforderungen zu gewährleisten.

Datenintegration und Interoperabilität

Datenintegration und Interoperabilität ist der Prozess der Integration von Daten über Tools und Prozesse hinweg. Da wir uns von einem Single-Stack-Ansatz für Analysen wegbewegen und zu einer heterogenen Cloud-Umgebung übergehen, in der verschiedene Tools Daten nach Bedarf verarbeiten, nehmen Integration und Interoperabilität einen immer größeren Teil der Arbeit des Data Engineers ein.

Die Integration erfolgt zunehmend über Allzweck-APIs und nicht mehr über benutzerdefinierte Datenbankverbindungen. Eine Datenpipeline könnte beispielsweise Daten von der Salesforce-API abrufen, sie in Amazon S3 speichern, die Snowflake-API aufrufen, um sie in eine Tabelle zu laden, die API erneut aufrufen, um eine Abfrage auszuführen, und dann die Ergebnisse in S3 exportieren, wo sie von Spark verwendet werden können.

All diese Aktivitäten können mit relativ einfachem Python-Code verwaltet werden, der mit Datensystemen kommuniziert, anstatt Daten direkt zu verarbeiten. Während die Komplexität der Interaktion mit Datensystemen abgenommen hat, sind die Anzahl der Systeme und die Komplexität der Pipelines erheblich gestiegen. Entwicklerinnen und Entwickler, die mit dem Programmieren beginnen, wachsen schnell über die Möglichkeiten der maßgeschneiderten Skripte hinaus und erkennen die Notwendigkeit der *Orchestrierung*. Orchestrierung ist eine unserer Unterströmungen, die wir in »Orchestrierung« auf Seite 98 ausführlich erörtern.

Management des Datenlebenszyklus

Das Aufkommen von Data Lakes ermutigte Unternehmen, die Archivierung und Löschung von Daten zu ignorieren. Warum sollte man Daten wegwerfen, wenn man einfach unbegrenzt mehr Speicherplatz hinzufügen kann? Zwei Veränderungen haben jedoch dazu geführt, dem, was am Ende des Data Engineering Lifecycle geschieht, mehr Aufmerksamkeit zu schenken.

Erstens werden Daten zunehmend in der Cloud gespeichert. Das bedeutet, dass die Speicherkosten nach tatsächlicher Nutzung abgerechnet werden, anstatt hohe Vorabinvestitionen für einen lokalen Datenspeicher zu tätigen. Wenn jedes Byte auf der monatlichen AWS-Abrechnung auftaucht, sehen CFOs Einsparmöglichkeiten. Cloud-Umgebungen machen die Datenarchivierung zu einem relativ unkomplizierten Prozess. Die großen Cloud-Anbieter bieten archivierungsspezifische Objektspeicherklassen an, die eine langfristige Datenaufbewahrung zu extrem niedrigen Kosten ermöglichen, sofern nur sehr selten auf die Daten zugegriffen wird (es sei darauf hingewiesen, dass das Abrufen von Daten nicht so billig ist, aber das ist ein anderes Thema). Diese Speicherklassen unterstützen auch zusätzliche Richtlinienkontrollen, um ein versehentliches oder absichtliches Löschen von wichtigen Archiven zu verhindern.

Zweitens müssen Data Engineers aufgrund von Gesetzen zum Schutz der Privatsphäre und zur Vorratsdatenspeicherung wie der DSGVO und dem CCPA die Da-

tenvernichtung aktiv verwalten, um das *Recht auf Vergessenwerden* der Nutzer zu respektieren. Data Engineers müssen wissen, welche Verbraucherdaten sie aufbewahren, und sie müssen über Verfahren verfügen, um Daten als Reaktion auf Anfragen und Compliance-Anforderungen zu vernichten.

Die Datenvernichtung ist in einem Cloud Data Warehouse einfach. Die SQL-Semantik erlaubt die Löschung von Zeilen, die mit einer `Where`-Bedingung übereinstimmen. In Data Lakes war die Datenvernichtung schwieriger, da hier das Standardspeicherverfahren *Write Once, Read Many* galt. Tools wie Hive ACID und Delta Lake ermöglichen eine einfache Verwaltung von Löschtransaktionen in großem Umfang. Mit den neuen Tools für die Verwaltung von Metadaten, die Datenabfolge und die Katalogisierung wird auch das Ende des Data Engineering Lifecycle optimiert.

Ethik und Datenschutz

Die Datenschutzverletzungen, Fehlinformationen und der unsachgemäße Umgang mit Daten in den letzten Jahren haben eines deutlich gemacht: Daten beeinflussen die Menschen. Galten ethische und datenschutzrechtliche Implikationen von Daten früher als »nice to have«, wie z. B. Sicherheit, so stehen sie heute im Mittelpunkt des gesamten Lebenszyklus von Daten. Data Engineers müssen das Richtige tun, auch wenn niemand zuschaut, denn eines Tages werden alle zuschauen.[6] Wir hoffen, dass mehr Organisationen eine Kultur der Datenethik und des Datenschutzes fördern werden.

