17
Wrapping Up the First Part of Your Learning Journey

In this book, we have explored many different aspects of the data engineering role by learning more about common architecture patterns, understanding how to approach designing a data engineering pipeline, and getting hands-on with many different AWS services commonly used by data engineers (for data ingestion, data transformation, orchestrating pipelines, and consuming data).

We examined some of the important issues surrounding data security and governance and discussed the importance of a data catalog to avoid a data lake turning into a data swamp. We also reviewed data marts and data warehouses and introduced the concept of a data lake house.

We learned about data consumers – the end users of the product that's produced by data engineering pipelines – and looked into some of the tools that they use to consume data (including Amazon Athena for ad hoc SQL queries and Amazon QuickSight for data visualization). Then, we briefly explored the topics of **machine learning** (**ML**) and **Artificial Intelligence** (**AI**) and learned about some of the AWS services that are used in these fields.

We also looked at some recent developments, such as the **data mesh** approach, which moves away from a centralized data engineering team and has teams that produce the data and also own the process of creating analytical versions of that data. We also discussed open table formats, such as **Apache Iceberg**, which overcome some of the techni-

cal challenges of traditional data lakes, such as doing record-level updates. Finally, we introduced some important concepts for building a modern data platform, including the pros and cons of building versus buying, and how a **DataOps** approach can help bring automation and observability to a modern data platform.

In this chapter, we're going to review the complexities of real-world data engineering environments (including some examples of real-world data pipelines) and discuss some emerging trends in the field. We'll then look at how to clean up your AWS account in the hands-on portion of this chapter.

In this chapter, we will cover the following topics:

- Understanding the complexities of real-world data environments
- Examining examples of real-world data pipelines
- Imagining the future – a look at emerging trends
- Hands-on – cleaning up your AWS account

Technical requirements

There are no specific technical requirements for the hands-on section of this chapter as we will just be cleaning up resources that we have created throughout this book. Optionally, however, there will be a section that covers deleting your AWS account. If you choose to do this, you will need access to the account's root user to log in with the email address that was used to create the account.

You can find the code files of this chapter in the GitHub repository using the following link: **https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/tree/main/Chapter17**

Understanding the complexities of real-world data environments

This book was never intended as a deep dive into one specific area of data engineering, although there are many other great books and resources out there that do focus on a single area (such as books that focus on Apache Spark programming, or on just how to use Kafka to ingest streaming data). Rather, we took a broad look at the many different areas that are covered by data engineering.

Because of this broad topic coverage, you have probably already begun to form a good idea of the different aspects of the bigger picture of data analytics. While it is quite common for data engineering roles to focus on just writing data transform jobs, or just managing the infrastructure to ingest and process streaming data, it is helpful to understand how this integrates with data warehouses/data marts, how different data consumers use data, and how ML and AI fit into the bigger data picture, as we have reviewed in this book. Having this broader understanding of the big data landscape makes you a better data engineer, no matter what your specific focus is.

We have also been focusing on the tasks from the perspective of a single data engineer, but in reality, most data engineers will work as part of a larger team. There may be different teams, or team members, focused on different aspects of the data engineering pipeline, but all team members need to work together. This is why DataOps processes are so important, as they help teams work together effectively in the process of building data products.

In most organizations, there are also likely to be **multiple environments**, such as a development environment, a test/**quality assurance (QA)** environment, and a production environment. The data infrastructure and pipelines must be deployed and tested in the **development environment** first, and then any updates should be pushed to a **test/QA environment** for automated testing, before finally being approved for deployment in the **production environment**.

In the following diagram, we can see that there are multiple teams responsible for different aspects of data engineering resources. We can also see that the data engineering resources are duplicated across multiple different environments, such as the development environment, test/QA environment, and production environment (and these would generally be separate AWS accounts). Each organization may structure its teams and environments a little differently, but this is an example of the complexity of data engineering in real life:

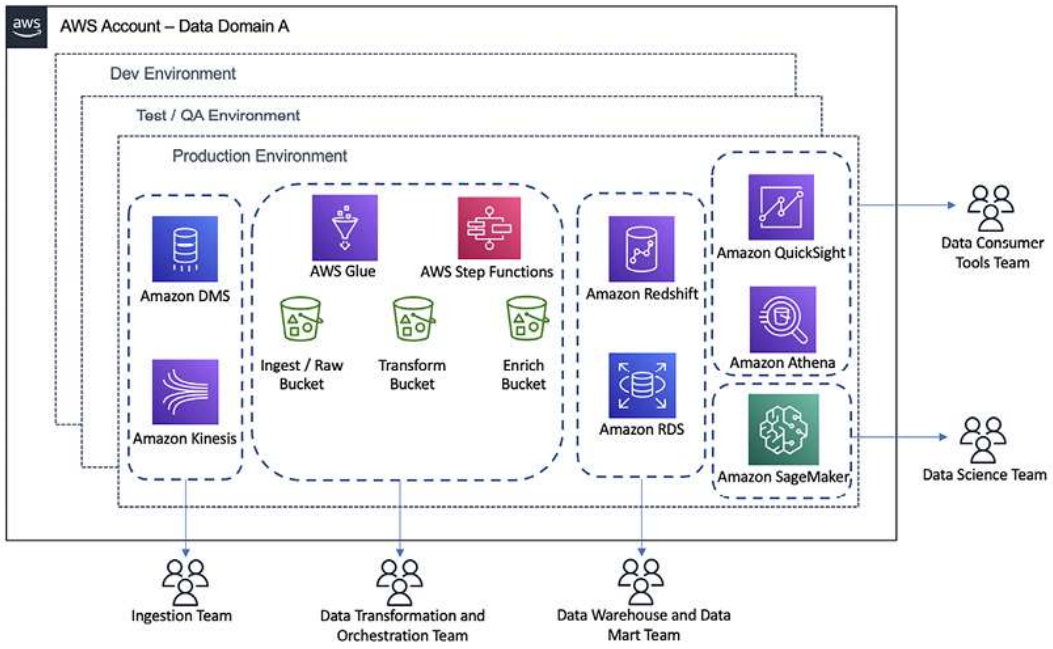

Figure 17.1: Data engineering teams and environments

If using a data mesh approach to organize your data teams, then each data domain would have multiple environments and teams, as shown in *Figure 17.1*. However, for smaller data domains, there may be a single team with a variety of skills that manages all of the aspects shown in the diagram. There is no best way to structure teams within a domain, as it mostly depends on the scale of the data (including the number and complexity of data sources), and the range and complexity of required data transforms. For a small domain with just a few data sources that are creating just one or two data products, it is possible that all the functions shown in *Figure 17.1* will be managed by a single

data engineer, building on automation and functionality provided by the central data mesh team (which we discussed in *Chapter 15*, *Implementing a Data Mesh Strategy*).

