
O'REILLY®
Compliments of
snowflake®
Architecting Data-Intensive SaaS Applications
Building Scalable Software with Snowflake
William Waddington, Kevin McGinley, Pui Kei Johnston Chu, Gjorgji Georgievski & Dinesh Kulkarni
REPORT

BUILD DATA-INTENSIVE APPLICATIONS WITHOUT OPERATIONAL BURDEN

developer.snowflake.com

Architecting Data-Intensive SaaS Applications

Building Scalable Software with Snowflake

William Waddington, Kevin McGinley, Pui Kei Johnston Chu, Gjorgji Georgievski, and Dinesh Kulkarni

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Architecting Data-Intensive SaaS Applications
by William Waddington, Kevin McGinley, Pui Kei Johnston Chu, Gjorgji Georgievski, and Dinesh Kulkarni

Copyright © 2021 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Jessica Haberman
Development Editor: Michele Cronin
Production Editor: Christopher Faucher
Copyeditor: Rachel Head
Proofreader: Tom Sullivan
Interior Designer: David Futato
Cover Designer: Kenn Vondrak
Illustrator: Kate Dullea

May 2021: First Edition

Revision History for the First Edition
2021-04-30: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Architecting Data-Intensive SaaS Applications*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors, and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Snowflake. See our statement of editorial independence.

978-1-098-10273-9

[LSI]

Table of Contents

CHAPTER 1

Data Applications and Why They Matter

In the last decade we've seen explosive growth in data, driven by advances in wireless connectivity, compute capacity, and proliferation of Internet of Things (IoT) devices. Data now drives significant portions of our lives, from crowdsourced restaurant recommendations to artificial intelligence systems identifying more effective medical treatments. The same is true of business, which is becoming increasingly data-driven in its quest to improve products, operations, and sales. And there are no signs of this trend slowing down: market intelligence firm IDC predicts the volume of data created each year will top 160 ZB by 2025,[1] a tenfold increase over the amount of data created in 2017.

This enormous amount of data has spurred the growth of *data applications*—applications that leverage data to create value for customers. Working with large amounts of data is a domain unto itself, requiring investment in specialized platforms to gather, organize, and surface that data. A robust and well-designed data platform will ensure application developers can focus on what they do best—creating new user experiences and platform features to help their customers—without having to spend significant effort building and maintaining data systems.

1 *https://blog.seagate.com/business/enormous-growth-in-data-is-coming-how-to-prepare-for-it-and-prosper-from-it*

We created this report to help product teams, most of which are not well versed in working with significant volumes of fast-changing data, to understand, evaluate, and leverage modern data platforms for building data applications. By offloading the work of data management to a well-designed data platform, teams can focus on delivering value to their customers without worrying about data infrastructure concerns.

This first chapter provides an introduction to data applications and some of the most common use cases. For each use case, you will learn what features a data platform needs to best support data applications of this type. This understanding of important data platform features will prepare you for Chapter 2, where you will learn how to evaluate modern data platforms, enabling you to confidently consider the merits of potential solutions. In Chapter 3 we'll explore design considerations for scalability, a critical requirement for meeting customer demand and enabling rapid growth. This chapter includes examples to show you how to put these best practices into action. Chapter 4 covers techniques for efficiently transforming raw data within the context of a data application and includes real-world examples. In addition to consuming data, teams building effective data applications need to consider how to share data with customers or partners, which you will learn about in Chapter 5. Finally, we will conclude in Chapter 6 with key takeaways and suggestions for further reading.

Throughout this report we provide examples of how to build data applications using Snowflake, a modern platform that enables data application developers to realize the full potential of the cloud while reducing costs and simplifying infrastructure.

NOTE

The Snowflake Data Cloud is a global network where thousands of organizations mobilize data with near-unlimited scale, concurrency, and performance.[2] Inside the Data Cloud, organizations unite their siloed data, easily discover and securely share governed data, and execute diverse analytic workloads. Wherever data or users live, Snowflake delivers a single, seamless experience across multiple public clouds.