Wie wirken sich Ethik und Datenschutz auf den Data Engineering Lifecycle aus? Data Engineers müssen sicherstellen, dass Datensätze personenbezogene Daten und andere sensible Informationen verbergen; Bias kann in Datensätzen erkannt und nachverfolgt werden. Die gesetzlichen Anforderungen und die Strafen für die Nichteinhaltung von Vorschriften nehmen ständig zu. Stellen Sie sicher, dass Ihre Datenbestände mit einer wachsenden Zahl von Datenvorschriften wie GDPR und CCPA konform sind. Bitte nehmen Sie dies ernst. Wir geben Ihnen in diesem Buch Tipps, wie Sie Ethik und Datenschutz in den Data Engineering Lifecycle integrieren können.

DataOps

DataOps überträgt die besten Praktiken aus der agilen Methodik, DevOps und der statistischen Prozesskontrolle (SPC) auf Daten. Während DevOps darauf abzielt, die Veröffentlichung und Qualität von Softwareprodukten zu verbessern, tut DataOps dasselbe für Datenprodukte.

6 Wir vertreten die Auffassung, dass ethisches Verhalten darin besteht, das Richtige zu tun, auch wenn niemand zuschaut, eine Idee, die in den Schriften von C. S. Lewis, Charles Marshall und vielen anderen Autoren auftaucht.

Datenprodukte unterscheiden sich von Softwareprodukten durch die Art und Weise, wie die Daten verwendet werden. Ein Softwareprodukt bietet spezifische Funktionen und technische Merkmale für Endbenutzer. Im Gegensatz dazu basiert ein Datenprodukt auf solider Geschäftslogik und Metriken, deren Benutzer Entscheidungen treffen oder Modelle erstellen, die automatisierte Aktionen durchführen. Ein Data Engineer muss sowohl die technischen Aspekte der Erstellung von Softwareprodukten als auch die Geschäftslogik, die Qualität und die Metriken verstehen, die zu hervorragenden Datenprodukten führen.

Wie DevOps lehnt sich auch DataOps an Lean Manufacturing und Supply Chain Management an und kombiniert Menschen, Prozesse und Technologie, um die Dauer bis zur Wertschöpfung zu verkürzen. Data Kitchen (Experten für DataOps) beschreibt es folgendermaßen:[7]

> DataOps ist eine Sammlung von technischen Praktiken, Arbeitsabläufen, kulturellen Normen und Architekturmustern, die Folgendes ermöglichen:
>
> - Innovation und Forschung, die den Kunden immer schneller neue Erkenntnisse liefern.
> - Äußerst hohe Datenqualität und sehr niedrige Fehlerquoten.
> - Zusammenarbeit über komplexe Bereiche von Menschen, Technologien und Bereiche hinweg.
> - Eindeutige Messung, Überwachung und Transparenz der Ergebnisse.

Schlanke Prozesse (z. B. Verkürzung der Durchlaufzeit und Minimierung von Fehlern) und die daraus resultierenden Qualitäts- und Produktivitätsverbesserungen sind etwas, das wir sowohl im Software- als auch im Datenbereich mit Freude beobachten.

Bei DataOps handelt es sich in erster Linie um eine Reihe kultureller Gewohnheiten. Das Data-Engineering-Team muss einen Zyklus der Kommunikation und Zusammenarbeit mit dem Unternehmen, des Überwindens von Insellösungen, des kontinuierlichen Lernens aus Erfolgen und Fehlern und der schnellen Iteration einführen. Nur wenn diese kulturellen Gewohnheiten etabliert sind, kann das Team die besten Ergebnisse mithilfe von Technologie und Tools erzielen.

Abhängig vom Reifegrad der Daten eines Unternehmens hat ein Data Engineer mehrere Möglichkeiten, DataOps in den Data Engineering Lifecycle zu integrieren. Besitzt das Unternehmen noch keine Dateninfrastruktur oder weist keine Datenpraktiken auf, ist DataOps eine neue Möglichkeit, die vom ersten Tag an integriert werden kann. Bei einem bestehenden Projekt oder einer Infrastruktur ohne DataOps kann ein Data Engineer damit beginnen, DataOps in die Arbeitsabläufe zu integrieren. Wir empfehlen, zunächst mit Beobachtbarkeit und Überwachung anzufangen, um einen Einblick in die Systemleistung zu erhalten, und dann Automatisierung und Reaktion auf Vorfälle hinzuzufügen. Ein Data Engineer kann mit einem bestehenden DataOps-Team zusammenarbeiten, um den Data Engineering Lifecycle in einem daten-

7 »What Is DataOps«, DataKitchen-FAQ-Seite, abgerufen am 5. Mai 2022, *https://oreil.ly/Ns06w*.

geführten Unternehmen zu verbessern. In jedem Fall muss ein Data Engineer mit der Philosophie und den technischen Aspekten von DataOps vertraut sein.

DataOps besteht aus drei Säulen: Automatisierung, Überwachung und Beobachtbarkeit sowie Reaktion auf Störfälle (siehe Abbildung 2-8). Im Folgenden werden die einzelnen Säulen und ihr Bezug zum Data Engineering Lifecycle erläutert.