Let's now look at what this may look like for an organization with multiple data domains, and how the different domains come together with a central data mesh platform, as shown in *Figure 17.2*.

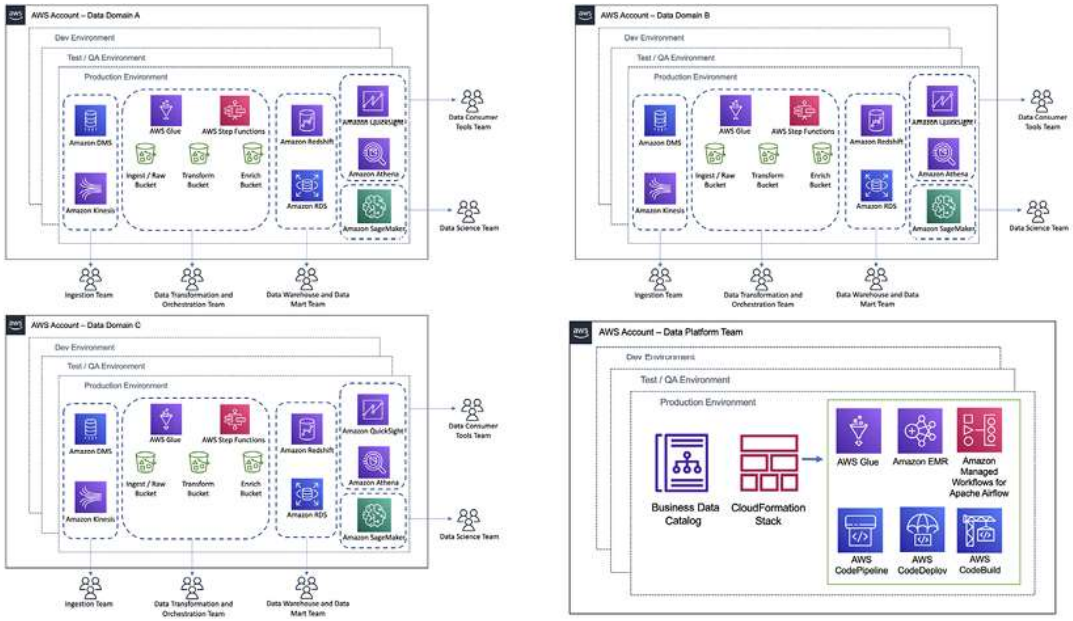

Figure 17.2: Data engineering environments in a data mesh

In *Figure 17.2*, we see three data domains, and at the bottom right, a central data platform. Note how the central data platform team is responsible for a business data catalog, which each data domain uses to publish metadata for their data products. The central data platform team may also create and manage **CloudFormation** stacks, which are used to automate the deployment of resources such as AWS Glue, Amazon EMR, and Amazon **Managed Workflows for Apache Airflow** (**MWAA**). The data platform team may also deploy AWS CodeBuild, CodeDeploy, and CodePipeline, which each domain uses to implement DevOps processes.

In this way, the data platform team helps simplify the process of implementing data engineering pipelines for each data domain, by providing automation for resource and code deployment.

Having discussed some of the complexities of real-world data environments, let's now look at some examples of real-world data pipelines.

Examining examples of real-world data pipelines

The data pipeline examples that we have used in this book have been based on common types of transformations and pipelines, but they have been relatively simple examples. As you can imagine, in large organizations, the types of data pipelines that are built can be a lot more complex and may end up processing extremely large sets of data.

In this section, we will examine two examples of more complex data engineering pipelines from two very well-known organizations – **Spotify** and **Netflix**. Both of these companies have public blogs that cover software and data engineering, and the details provided about their pipelines in this section have been taken from the public information that's been made available in a variety of blog posts and articles.

By learning more about these real-world big data pipelines, you can be better prepared for what to expect when you start working with very large datasets. Also, these examples teach valuable lessons from cutting-edge companies about how they approach complex big data processing tasks, and you can apply these types of approaches and lessons learned to your own complex engineering challenges in the future.

A decade of data wrapped up for Spotify users

Each year, the music streaming service *Spotify* uses the extensive data they have on their users' listening history to generate interesting stats

for each user. This information is made available to each user at the end of the year and includes information such as how many minutes of Spotify audio they streamed that year, as well as their top artist, top track, and top genre for the year.

In 2022, Spotify added a new feature – an overview of a user's "listening personality" that displayed a four-letter code that was a combination of four binary attributes that each try to measure and describe one aspect of *how* a user listens to music, independent of *what* music they like. For example, it analyzed how much a user listened to their favorite artists versus how much they explored new artists, and how much they listened to newly released songs versus listening to older songs.

To generate these statistics and metrics related to each user of the platform, the data engineering team at Spotify has to gather and analyze detailed information about every user's listening history, such as:

- Which songs they played
- Which artists they listened to
- The top genres they listen to
- Whether the user listens to the same artist all the time, or often listens to different artists
- The age of each song they listened to, in order to determine whether they listened to more new songs or preferred older songs

A summary of the information is then generated for users, and users can review this in a feature called **Spotify Wrapped**. Putting this all together is a massive undertaking for multiple teams at Spotify, including marketing, frontend app engineering, and, of course, data engineering.

While Spotify has been presenting the Spotify Wrapped feature for several years, in 2019, they decided to add a new feature that reports a user's listening trends for each year of the past decade (2010 – 2019).

In an official Spotify blog post, *Spotify Unwrapped: How we bought you a decade of data* (https://engineering.atspotify.com/2020/02/18/spotify-unwrapped-how-we-brought-you-a-decade-of-data/), the Spotify data engineering team revealed some of the behind-the-scenes work they did to aggregate user data by year, over 10 years.