2 *https://www.snowflake.com*

Data Applications Defined

Data applications are customer- or employee-facing applications that process large volumes of complex and fast-changing data, embedding analytics capabilities that allow users to harness their data directly within the application. Data applications are typically built by software companies that market their applications to other businesses. As you learn about some of the most common use cases of data applications in this chapter, you will get a sense of the breadth of this landscape. Truly, we are living in a time when most applications are becoming data applications.

Retail tracking systems, such as those used by grocery stores to track shopping habits and incentivize shoppers, are data applications. Real-time financial fraud detection, assembly line operation monitoring, and machine learning systems improving security threat detection are all data applications. Data applications embed tools, including dashboards and data visualizations, that enable customers to better understand and leverage their data. For example, an online payments platform with an integrated dashboard enables businesses to analyze seasonal trends and forecast inventory needs for the coming year.

As shown in Figure 1-1, data applications provide these services by embedding data platforms to process a wide variety of datasets, making this data actionable to customers and partners through a user interface layer.

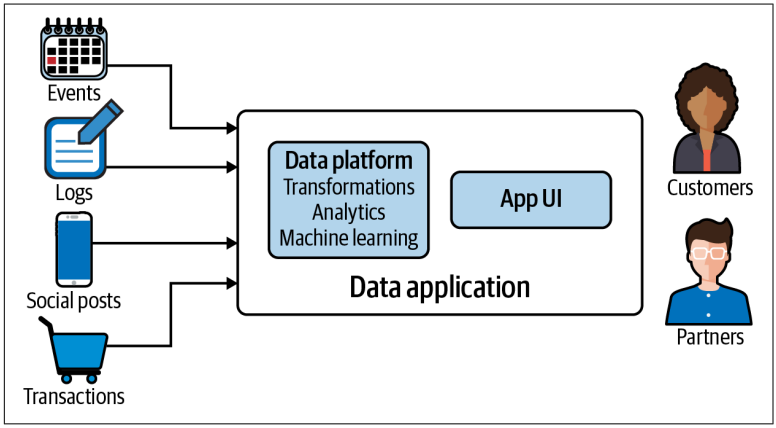

Figure 1-1. Data applications

In the following sections we'll review five of the most common use cases of data applications. For each case we will highlight key data platform considerations, which we will then cover in more detail in Chapter 2.

Common Data Application Use Cases

The use cases we will cover are:

Customer 360
: Applications in marketing or sales automation that require a complete view of the customer relationship to be effective. Examples include targeted email campaigns and generating personalized offers using historical and real-time data.

IoT (Internet of Things)
: Applications that use large volumes of time-series data from IoT devices and sensors to make predictions or decisions in near real time. Inventory management and utility monitoring are examples of IoT data applications.

Application health and security
: Applications for identification of potential security threats and monitoring of application health through analysis of large volumes of current and historical data. Examples include analyzing logs to predict threats and real-time monitoring of application infrastructure to prevent downtime.

Machine learning and data science
: Applications focusing on the training and deployment of machine learning models in order to build predictive applications, such as recommendation engines based on purchase history and clickstream data.

Embedded analytics
: Data-intensive applications that deliver branded analysis and visualizations, enabling users to leverage insights within the context of the application.

Customer 360

From clickstreams telling the story of how a user engages digitally to enriching customer information with third-party data sources, it is now possible to get a holistic, 360-degree view of customers.

Bringing together data on customers enables highly personalized, targeted advertising and customer segmentation, leading to more opportunities to cross-sell and upsell. Both through better understanding of customers and by taking advantage of machine learning, you can create compelling experiences to drive conversion.

The challenge with Customer 360 applications is dealing with the massive amount and variety of data available. Basic data, including contact and demographic information, can be purchased from third-party sources. As this data tends to be stored in customer relationship management (CRM) solutions, it is typically well structured, available as an export at a point of time or via an API. Interaction data shows how a customer interacts with digital content. This can include tracking interaction with links in marketing emails, counting the number of times a whitepaper is downloaded, and using web analytics to understand the path users take through a website. Interaction data is typically semi-structured and requires more data processing to realize its value.