Abbildung 2-8: Die drei Säulen von DataOps

Automatisierung

Automatisierung sorgt für Zuverlässigkeit und Konsistenz im DataOps-Prozess und ermöglicht den Data Engineers, neue Produktfunktionen und Verbesserungen an bestehenden Workflows schnell zu implementieren. Die DataOps-Automatisierung hat einen ähnlichen Rahmen und Arbeitsablauf wie DevOps, bestehend aus Änderungsmanagement (Umgebungs-, Code- und Datenversionskontrolle), kontinuierlicher Integration/kontinuierlicher Bereitstellung (CI/CD) und Konfiguration als Code. Wie DevOps überwachen und erhalten DataOps-Methoden die Zuverlässigkeit von Technologien und Systemen (Datenpipelines, Orchestrierung usw.), wobei zusätzlich die Datenqualität, die Daten-/Modellabweichung, die Integrität der Metadaten usw. überprüft werden.

Betrachten wir kurz die Entwicklung der DataOps-Automatisierung in einem hypothetischen Unternehmen. Ein Unternehmen mit einem geringen DataOps-Reifegrad versucht oft, mehrere Phasen von Datentransformationen mithilfe von Cron-Jobs zu planen. Dies funktioniert eine Zeit lang gut. Sobald die Datenpipelines jedoch komplizierter werden, kann es zu einer Reihe von Problemen kommen. Werden die Cron-Jobs auf einer Cloud-Instanz gehostet, könnte die Instanz eine Betriebsstörung haben, die dazu führt, dass die Aufträge nicht mehr ausgeführt werden. Da die Abstände zwischen den Aufträgen immer geringer werden, wird ein Auftrag schließlich zu lange laufen, was dazu führt, dass ein nachfolgender Auftrag fehlschlägt oder veraltete Daten produziert. Möglicherweise bemerken Entwickler die Auftragsausfälle erst, wenn sie von Analystinnen erfahren, dass ihre Berichte veraltet sind.

Mit zunehmender Datenreife des Unternehmens werden die Data Engineers in der Regel ein Orchestrierungsframework einführen, vielleicht Airflow oder Dagster. Data Engineers sind sich bewusst, dass Airflow eine operative Belastung darstellt, aber die Vorteile der Orchestrierung überwiegen schließlich die Komplexität. Die Entwickler werden ihre Cron-Jobs nach und nach auf Airflow-Jobs umstellen. Jetzt werden die Abhängigkeiten geprüft, bevor die Aufträge ausgeführt werden. Es können mehr Transformationsaufträge in eine bestimmte Zeit gepackt werden, da jeder Auftrag gestartet werden kann, sobald die Upstream-Daten bereit sind, anstatt zu einem festen, vorher festgelegten Zeitpunkt.

Das Data-Engineering-Team hat noch Raum für operative Verbesserungen. Ein Data Scientist setzt irgendwann einen defekten DAG (*Directed Acyclic Graph*, gerichteter azyklischer Graph) ein, was den Airflow-Webserver zum Absturz bringt und das Datenteam betriebsblind zurücklässt. Nach mehreren solcher Situationen erkennen die Mitglieder des Data-Engineering-Teams, dass sie keine manuellen DAG-Bereitstellungen zulassen dürfen. In der nächsten Phase der betrieblichen Reife führen sie die automatische DAG-Bereitstellung ein. Die DAGs werden vor der Bereitstellung getestet, und Überwachungsprozesse stellen sicher, dass die neuen DAGs ordnungsgemäß in Betrieb genommen werden. Darüber hinaus blockieren die Data Engineers die Bereitstellung neuer Python-Abhängigkeiten, bis die Installation validiert ist. Nachdem die Automatisierung eingeführt wurde, ist das Datenteam viel zufriedener und hat weniger Probleme.

Einer der Grundsätze des DataOps-Manifests (*https://oreil.ly/2LGwL*) lautet »Mut zur Veränderung«. Damit ist nicht die Veränderung um der Veränderung willen gemeint, sondern ein zielgerichteter Wandel. In jeder Phase unserer Automatisierungsreise gibt es Möglichkeiten zur betrieblichen Verbesserung. Selbst bei dem hohen Reifegrad, den wir hier beschrieben haben, bleibt noch Raum für weitere Verbesserungen. Data Engineers könnten ein neues Orchestrierungsframework einführen, das bessere Metadatenfunktionen enthält. Oder sie könnten versuchen, ein Framework zu entwickeln, das DAGs automatisch auf der Grundlage von Vorgaben für die Datenherkunft erstellt. Der wichtigste Punkt ist, dass Entwickler ständig versuchen, Verbesserungen in der Automatisierung zu implementieren, die ihre Arbeitslast verringern und den Wert, den sie dem Unternehmen liefern, erhöhen.