In this blog post, the data engineering team talks about some challenges they faced with the Wrapped project in 2018, and how they had to work closely with Google (their cloud provider) to be able to achieve the required processing scale. For 2019, they were planning to do something similar to 2018, but they had more users (totaling 248 million monthly active users at the time) and were planning to do this for 10 years of listening history. As a result, they used the lessons they had learned from their 2018 experience to modify their approach for 2019.

Spotify considers each statistic they want to report for an individual user (such as top artist or top track) as a separate data story. So, to meet the scale requirements for a decade of data, they decided to persist intermediate data and final data for Spotify Wrapped 2019 in **Google BigTable** (a NoSQL database that is somewhat similar to Amazon DynamoDB). For every Spotify user, they had a row in BigTable with a column for each data story, for each year of the decade. This was a significant change from how they had processed and collected different data stories for each user in previous years, but this led to a significantly improved process as data was now pre-grouped and collated per user in BigTable.

They could then write separate jobs for most data stories (decoupling the data stories from each other) and run these individually, but could also run multiple different data story jobs in parallel. The output of each of these data story jobs would then be saved in BigTable, with a row for each Spotify user. End-of-decade top statistics could then be aggregated directly from the data in BigTable.

The key takeaways that we can learn from this example are as follows:

- It is good to iterate on data engineering pipelines and continually reevaluate the architecture and approach you use to identify better ways to do things.
- Breaking down large jobs into smaller, decoupled jobs can lead to improved efficiencies. Keep a modular design for your jobs and avoid the temptation to create a single job that does everything.
- Be versatile and flexible in the tools you use. While we did not have space to cover NoSQL databases in any significant way in this book, a NoSQL database may be an ideal target for storing some of the output from your big data processing jobs. For example, **DynamoDB** was designed to handle billions of rows of data in a table, as well as enable extremely fast access to individual rows from that large dataset.

Data engineers are often challenged to come up with innovative new ways to draw insights out of extremely large datasets, as demonstrated in this real-world example from Spotify. Now, let's look at another real-life data processing example, this time from Netflix.

Ingesting and processing streaming files at Netflix scale

Netflix, the world's leading streaming video platform, with over 230 million subscribers worldwide, predominantly uses AWS for its compute infrastructure. As you can imagine, it takes a lot of compute power, and many different microservices and applications, to support a user base of that size.

Monitoring and understanding how network traffic flows between all the different Netflix microservices, across many separate AWS accounts, is key for the following:

- Maintaining a resilient service

- Understanding dependencies between services
- Troubleshooting when things do go wrong
- Identifying ways to improve the user experience

One of the features of the Amazon **Virtual Private Cloud** (**VPC**) service (a private cloud-based network environment in an AWS account) is the ability to generate **VPC Flow Logs**, which capture details on network traffic between all network interfaces in a VPC.

However, most AWS services make use of dynamic IP addresses, meaning that the IP address that's used by a system can frequently change. So, while VPC Flow Logs provide rich information on network communications between IP addresses, if you don't know which applications or services had the IP addresses being reported on at that time, the Flow Logs are largely meaningless.

Enriching VPC Flow Logs with application information

To have data that was meaningful, Netflix determined that they needed to enrich VPC Flow Logs with information about which application was using a specific IP address at the point in time recorded in the VPC Flow Log. To capture this information, Netflix created an internal system called Sonar that uses CloudWatch Events, Netflix Events, API calls, and various other methods to capture a stream of IP change events.

At the 2017 AWS Summit in Chicago, Netflix presented a breakout session about this solution (available on YouTube at `https://www.youtube.com/watch?v=8tsIqfvizpU`). In the video, Netflix explains how they used a large **Kinesis Data Streams cluster** (with hundreds of shards) to process incoming VPC Flow Logs. An internal Netflix application known as **Dredge** was created to read incoming data from the Kinesis data stream, as well as enrich the VPC Flow Log data with application metadata from the Sonar stream of IP

change events, identifying the applications or microservices involved with each VPC Flow Log record. This enriched data was then loaded into an open-source, high-performance, real-time analytics database called **Druid**, where users could efficiently analyze network data for troubleshooting and to gain improved insights into network performance (to learn more about Apache Druid, see https://druid.apache.org/).

Amazon VPC enhancements and changing the architecture

In the cloud, things change frequently, and AWS is constantly enhancing its services and adding additional services in response to customer feedback. In August 2018, AWS enhanced the VPC Flow Logs service so that logs could be delivered directly to Amazon S3, without needing to be processed via Kinesis first.

In May 2020, Netflix posted a public blog post titled *How Netflix is able to enrich VPC Flow Logs at Hyper Scale to provide Network Insight* (https://netflixtechblog.com/hyper-scale-vpc-flow-logs-enrichment-to-provide-network-insight-e5f1db02910d). This blog post shows how Netflix has changed its architecture to make the best use of the updated functionality in the VPC Flow Logs service.

In this blog post, Netflix talks about a common pattern that they have for processing newly uploaded S3 files. When a new file is uploaded to S3, it is possible to configure an action to take place in response to the newly uploaded file (as we did in *Chapter 3*, *The AWS Data Engineer's Toolkit*, where we triggered a Lambda function to transform a CSV file into Parquet format whenever a new CSV file was uploaded to a specific S3 bucket prefix).

Netflix commonly uses this pattern to write details of newly uploaded files to an Amazon SQS queue, and they can then read events from the queue to process the newly arrived files. This enables them to decou-

ple the S3 event from the action that they wish to perform in response to this event.

In this case, Netflix intended to read through the entries on the SQS queue and use the file size information included in the event notification to determine the number of newly ingested VPC Flow Log files to process in a batch (which they refer to as a *mouthful* of files). They intended to use an Apache Spark job that would enrich the VPC Flow Log with application metadata based on the IP addresses recorded in each record. They would tune the Apache Spark job to optimally process a certain amount of data, which is why they would read the file size information contained in the SQS messages to create an optimally sized *mouthful* (batch) of files to send to the Spark job.

With the Amazon SQS service, messages are read from the queue and processed. If the processing is successful, the processed messages are deleted from the queue. During the time that a batch of messages is being processed, the messages are considered to be *in flight* and will be hidden from the queue so that no other application attempts to process the same files. If something goes wrong and the files are not successfully processed and deleted from the queue, the messages will become visible again after a certain amount of time (known as the *visibility timeout period*) so that they can be picked up by an application again for processing.