Realizing value from customer data involves bringing together the various data types to run analysis and build machine learning models. To support these endeavors a data platform needs not only to be able to ingest all these different types of data but also to gain insights from the available data through analysis and machine learning. We will talk about data platform needs in this area in "Machine Learning and Data Science" on page 6.

IoT

IoT data applications analyze large volumes of time-series data from IoT devices, sometimes requiring near-real-time analytics. Enabled by the confluence of widespread wireless connectivity and advances in hardware miniaturization, IoT devices have proliferated across multiple industries. From connected refrigerators to inventory management devices and fleets of on-demand bicycle and scooter rentals, the IoT has created an entirely new segment of data, with spending in this sector estimated to have reached $742 billion in 2020.[3]

3 *https://www.idc.com/getdoc.jsp?containerId=prUS46609320*

A smart factory offers some good of IoT data applications.[4] Real-time sensor data can be transformed into insights for human or autonomous decision making, enabling automated restocking when inventory levels dip below a threshold and visualization of operational status to monitor equipment health.

A theme in IoT applications is the need to both gather data and relay that information to be consumed by larger systems. IoT devices use sensors to gather data which is then published over a wireless connection. We have all experienced the patchy nature of wireless networks, with dropped calls and unreliable internet connections. These problems exist in IoT networks as well, resulting in some data from IoT devices arriving out of chronological order. If an IoT data application is monitoring the health of factory equipment it is important to be able to reconstruct the timeline to detect and track issues reliably.

In addition to supporting semi-structured data and the ability to efficiently order time-series data, a data platform supporting IoT use cases must be able to quickly scale up to service the enormous amount of data produced by IoT devices. As IoT data is often consumed in aggregate, creation of aggregates directly from streaming inputs is an important feature for data platforms as well.

Machine Learning and Data Science

It comes as no surprise that as the volume of data has grown rapidly, so has the ability to leverage data science to make predictions. From reducing factory downtime by predicting equipment failures before they occur to preventing security breaches through rapid detection of malicious actors, data science and machine learning have played a significant role across many industries.[5]

As with the Customer 360 use case, data applications leveraging machine learning require ingestion of large amounts of different types of data, making support for data pipelines essential. Efficient use of compute resources is also important, as generating predictions from a machine learning model can be extremely resource-

4 *https://www2.deloitte.com/us/en/insights/focus/industry-4-0/smart-factory-connected-manufacturing.html*

5 *https://theiotmagazine.com/how-is-sensor-data-used-for-iot-predictive-analytics-d7e44668631c*

intensive. The elasticity of cloud-first systems (discussed in Chapter 2) can ensure that expensive compute resources are provisioned only when needed.

The development process for machine learning can benefit from significant amounts of data to construct and train models. A data platform with the ability to quickly and efficiently make copies of data to support experimentation will increase the velocity of machine learning development.

For data science and analysis, a data platform should support popular languages such as SQL to provide direct access to underlying data without the need for middleware to translate queries. External libraries for data analysis and machine learning can greatly streamline the process of building models, so support for leveraging third-party packages is also important.

Application Health and Security

Application health and security data applications analyze large volumes of log data to identify potential security threats and monitor application health. Many new businesses have been formed specifically around the need to process and understand log data from these sources. These businesses turn log data into insights for customers through application health dashboards and security threat detection. In the security domain, machine learning has improved malware classification and network analysis.[6]

The ability to rapidly act on data is a critical feature of data applications in this area. Thus, real-time, fast data ingestion is a key requirement for data platforms supporting application health and security applications. Delays in surfacing data for analysis represent time lost for identifying and mitigating security issues. Often, triage involves looking back to observe events that led up to a security incident. Being able to time travel and observe data in a previous state can help piece together what led to a security breach.

Much of the data related to application health and security comes from log files. These can take up a significant amount of space, especially if you want to be able to time travel to previous versions.