Beobachtbarkeit und Überwachung

Wir sagen unseren Kunden: »Daten sind lautlose Killer.« Wir haben zahllose Beispiele für schlechte Daten gesehen, die über Monate oder Jahre in Berichten schlummerten. Führungskräfte treffen möglicherweise wichtige Entscheidungen auf der Grundlage dieser schlechten Daten und entdecken den Fehler erst viel später. Die Konsequenzen sind in der Regel schwerwiegend und manchmal katastrophal für das Unternehmen. Initiativen werden unterminiert und zerstört, jahrelange Arbeit wird vergeudet. In einigen der schlimmsten Fälle können schlechte Daten Unternehmen in den finanziellen Ruin treiben.

Ein weiteres Schreckensszenario ist, dass die Systeme, mit denen die Daten für die Berichte erstellt werden, plötzlich nicht mehr funktionieren, sodass sich die Berichte um mehrere Tage verzögern. Das Datenteam weiß erst Bescheid, wenn es von den Beteiligten gefragt wird, warum die Berichte zu spät kommen oder veraltete Informationen liefern. Schließlich verlieren verschiedene Interessengruppen das Vertrauen in die Fähigkeiten des zentralen Datenteams und gründen ihre eigenen Splitterteams. Das Ergebnis sind viele verschiedene instabile Systeme, inkonsistente Berichte und Insellösungen.

Wenn Sie Ihre Daten und die Systeme, die die Daten produzieren, nicht beobachten und überwachen, werden Sie unweigerlich Ihre eigene Daten-Gruselgeschichte erleben. Beobachtbarkeit, Überwachung, Protokollierung, Alarmierung und Rückverfolgung sind entscheidend, um Problemen im Data Engineering Lifecycle zuvorzukommen. Wir empfehlen Ihnen, SPC einzubinden, um zu verstehen, ob die überwachten Ereignisse aus dem Rahmen fallen und auf welche Vorfälle Sie reagieren sollten.

Die bereits in diesem Kapitel erwähnte DODD-Methode von Petrella bietet einen ausgezeichneten Rahmen, um über die Beobachtbarkeit von Daten nachzudenken. DODD ist vergleichbar mit der testgetriebenen Entwicklung (Test-Driven Development, TDD) in der Softwareentwicklung:[8]

> Ziel von DODD ist es, allen an der Datenkette Beteiligten einen Einblick in die Daten und Datenanwendungen zu geben, sodass jeder an der Datenwertschöpfungskette Beteiligte in der Lage ist, Änderungen an den Daten oder Datenanwendungen bei jedem Schritt – von der Ingestion über die Transformation bis hin zur Analyse – zu erkennen, um Datenprobleme zu beheben oder zu vermeiden. DODD konzentriert sich darauf, die Beobachtbarkeit von Daten zu einem erstklassigen Faktor im Data Engineering Lifecycle zu machen.

Viele Aspekte der Überwachung und Beobachtbarkeit während des gesamten Data Engineering Lifecycle werden in nachfolgenden Kapiteln behandelt.

Reaktion auf Störfälle

Ein gut funktionierendes Datenteam, das DataOps einsetzt, kann schnell neue Datenprodukte liefern. Aber Störfälle sind unvermeidlich. Ein System kann ausfallen, ein neues Datenmodell kann nachgelagerte Berichte beschädigen, ein ML-Modell kann veraltet sein und schlechte Vorhersagen liefern – unzählige Probleme können den Data Engineering Lifecycle unterbrechen. Bei der *Reaktion auf Störfälle* geht es darum, die bereits erwähnten Automatisierungs- und Beobachtungsfunktionen zu nutzen, um die Ursachen einer Störung schnellstmöglich zu ermitteln und sie so zuverlässig und zeitnah wie möglich zu beheben.

Bei der Reaktion auf Vorfälle geht es nicht nur um Technologie und Tools – auch wenn diese von Vorteil sind –, es geht auch um eine offene und tadellose Kommunikation sowohl im Data-Engineering-Team als auch im gesamten Unternehmen. Wie Werner Vogels, CTO von Amazon Web Services, zu sagen pflegt: »Alles geht ständig kaputt.« Data Engineers müssen auf eine Katastrophe vorbereitet und bereit sein, so schnell und effizient wie möglich zu reagieren.

Data Engineers sollten proaktiv nach Problemen suchen, bevor das Unternehmen sie meldet. Fehler passieren, und wenn Stakeholder oder Endnutzer Probleme sehen, werden sie diese melden. Sie werden darüber nicht erfreut sein. Anders sieht

8 Andy Petrella, »Data Observability Driven Development: The Perfect Analogy for Beginners«, Kensu, abgerufen am 5. Mai 2022, *https://oreil.ly/MxvSX*.

es aus, wenn sie diese Probleme einem Team melden und sehen, dass bereits aktiv an einer Lösung gearbeitet wird. Welchem Team würden Sie als Nutzer mehr vertrauen? Es dauert lange, Vertrauen aufzubauen, und es kann innerhalb von Minuten verspielt werden. Bei der Reaktion auf Störfälle geht es sowohl um die nachträgliche Reaktion auf Vorfälle als auch um deren proaktive Behebung, noch bevor sie eintreten.