In the case of Netflix, they would send a mouthful of files to an Apache Spark job, and once the Spark job successfully processed the messages, the messages would be deleted from the queue.

However, the Amazon SQS service has a limit on the number of files that can be considered to be in flight at any point (the default quota limit is 120,000 messages). Netflix found that because the Spark jobs would take a little while to process the files, they were regularly ending up with 120,000 or more messages in flight, causing issues. As a re-

sult, they came up with an innovative way to work around this by using two different SQS queues.

Working around Amazon SQS quota limits

The re-architected Netflix solution reads the SQS queue containing the S3 events and runs a process to create a mouthful of files (evaluating each file's size to create a batch that is the optimal size for their Spark jobs).

This process can complete very quickly as it does not need to read or process the files, just read the metadata contained in the SQS messages to group a mouthful of files to be processed by a Spark job.

The output of the first job writes a message to a second SQS queue, and each message contains the list of files in a single mouthful. While the blog does not provide any indication of how many files may usually be contained in a mouthful of files, if we assumed it was, on average, around 10 files, it would reduce the number of messages on the second SQS queue by 90%. If a mouthful of files was, on average, 100 files, then the number of messages written to the secondary SQS queue would be reduced by 99%.

The Netflix blog does not provide enough details to be able to describe the exact architecture of the solution, but the following diagram shows an example of a potential architecture we could design for this solution (however, this may be very different from the architecture that Netflix implemented):

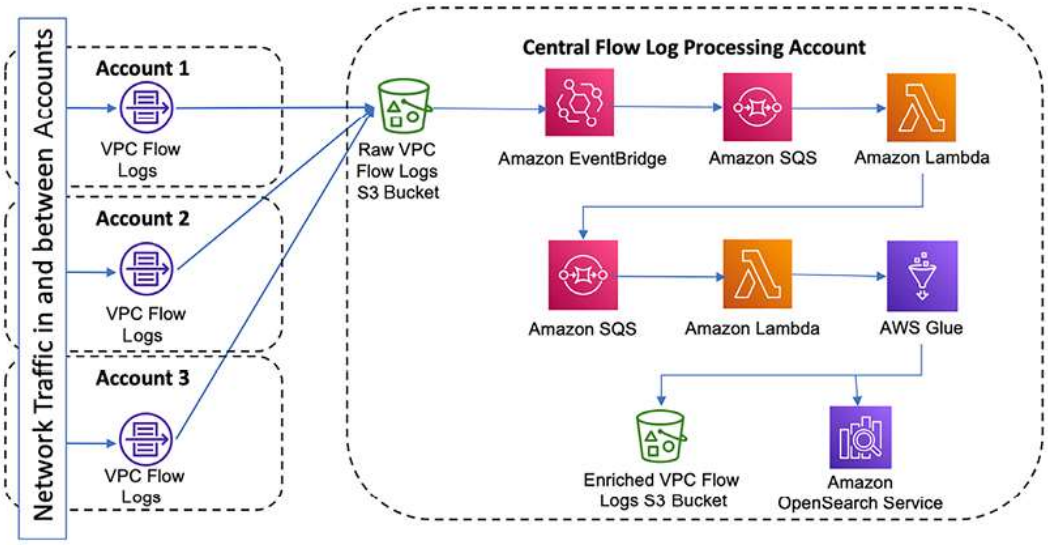

Figure 17.3: A potential architecture for VPC Flow Logs processing and enrichment

In the preceding diagram, we have VPC Flow Logs configured to write to an Amazon S3 bucket in a central flow log processing account. As each new Flow Log is written into this central bucket, it triggers an **EventBridge** rule with details about the newly written file, including the size of the file.

The EventBridge rule has an Amazon SQS queue configured as a target, so as each new file is written to the Amazon S3 bucket, EventBridge forwards the event information to an Amazon SQS queue.

A Lambda function has been configured to read messages from the Amazon SQS queue and uses the file size metadata contained in the message to create a batch of files of an optimal size. The list of files in the batch is written to a separate SQS queue as a single message. At this point, our first Lambda function can complete, and remove messages from the first SQS queue very quickly, since it is only processing metadata in the SQS message, not reading/writing S3 files, and running a Spark job to enrich the files.

A separate Lambda function processes the much smaller number of messages in the secondary SQS queue by reading the list of files (the mouthful of files) in each message. The list of files is passed to a Spark job (in our example architecture, we are using AWS Glue to run the job), and this job enriches the VPC Flow Log files in this mouthful with data from other sources. Enriched files are written to another Amazon S3 bucket, and/or a system designed for storing and searching through log-style data, such as **Amazon OpenSearch Service**.

The key takeaways that we can learn from this example are as follows:

- It is important to know what the AWS quotas/limits are for the services that you use. Some limits can be raised by contacting AWS support, but some limits are hard limits that cannot be increased.
- It is important to stay up to date with *what's new* announcements from AWS. AWS regularly launches new services, as well as major new features, for existing services. Bookmark the following web page, which lists all new AWS features and services: `https://aws.amazon.com/new`.

As shown in this blog post, new features launched by AWS may help you significantly simplify existing architectures and reduce costs (based on this blog post, it would seem that Netflix may no longer need their Kinesis Data Streams cluster, configured with hundreds of shards, in order to process VPC Flow Logs).

In the next section, we will look at upcoming trends and what the future may hold for data engineers.

Imagining the future – a look at emerging trends

Technology seems to progress at an increasing velocity. For decades, relational databases from vendors such as Oracle were the primary technology for managing all data. Today, there is a wide range of dif-

ferent database types that can be used, depending on the use case (such as graph databases for highly connected datasets, NoSQL databases for low-latency reading and writing of very large tables, and vector databases, which have become popular for ML applications such as generative AI).

It was also not all that long ago that **Hadoop MapReduce** was the state-of-the-art technology for processing very large datasets, but today, most new projects would choose **Apache Spark** over a MapReduce implementation. Apache Spark itself continues to progress from its initial release, with Spark 3.4 having been released in April 2023. We have also seen the introduction of **Spark Streaming**, **Spark ML**, and **Spark GraphX** for different use cases.

No one can tell for certain what the next big thing will be, but in this section, we will look at a few emerging concepts and technologies, as well as expected trends, that are likely to be of relevance to data engineers.

Increased adoption of a data mesh approach

In the past few years, we have seen a rapid increase in the number of companies that are looking to implement a data mesh-type approach to how they organize and structure data responsibilities. This will continue to lead to a move away from centralized data engineering teams to data engineers being assigned to specific data domains.

This will also lead to an increase in dedicated *data platform engineer* roles, with the platform engineers being responsible for the central data platform and catalog, and not responsible for developing ETL pipelines that perform data transformation. Instead, data engineers responsible for developing data transformation pipelines will be embedded in data domains, and make use of infrastructure templates managed by the central data platform team.

There will also be an increase in roles that are responsible for data governance, ensuring that data quality is high and that data lineage and protection of PII data are prioritized.

As discussed in *Chapter 15*, *Implementing a Data Mesh Strategy*, the term data mesh means different things to different people and was never intended as a technical solution but more of a theoretical approach. Therefore, not every data mesh implementation will be the same, and we will not always see a role for people who are focused on architecting the data mesh implementation for a company.

Requirement to work in a multi-cloud environment

While this book focuses on data engineering using AWS services, it is not uncommon for many larger companies to have a **multi-cloud strategy**, where they use more than one cloud provider for services.

Having a multi-cloud strategy can introduce numerous challenges across **information technology** (**IT**) teams, including challenges for data engineering teams that need to work with data stored with different cloud providers.

Another challenge for IT teams and data engineers is the need to learn the different service implementations for each cloud provider (for example, AWS, Azure, and Google Cloud each offer a managed Apache Spark environment, but the implementation details are different for each provider).

From a data analytics perspective, each of the cloud providers is starting to do more to support querying analytics data across clouds. For example, Amazon Athena and AWS Glue both have connectors that you can deploy to enable those services to read data from sources such as Google BigQuery and Azure Data Lake Storage.

Perhaps a bigger challenge is to have a unified data catalog that can efficiently span across multiple clouds, in order to create a centralized repository of business data, no matter which cloud the data is stored in.

There are many different reasons for organizations wanting to adopt a multi-cloud strategy, but the pros and cons need to be carefully thought through. However, in many cases, data engineers will have no option but to take up the challenge of becoming comfortable with working in, and across, multiple different cloud provider environments.

Migration to open table formats

As we discussed in *Chapter 14*, *Building Transactional Data Lakes*, the last few years have seen the development of a number of **open table formats**, which provide a modern approach to developing data lakes that are able to be easily updated in a consistent manner (in addition to other features).

While difficult to predict which of the table formats will become dominant, it appears that **Apache Iceberg** has the most momentum currently. But whichever format (or formats) ends up becoming the most popular, you can expect that many organizations will be working on migrating to one of the open table formats over the next few years.

This will involve migrating existing Hive-based data lake tables to one of the new open table formats, and modifying existing ETL jobs to make use of the new features provided. Considering that most large organizations likely have thousands of datasets and transformation jobs, this will be a significant undertaking. However, there are significant benefits provided by these table formats, so it will be a worthwhile project for these organizations. These migration projects are likely to be long-term projects, and it may take a number of years to

migrate all existing data lake tables and transformation jobs to a new open table format.

Managing costs with FinOps

In recent years, there has been increasing focus on monitoring and managing cloud-related spend, and this of course also applies to data-related spend.

This trend is likely to continue, and as a result, organizations are increasingly developing a **FinOps** function. The purpose of FinOps is to create a cross-functional team that helps monitor IT-related spend and bring engineering, finance, technology, and business teams together to work on making data-driven spending decisions.

It is important for data teams to have a program in place that helps educate the data engineering team, the central data platform team, and all other associated teams about best practices for cloud cost management. This team should be part of, or be represented within, the FinOps function.

While the cloud offers incredible scale, flexibility, and agility, it can be more challenging to predict future costs. It is great to provide a data engineering team access to a development environment where they can create and run AWS Glue Spark-based jobs on demand, and then deploy those to production. But a data engineer's primary role is to transform the data, and the cost of that transformation is often not a priority for the data engineer.

By default, a Glue job is assigned 10 DPUs, but many smaller jobs may not require that many DPUs. Also, auto-scaling is an option in Glue jobs (where Glue automatically uses only the needed DPUs, up to a maximum number of DPUs), but this needs to be enabled for each job. If only using the default Glue settings, a job may cost more than re-

quired. But with just a little bit of education, each team can learn how to easily cost-optimize their Glue jobs.

To be a successful data engineer, it is increasingly important that you learn how to manage and optimize costs for your data engineering pipelines and data platform. And ensuring that your organization has a cross-functional FinOps team is, or will become, critical to the success of your cloud projects.

The merging of data warehouses and data lakes

Another trend that we have seen over the past few years is the merging of data warehouses and data lakes.

Snowflake architecture resembles that of a data warehouse, and the **Databricks** platform architecture provides more of a data lake approach. However, in their marketing, one of the Snowflake messaging campaigns is about "*Snowflake for Data Lakes*." The Databricks documentation talks about their Databricks Lakehouse platform providing a "*complete end-to-end data warehousing solution*."

For a while, many companies used variations of the term "Data Lake House" to describe a combination of a data lake and a data warehouse. These terms were used by different marketing teams to mean slightly different things, but ultimately it was used to describe how data lake and data warehouse technologies were becoming more interoperable.

For example, within AWS, Redshift Spectrum can be used to query data in an Amazon S3-based data lake, Amazon Athena can be used to query and join data in multiple sources (including an Amazon S3-based data lake, and an Amazon Redshift data warehouse), and AWS Glue works with open table formats to enable more data-warehouse-type functionality.

This trend continues, even if the terms somewhat change. Over the next few years, we are likely to continue to see the lines blur between data lakes and data warehouses, with traditional solutions for each encompassing more of the features and functionality of the other. The advent of open table formats (previously discussed) is enabling and accelerating this trend.

Data warehouses are increasingly able to query open table formats that exist on object storage (such as Amazon S3), and data lake technologies are increasingly including functionality that enables them to act more like a data warehouse (also enabled by open table formats). Ultimately, as we see the increase in the adoption of open table formats, it will become more difficult to separate the functions of a data lake and a data warehouse.