Zusammenfassung DataOps

Zum jetzigen Zeitpunkt befindet sich DataOps noch in der Entwicklung. Experten haben gute Arbeit geleistet, indem sie die DevOps-Prinzipien an den Datenbereich angepasst und mit dem DataOps-Manifest und anderen Ressourcen eine erste Vision entworfen haben. Data Engineers täten gut daran, DataOps-Methoden bei ihrer gesamten Arbeit eine hohe Priorität einzuräumen. Der anfängliche Aufwand wird sich langfristig durch eine schnellere Bereitstellung von Produkten, eine höhere Zuverlässigkeit und Genauigkeit der Daten und einen höheren Gesamtwert für das Unternehmen auszahlen.

Im Vergleich zur Softwareentwicklung sind die Abläufe im Data Engineering noch relativ unausgereift. Viele Data-Engineering-Tools, insbesondere ältere Lösungen, sind nicht automatisierbar. In jüngster Zeit hat sich eine Bewegung entwickelt, die bewährte Automatisierungsverfahren für den gesamten Data Engineering Lifecycle einführen will. Tools wie Airflow haben den Weg für eine neue Generation von Automatisierungs- und Datenmanagementtools geebnet. Die allgemeinen Verfahren, die wir für DataOps beschreiben, sind wünschenswert, und wir schlagen vor, dass Unternehmen versuchen, sie angesichts der Tools und des Wissens, die heute zur Verfügung stehen, so weit wie möglich zu übernehmen.

Datenarchitektur

Eine Datenarchitektur spiegelt den aktuellen und zukünftigen Zustand von Datensystemen wider, die den langfristigen Datenbedarf und die Strategie eines Unternehmens unterstützen. Da sich die Datenanforderungen eines Unternehmens wahrscheinlich schnell ändern werden und fast täglich neue Tools und Verfahren auf den Markt kommen, müssen Data Engineers gute Datenarchitekturen verstehen. Kapitel 3 befasst sich eingehend mit der Datenarchitektur, aber wir möchten an dieser Stelle hervorheben, dass die Datenarchitektur eine der Unterströmungen im Data Engineering Lifecycle ist.

Ein Data Engineer sollte zunächst die Bedürfnisse des Unternehmens verstehen und Anforderungen für neue Anwendungsfälle sammeln. Als Nächstes muss ein Data Engineer diese Anforderungen umsetzen, um neue Wege der Datenerfassung und -bereitstellung zu entwerfen, wobei Kosten und Benutzerfreundlichkeit berücksichtigt werden müssen. Dies bedeutet, dass er die Kompromisse bei Entwurfsmustern, Technologien und Tools für Quellsysteme sowie Ingestion, Speicherung, Transformation und Bereitstellung von Daten kennen muss.

Dies bedeutet nicht, dass ein Data Engineer auch ein Datenarchitekt ist, da es sich in der Regel um zwei verschiedene Rollen handelt. Wenn ein Data Engineer mit einem Datenarchitekten zusammenarbeitet, sollte der Data Engineer in der Lage sein, die Entwürfe des Datenarchitekten umzusetzen und Feedback zur Architektur zu geben.

Orchestrierung

> Wir sind der Meinung, dass die Orchestrierung wichtig ist, weil wir sie als den Dreh- und Angelpunkt sowohl der Datenplattform als auch des Datenlebenszyklus und des Lebenszyklus der Softwareentwicklung im Zusammenhang mit Daten betrachten.
>
> *– Nick Schrock, Gründer von Elementl*[9]

Die Orchestrierung ist nicht nur ein zentraler DataOps-Prozess, sondern auch ein entscheidender Teil des Entwicklungs- und Bereitstellungsablaufs für Datenaufträge. Was also ist Orchestrierung?

Unter *Orchestrierung* versteht man den Prozess der Koordinierung vieler Aufgaben, damit diese so schnell und effizient wie möglich nach einem festgelegten Zeitplan ausgeführt werden. So werden beispielsweise Orchestrierungswerkzeuge wie Apache Airflow häufig als *Scheduler* bezeichnet. Das ist nicht ganz richtig. Ein reiner Scheduler, wie z.B. Cron, achtet nur auf die Zeit, eine Orchestrierungs-Engine baut Metadaten zu den Jobabhängigkeiten ein, im Allgemeinen in Form eines gerichteten azyklischen Graphen (DAG). Der DAG kann einmalig oder in einem festen Intervall (täglich, wöchentlich, jede Stunde, alle fünf Minuten usw.) ausgeführt werden.

Wenn wir in diesem Buch über Orchestrierung sprechen, gehen wir davon aus, dass ein Orchestrierungssystem immer online ist. Dies ermöglicht dem Orchestrierungssystem, ohne menschliches Eingreifen ständig zu erfassen und zu überwachen sowie jederzeit neue Aufgaben auszuführen, wenn sie bereitgestellt werden. Ein Orchestrierungssystem überwacht die von ihm verwalteten Aufträge und stößt neue Aufgaben an, wenn die internen DAG-Abhängigkeiten abgeschlossen sind. Es kann auch externe Systeme und Tools überwachen, um zu sehen, ob Daten ankommen und Vorgaben erfüllt werden. Wenn bestimmte Konditionen nicht eingehalten werden, legt das System auch Fehlerbedingungen fest und sendet über E-Mail oder andere Kanäle Warnmeldungen. Beispielsweise kann für tägliche Datenpipelines, die über Nacht laufen, eine erwartete Fertigstellungszeit von 10 Uhr morgens festgelegt werden. Sind die Aufträge bis zu diesem Zeitpunkt nicht erledigt, werden Warnungen an die Data Engineers und Nutzer ausgegeben.

Außerdem bieten Orchestrierungen auch Funktionen für Auftragshistorie, Visualisierung und Alarmierung. Fortgeschrittene Orchestrierungs-Engines können neue

9 Ternary Data, »An Introduction to Dagster: The Orchestrator for the Full Data Lifecycle – UDEM June 2021«, YouTube-Video, 1:09:40, *https://oreil.ly/HyGMh*.

DAGs oder einzelne Aufgaben, die einem DAG hinzugefügt werden, mit Backfills versehen. Sie unterstützen auch Abhängigkeiten über einen bestimmten Zeitraum hinweg. So kann beispielsweise ein monatlicher Berichtsauftrag vor dem Ausführen prüfen, ob der ETL-Auftrag für den gesamten Monat bereits abgeschlossen wurde.

Die Orchestrierung ist seit Langem eine Schlüsselfunktion für die Datenverarbeitung, war aber außer in größten Unternehmen kaum ein Thema und auch nicht für jedermann zugänglich. Unternehmen nutzten verschiedene Tools zur Verwaltung von Auftragsflüssen, aber diese waren teuer, für kleine Start-ups unerschwinglich und im Allgemeinen nicht erweiterbar. Apache Oozie war in den 2010ern sehr beliebt, aber es wurde für die Arbeit innerhalb eines Hadoop-Clusters entwickelt und war in einer heterogeneren Umgebung schwer zu verwenden. Facebook entwickelte in den späten 2000ern Dataswarm für den internen Gebrauch; dies inspirierte beliebte Tools wie Airflow, das 2014 von Airbnb eingeführt wurde.

Airflow war von Anfang an quelloffen und fand weite Verbreitung. Es wurde in Python geschrieben, was es für fast jeden denkbaren Anwendungsfall erweiterbar macht. Es gibt zwar viele andere interessante Open-Source-Orchestrierungsprojekte wie Luigi und Conductor, aber Airflow ist derzeit wohl die führende Lösung. Airflow kam gerade zu dem Zeitpunkt auf den Markt, als die Datenverarbeitung abstrakter und zugänglicher wurde und Data Engineers zunehmend daran interessiert waren, komplexe Abläufe über mehrere Prozessoren und Speichersysteme hinweg zu koordinieren, insbesondere in Cloud-Umgebungen.

Gegenwärtig gibt es mehrere Open-Source-Projekte, die darauf abzielen, die besten Elemente des Airflow-Kerndesigns nachzuahmen und es gleichzeitig in wichtigen Bereichen zu verbessern. Einige der interessantesten Beispiele sind Prefect und Dagster, deren Ziel es ist, die Portabilität und Testbarkeit von DAGs zu verbessern, um Data Engineers den Übergang von der lokalen Entwicklung zur Produktion zu erleichtern. Argo ist eine Orchestrierungs-Engine, die auf Kubernetes-Primitiven aufbaut; Metaflow ist ein Open-Source-Projekt von Netflix zur Verbesserung der Orchestrierung von Data Science.

Wir müssen darauf hinweisen, dass die Orchestrierung ein reines Batch-Konzept ist. Die Streaming-Alternative zu orchestrierten Aufgaben-DAGs ist der Streaming-DAG. Streaming-DAGs sind nach wie vor schwierig zu erstellen und zu warten, aber neuere Streaming-Plattformen wie Pulsar sollen den technischen und betrieblichen Aufwand deutlich reduzieren. Mehr über diese Entwicklungen erfahren Sie in Kapitel 8.

Softwareentwicklung

Die Softwareentwicklung war schon immer eine Kernkompetenz von Data Engineers. In den Anfängen des heutigen Data Engineering (2000–2010) arbeiteten Data Engineers an systemnahen Frameworks und schrieben MapReduce-Aufgaben

in C, C++ und Java. Auf dem Höhepunkt der Big-Data-Ära (Mitte der 2010er-Jahre) begannen die Data Engineers mit der Verwendung von Frameworks, die von diesen systemnahen Details abstrahierten.

Diese Entwicklung setzt sich heute fort. Cloud Data Warehouses unterstützen leistungsfähige Transformationen unter Verwendung von SQL-Semantik; Tools wie Spark sind benutzerfreundlicher geworden, indem sie von einfachen Programmierdetails zu benutzerfreundlichen Dataframes übergegangen sind. Trotz dieser Entwicklung bleibt die Softwareentwicklung für das Data Engineering entscheidend. Wir möchten kurz einige allgemeine Bereiche des Software-Engineerings erörtern, die für den Data Engineering Lifecycle von Bedeutung sind.