The application of generative AI to business intelligence and analytics

With the growing popularity of **generative AI** solutions (such as **ChatGPT**) in recent times, there is increasing interest in being able to query data in a data store (such as a data lake) using natural language. The ultimate goal would be to enable an executive to type something like "*On which day of the week do we see the highest sales of our chocolate flavored ice cream?*". And in response, the generative AI-powered assistant may ask some necessary clarifying questions, but will ultimately return an accurate result, or perhaps even a visualization showing the requested data.

Today, there are some experimental approaches that attempt to use **generative AI**/**LLMs** to generate the SQL required to query a data store, in response to a prompt (for an example, see the following AWS blog post – `https://aws.amazon.com/blogs/machine-learning/reinventing-the-data-experience-use-generative-ai-and-modern-data-architecture-to-unlock-insights/`). However, these are not yet at a point where they are can be counted on to pro-

vide guaranteed accurate and high-quality results to all potential queries.

Over the next few years, we can expect to see improvements in applying generative AI/LLMs to data analytics. For example, in July 2023, AWS announced new generative **business intelligence** (**BI**) features were coming to Amazon QuickSight. This functionality builds on the existing functionality of QuickSight Q (which we covered in *Chapter 12*, *Visualizing Data with Amazon QuickSight*).

In the announcement, AWS explained that the new generative BI capabilities would include functionality for:

- Rapidly creating visuals using a new QuickSight Q-powered visual authoring experience
- Fine-tuning and formatting visuals using natural language
- Creating calculations using natural language without needing to know specific syntax
- Enabling business users to create generative BI-powered *stories*, which automates much of the process of generating narrative and visual presentations around a data topic by simply entering a description of the story they want to tell

For example, with *stories*, a business user can type something such as "*Build a story about how we can increase the conversion of free trial customers into paying accounts so we can boost sales*." QuickSight will put together a document that includes visuals, along with narrative text, that explores the topic. To learn more about this, see the AWS blog post at **https://aws.amazon.com/blogs/business-intelligence/announcing-generative-bi-capabilities-in-amazon-quicksight/**.

The application of generative AI to building transformations

In addition to generative AI being used to simplify BI and analytic tasks, we can expect to see increasing usage of these new LLMs for building ETL-type transformations.

For example, AWS Glue already supports **Amazon CodeWhisperer**, an AI coding companion that can help developers as they write ETL transformation code, such as PySpark code. CodeWhisperer has been built into AWS Glue Studio notebooks, enabling a developer to enter a comment in their code describing what they are trying to do and have CodeWhisperer generate suggested code to achieve that function.

For example, if a developer has a Spark DataFrame that includes a column named `quantity` and another named `price`, they can write the following comment in their Glue Studio notebook:

```
# Add a column to calculate the total price
```

CodeWhisperer will then generate some suggested code, and the developer can use the up and down arrows to look through different code suggestions and then hit **TAB** to accept the suggestion. For example, CodeWhisperer may suggest the following code, which the developer can accept by hitting **TAB**:

```
df = df.withColumn('total_price',  df.quantity * df.price)
```

For more details on using Amazon CodeWhisperer with AWS Glue, see the following blog post: https://aws.amazon.com/blogs/big-data/build-data-integration-jobs-with-ai-companion-on-aws-glue-studio-notebook-powered-by-amazon-codewhisperer/. For general information on CodeWhisperer and integrations with various IDEs, such as Visual Studio Code, see the following documentation:

https://docs.aws.amazon.com/codewhisperer/latest/userguide/setting-up.html.

Over the next few years, we can expect that these AI coding assistants will significantly improve, perhaps to the point where an analyst can describe the transformation they want to apply to a dataset in natural language, and the AI assistant will generate all the code required and then show a sample of the result for the analyst to approve. Once approved, the AI assistant can then automatically commit the code to a code repository. The analyst can then describe, in natural language, a pipeline that runs multiple transformations, and the AI assistant will generate an Apache Airflow DAG for the pipeline, commit the DAG to a code repository, and then deploy the pipeline.

It is not possible to know exactly how generative AI and LLMs will impact the future of data engineering, but it will be an interesting topic to watch and keep up with.

Having looked at some practical implementations of real-world data engineering, examples of real-world data pipelines, and emerging trends and concepts, we will now move on to our final hands-on section of this book.

Hands-on – cleaning up your AWS account

In the hands-on section of *Chapter 1*, *An Introduction to Data Engineering*, we went through how to create a new AWS account. If you created a new account at that point, and have used that account to work through the exercises in this book, you may want to delete that account now that you have reached the final chapter of this book. We'll include instructions on how to do that here.

However, if this was your first AWS account, you may decide that you want to keep the account open so that you can continue to explore and learn more about AWS using other resources. If that is the case, we'll include some instructions on how to check your account billing to detect which resources you are still being charged for.

Reviewing AWS Billing to identify the resources being charged for

While many of the services that we have used in this book are included in the AWS Free Tier (where you are not charged for certain limited usage of specific services), other services will have had a cost associated with their use.

In this section, we will go through how to review the AWS Billing console to determine which resources you are being charged for:

1. Log in to the **AWS Billing** console using the following link: `https://console.aws.amazon.com/billing/home`.
2. On the **Billing Dashboard**, we can immediately see a forecast of what the current month's spend will be, as well as the actual month-to-date cost, and a comparison of the current month's spend compared to the previous month at the same point:

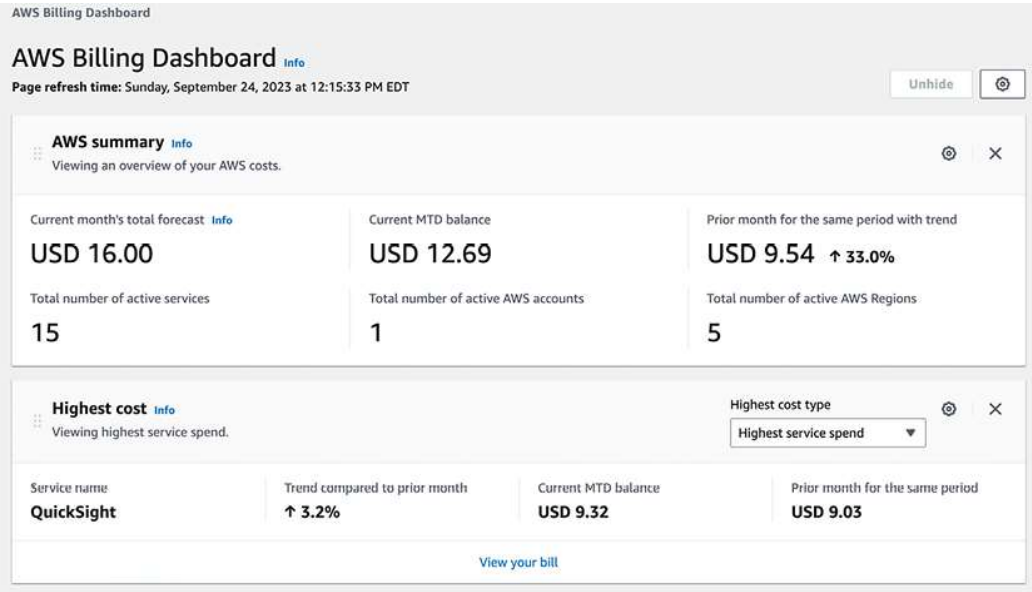

Figure 17.4: AWS Billing Dashboard

In the preceding screenshot, you can see that I have spent **$12.69** so far this month, and that the forecast for the full month is a total of **$16.00**.

3. Scroll further down the Billing Dashboard and review the section titled **Cost trend by top five services**.

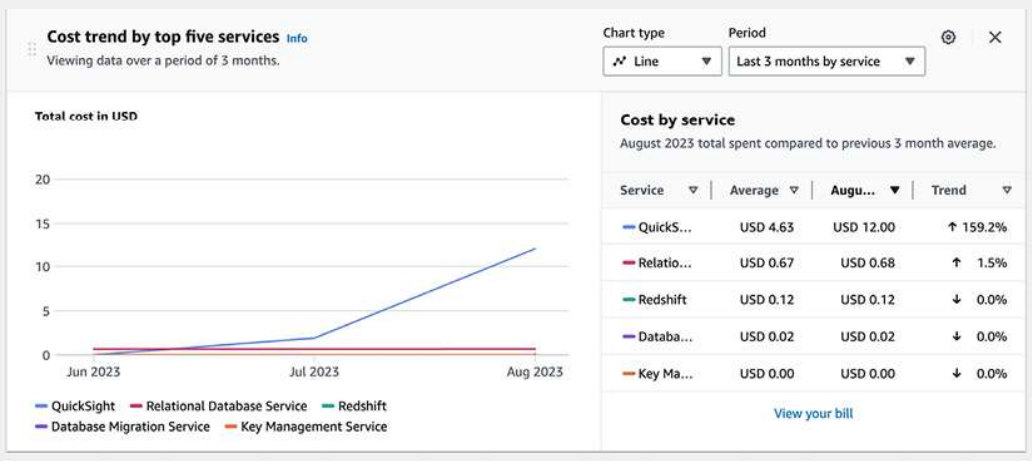

Figure 17.5: AWS Billing Dashboard – Cost trend by top five services

I did not cancel my **QuickSight** subscription after completing the exercises in *Chapter 12*, *Visualizing Data with Amazon QuickSight*, and my free 30-day trial ended. If I wanted to cancel my QuickSight subscription now to avoid any future charges after this month, I could follow the instructions in *Deleting your Amazon QuickSight subscription and closing the account* (https://docs.aws.amazon.com/quicksight/latest/user/closing-account.html).

4. I can also see charges for **Relational Database Service** and **Redshift**. I am not sure what these charges relate to, so to investigate this further, I can click on **Bills** in the left-hand menu. Once on the **Bills** page, I can scroll down and review the section titled **Amazon Web Services, Inc. charges by service**.
   On this page, I can expand the **Redshift** item to examine those costs.

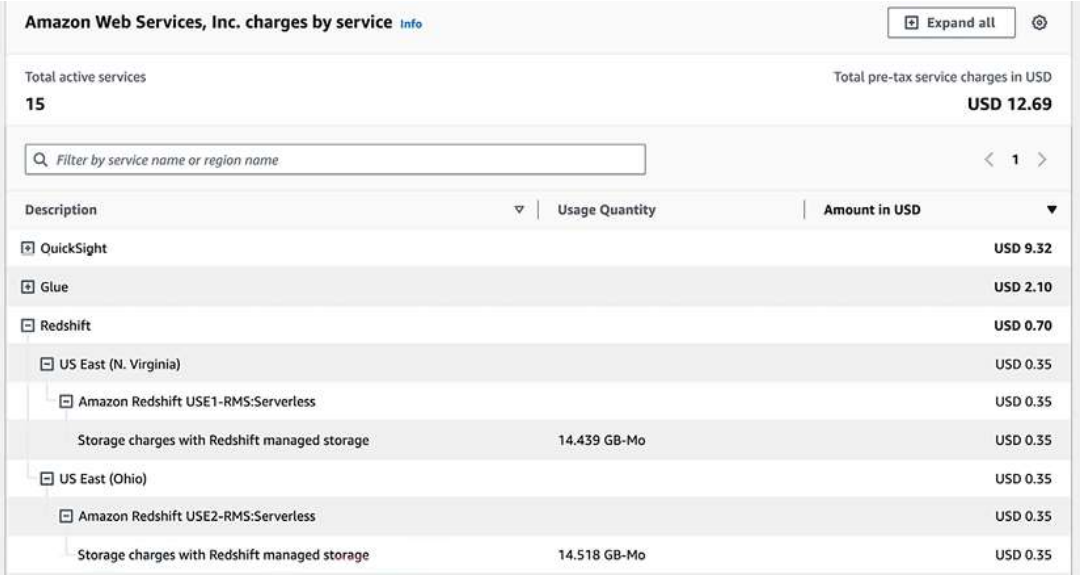

Figure 17.6: AWS bill details view

Here, I can see that the Redshift charges are related to two Amazon Redshift Serverless clusters that I had previously created – one in the US East (N. Virginia) Region and one in the US East (Ohio) Region. I have not performed any queries with the cluster recently, but I am being billed for the amount of storage I am using, from when I previously loaded data into the serverless cluster. If I no longer needed the cluster and data, then to avoid being charged in the future, I could go to the Redshift console, view the configured **namespace**, then delete any associated **workgroups** within that namespace, and then delete the namespace. I would of course need to do this for each Region where I had Redshift Serverless configured.