Code für die Kerndatenverarbeitung

Auch wenn die Datenverarbeitung abstrakter und einfacher zu handhaben ist, muss immer noch Kerncode geschrieben werden, und zwar während des gesamten Data Engineering Lifecycle. Ob bei der Dateningestion, -transformation oder -bereitstellung, Data Engineers müssen Frameworks und Sprachen wie Spark, SQL oder Beam sehr gut beherrschen und produktiv einsetzen können; wir weisen die Vorstellung zurück, dass SQL kein Code ist.

Es ist auch unerlässlich, dass ein Data Engineer die richtigen Testverfahren versteht, wie Unit, Regression, Integration, End-to-End und Smoke.

Entwicklung von Open-Source-Frameworks

Viele Data Engineers sind stark an der Entwicklung von Open-Source-Frameworks beteiligt. Sie übernehmen diese Frameworks, um bestimmte Probleme im Data Engineering Lifecycle zu lösen, und entwickeln dann den Framework-Code weiter, um die Tools für ihre Anwendungsfälle zu verbessern und einen Beitrag für die Community zu leisten.

Zu Zeiten von Big Data erlebten wir eine wahre Explosion von Datenverarbeitungsframeworks innerhalb des Hadoop-Ökosystems. Diese Tools konzentrierten sich in erster Linie auf die Phasen Transformation und Bereitstellung des Data Engineering Lifecycle. Die Spezifizierung von Data-Engineering-Tools hat nicht aufgehört oder sich verlangsamt, aber der Schwerpunkt hat sich weg von der direkten Datenverarbeitung auf der Abstraktionsleiter nach oben verlagert. Diese neue Generation von Open-Source-Tools unterstützt die Data Engineers bei der Verwaltung, Verbesserung, Verbindung, Optimierung und Überwachung von Daten.

So dominierte Airflow von 2015 bis Anfang der 2020er-Jahre den Bereich der Orchestrierung. Jetzt ist eine neue Reihe von Open-Source-Konkurrenten (einschließlich Prefect, Dagster und Metaflow) aufgetaucht, um die vermeintlichen Einschränkungen von Airflow zu beheben und eine bessere Metadatenverarbeitung, Portabilität und Abhängigkeitsverwaltung zu bieten. Wohin sich die Zukunft der Orchestrierung entwickelt, ist ungewiss.

Bevor Data Engineers mit der Entwicklung neuer interner Tools beginnen, tun sie gut daran, sich einen Überblick über die Landschaft der öffentlich verfügbaren Tools zu verschaffen. Achten Sie auf die Gesamtbetriebskosten (TCO) und die Opportunitätskosten im Zusammenhang mit der Implementierung eines Tools. Es ist gut möglich, dass es bereits ein Open-Source-Projekt gibt, das sich mit dem Problem befasst, das sie lösen wollen, und sie täten gut daran, zusammenzuarbeiten, anstatt das Rad neu zu erfinden.

Streaming

Die Verarbeitung von Streaming-Daten ist von Natur aus komplizierter als die Batch-Verarbeitung, und die Tools und Paradigmen sind weniger ausgereift. Da das Streaming von Daten in jeder Phase des Data Engineering Lifecycle allgegenwärtig ist, stehen Data Engineers vor interessanten Herausforderungen bei der Softwareentwicklung.

So werden beispielsweise Aufgaben wie Joins, die wir in der Welt der Batch-Verarbeitung für selbstverständlich halten, in Echtzeit oft komplizierter und erfordern eine komplexere Softwareentwicklung. Die Programmiererinnen und Programmierer müssen auch Code schreiben, um eine Vielzahl von *Fensterfunktionen* anzuwenden. Diese Funktionen ermöglichen Echtzeitsystemen die Berechnung wertvoller Metriken wie z. B. Nachlaufstatistiken. Data Engineers stehen viele Frameworks zur Verfügung, darunter verschiedene Funktionsplattformen (OpenFaaS, AWS Lambda, Google Cloud Functions) für die Verarbeitung einzelner Ereignisse oder spezielle Stream-Prozessoren (Spark, Beam, Flink oder Pulsar) für die Analyse von Streams zur Unterstützung von Berichten und Echtzeitaktionen.

Infrastruktur als Code

Infrastruktur als Code (IaC) wendet Methoden aus der Softwareentwicklung auf die Konfiguration und Verwaltung der Infrastruktur an. Der Aufwand für die Verwaltung der Infrastruktur in Zeiten von Big Data hat sich verringert, da Unternehmen zu verwalteten Big-Data-Systemen – wie Databricks und Amazon Elastic MapReduce (EMR) – und Cloud Data Warehouses migriert sind. Wenn Data Engineers ihre Infrastruktur in einer Cloud-Umgebung verwalten müssen, tun sie dies zunehmend mithilfe von IaC-Frameworks, anstatt manuell Instanzen aufzusetzen und Software zu installieren. Mehrere allgemeine und Cloud-Plattform-spezifische Frameworks ermöglichen die automatische Bereitstellung der Infrastruktur auf der Grundlage einer Reihe von Spezifikationen. Viele dieser Frameworks können sowohl Cloud-Dienste als auch die Infrastruktur verwalten. Es gibt auch ein Konzept für IaC mit Containern und Kubernetes, das Tools wie Helm verwendet.