In a similar way, I could expand the lines for **Glue** and **Relational Database Service** (**RDS**), and identify what items I am being charged for. In my case, the Glue charges were related to a Glue job that I had run, so as long as I am not running any future jobs, I will not have any future Glue charges. For RDS, the charges were for backups that I had taken of database instances, so I could delete those snapshots if no longer needed, and that way I would not have any future RDS charges.

If I canceled my QuickSight subscription, deleted my Redshift workgroups and namespaces, and deleted my RDS snapshots, I could continue using my AWS account without incurring additional charges for those items. However, it is strongly recommended that you regularly check the Billing console and set billing alarms to alert you of spending above the limit you've set.

For more information, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/monitor_estimated_charges_with_cloudwatch.html.

If you want to keep your AWS account so you can continue to explore and experiment with AWS services, then going through the Billing console to identify what AWS services you are paying for and deleting those resources you don't need is all you need to do for now. However, in the next section, we'll cover what to do if you want to fully delete your account.

Closing your AWS account

If you decide that you want to close your AWS account, you can do so with the following steps.

Before proceeding, make sure that you have read the *Considerations before you close your AWS account* section of the AWS documentation at https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/close-account.html. If you want to close your account, use the following steps:

1. Log in to your AWS account as the root user of the account (that is, using the email address and password you registered when you opened the account). Use the following link to log in: https://console.aws.amazon.com.

If you're prompted for an IAM username and password, click on the link for **Sign in using root user email**.

2. Enter your root user email address and password when prompted.
3. Open the Billing console with the following link: https://console.aws.amazon.com/billing/home#/.
4. In the top-right corner, select the dropdown next to your account number (or account alias, if set). From this dropdown, select **Account**:

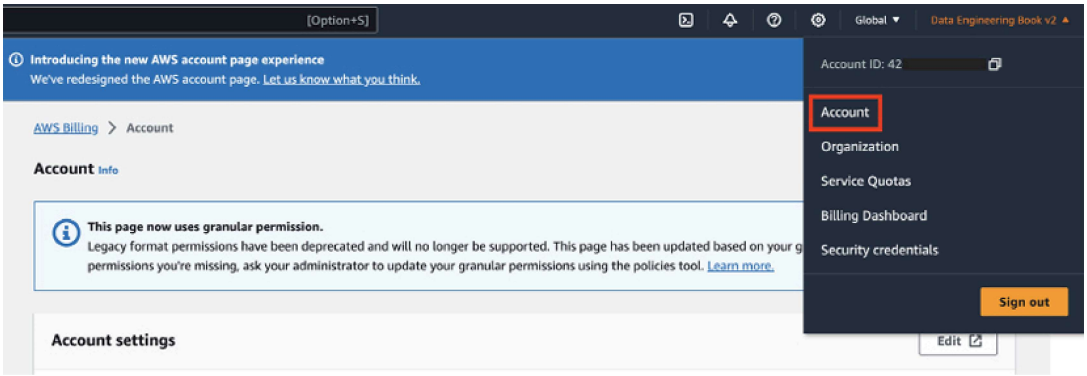

Figure 17.7: Accessing the Account screen in the AWS Management Console

5. Scroll to the bottom of the **Account** page. Read and ensure you understand the account closure guidance. If you still want to close your account, click **Close account**:

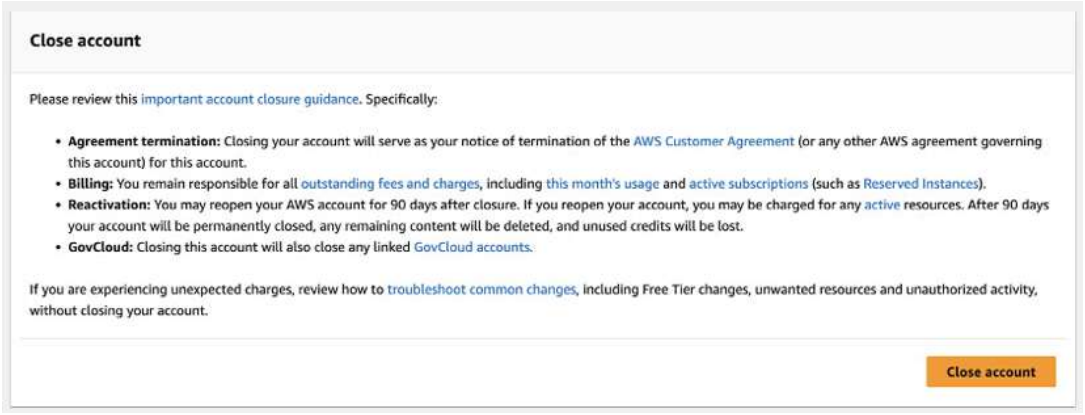

Figure 17.8: Closing your AWS account

6. In the pop-up box, click **Close account** to confirm that you want to close your account.

Subsequently, if you change your mind about closing your account, it may still be possible to reopen your account within 90 days of choosing to close it. To do so, contact AWS support.

Summary

Data engineering is an exciting role to be in and is one that offers interesting challenges, constant learning opportunities, and increasing importance in helping organizations draw out the maximum value that they can from their data assets. And the cloud is an exciting place to build data engineering platforms and pipelines.

AWS has a proven track record of listening to their customers and continuing to innovate based on their customers' requirements, so you can expect new features and services at a virtually constant pace. Things move quickly with AWS services, so hold on tight for the ride.

If you're new to data engineering on AWS, then this book is just the start of what could be a long and interesting journey for you. There is much more to be learned than what could ever be captured in a single book, or even a volume of books. Much of what you will learn will be through practical experience and things you learn on the job, as well as from other data engineers.

But this book, and other books like it, as well as resources such as podcasts, YouTube videos, and blogs, are all useful vehicles along your journey. Let the end of this book be just the end of the first chapter of your learning journey about data engineering with AWS.

Learn more on Discord

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:

https://discord.gg/9s5mHNyECd