Diese Verfahren sind ein wesentlicher Bestandteil von DevOps, da sie die Versionskontrolle und die Wiederholbarkeit von Implementierungen ermöglichen. Natürlich sind diese Funktionen während des gesamten Data Engineering Lifecycle von entscheidender Bedeutung, insbesondere wenn wir DataOps-Methoden übernehmen.

Pipelines als Code

Pipelines als Code sind das Kernkonzept heutiger Orchestrierungssysteme, die jede Phase des Data Engineering Lifecycle betreffen. Data Engineers verwenden Code (in der Regel Python), um Datenaufgaben und Abhängigkeiten zwischen ihnen zu deklarieren. Die Orchestrierungs-Engine interpretiert diese Anweisungen und führt sie unter Verwendung der verfügbaren Ressourcen aus.

Problemlösung für allgemeine Aufgaben

In der Praxis werden Data Engineers, unabhängig davon, welche hoch entwickelten Tools sie einsetzen, während des gesamten Data Engineering Lifecycle auf kritische Situationen stoßen, die es erforderlich machen, Probleme außerhalb der Grenzen der von ihnen gewählten Tools zu lösen und eigenen Code zu schreiben. Bei der Verwendung von Frameworks wie Fivetran, Airbyte oder Matillion werden Data Engineers auf Datenquellen stoßen, für die es keine Konnektoren gibt, sodass sie ihren eigenen Code schreiben müssen. Sie sollten über Kenntnisse in der Softwareentwicklung verfügen, um APIs zu verstehen, Daten zu extrahieren und zu transformieren, Ausnahmen zu behandeln usw.

Fazit

Die meisten Diskussionen, die wir in der Vergangenheit über Data Engineering geführt haben, beziehen sich auf Technologien, gehen aber am Gesamtbild des Managements des Datenlebenszyklus vorbei. Da Technologien immer abstrakter werden und immer mehr Aufgaben übernehmen, hat ein Data Engineer die Möglichkeit, auf einer höheren Ebene zu denken und zu handeln. Der Data Engineering Lifecycle ist, unterstützt durch seine Unterströmungen, ein äußerst nützliches mentales Modell, um die Arbeit im Data Engineering zu organisieren.

Wir unterteilen den Data Engineering Lifecycle in die folgenden Phasen:

- Generierung
- Speicherung
- Ingestion
- Transformation
- Bereitstellung

Es gibt darüber hinaus mehrere Themen, die sich durch den Data Engineering Lifecycle ziehen. Dies sind die Unterströmungen des Data Engineering Lifecycle:

- Sicherheit
- Datenmanagement
- DataOps

- Datenarchitektur
- Orchestrierung
- Softwareentwicklung

Ein Data Engineer hat mehrere übergeordnete Ziele für den Datenlebenszyklus: Er muss eine optimale Rendite erzielen und die Kosten (finanzielle und Opportunitätskosten) senken, die Risiken (Sicherheit, Datenqualität) reduzieren und den Wert und Nutzen der Daten maximieren.

In den nächsten beiden Kapiteln wird untersucht, wie sich diese Elemente auf ein gutes Architekturdesign auswirken und wie man die richtigen Technologien auswählt. Wenn Sie mit diesen beiden Themen vertraut sind, können Sie gern zu Teil II übergehen, in dem wir die einzelnen Phasen des Data Engineering Lifecycle behandeln.

Weitere Quellen

- »A Comparison of Data Processing Frameworks« (*https://oreil.ly/tq61F*) von Ludovic Santos
- Webseite von DAMA International (*https://oreil.ly/mu7oI*)
- »The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing« (*https://oreil.ly/nmPVs*) von Tyler Akidau et al.
- »Data Processing«, Wikipedia-Seite (*https://oreil.ly/4mllo*)
- »Data Transformation«, Wikipedia-Seite (*https://oreil.ly/tyF6K*)
- »Democratizing Data at Airbnb« (*https://oreil.ly/E9CrX*) von Chris Williams et al.
- »Five Steps to Begin Collecting the Value of Your Data« Lean-Data-Webseite (*https://oreil.ly/F4mOh*)
- »Getting Started with DevOps Automation« (*https://oreil.ly/euVJJ*) von Jared Murrell
- »Incident Management in the Age of DevOps« Atlassian-Webseite (*https://oreil.ly/O8zMT*)
- »An Introduction to Dagster: The Orchestrator for the Full Data Lifecycle« Video (*https://oreil.ly/PQNwK*) von Nick Schrock
- »Is DevOps Related to DataOps?« (*https://oreil.ly/J8ZnN*) von Carol Jang und Jove Kuang
- »The Seven Stages of Effective Incident Response« Atlassian-Webseite (*https://oreil.ly/Lv5XP*)
- »Staying Ahead of Debt« (*https://oreil.ly/uVz7h*) von Etai Mizrahi
- »What Is Metadata« (*https://oreil.ly/65cTA*) von Michelle Knight