6 For more on this topic, see *Machine Learning and Security* by Clarence Chio and David Freeman (O'Reilly).

The ability to cheaply store this data while enabling analysis is another important data platform feature in this space.

The value of data applications in this space lies not only in enabling rapid identification of issues, but also the ability to act on findings when they occur. Integrating data applications with ticketing and alerting systems will ensure customers are notified in a timely fashion, and further integration with third-party services will allow for direct action to be taken. For example, if a data application monitoring cloud security identifies an issue with a compute instance, it could terminate it and then send an alert to the team indicating that the issue has already been taken care of.

Embedded Analytics

Customers rely on data from the applications they use to drive business decisions. Embedded analytics refers to data applications that provide data insights to customers from within the application.[7] For example, a point of sale application with embedded demand forecasting provides additional value to customers beyond the primary function of the application. Leveraging application data to provide these additional services enables companies building data applications to generate new revenue streams by selling these extended services and thereby differentiate themselves from competitors.

Without embedded analytics, application users are limited in the value they can get from their data. They may request exports of their data, but this is inferior to an embedded experience due to the loss of context when data is exported from an application. Application users then must interact with multiple systems: the data application and third-party business intelligence (BI) and visualization tools. Customers must also contend with the additional cost and delay of storing and processing exported data. Instead, a data platform that supports embedding of third-party tools for data visualization and exploration will enable users to stay within the data application. This lets them work with fresh data and reduces overhead in supporting exports from the data application.

7 For more on how Snowflake addresses the challenges product teams face when building embedded analytics applications, see the "How Snowflake Enables You to Build Scalable Embedded Analytics Apps" whitepaper.

Because customers access embedded analytics on demand, it is not easy to predict usage. An elastic compute environment will ensure that you can deliver on performance service-level agreements (SLAs) during peak load, with the added benefit that you will not pay for idle resources when load subsides. Data platforms that can scale up and down automatically to meet variable demand patterns will offload this burden from the data application team. You will learn more about different approaches for scaling resources in Chapter 3.

Data platforms that support embedded analytics applications need support for standard SQL and the ability to isolate workloads. Support for standard SQL will enable embedding of popular BI tools, reducing demand on product teams to build these tools in-house. The ability to isolate workloads from different customers is important to prevent performance degradation.

Summary

Data applications provide value by harnessing the incredible amount and variety of data available to drive new and existing business opportunities. In this chapter we introduced data applications and five major use cases where data applications are making a significant impact: Customer 360, IoT, application health and security, machine learning and data science, and embedded analytics.

With an understanding of the key requirements in each use case, you are now ready to learn what to look for when evaluating data platforms.

CHAPTER 2

What to Look For in a Modern Data Platform

In order to take advantage of the rapidly growing demand for data applications, product teams need to invest in data platforms to gather, analyze, and work with large amounts of data in near real or real time. These platforms must support different data types and structures, be able to interoperate with external tools and data sources, and scale efficiently to manage the demands of customers without wasting resources.

If your data platform does not support these capabilities, your engineering team will spend significant time developing and maintaining systems to service these needs, reducing the amount of resources available for application development. In this chapter you will learn what to look for in a modern data platform to ensure engineering effort can remain focused on building your product. We will dive into the use case–specific needs covered in "Application Health and Security" on page 7, as well as other areas of importance for data platform assessment. By the end of this chapter you will understand what features to look for in a data platform for building data applications and why they are important.

Benefits of Cloud Environments

It is difficult to meet the challenges of modern data applications with legacy, on-premises data platforms. It takes significant time and resources to bring an on-premises system online, requiring physical

machines to be purchased, configured, and deployed. With cloud environments you can bring an application online in minutes, and adding additional capacity is just as quick.

In addition to speed, cloud environments outperform on-premises solutions in scalability, cost, and maintenance. The virtually infinite capacity and elasticity of the cloud allows resources to be scaled up to meet demand for a much lower cost than expanding a data center —and cloud environments can also easily scale down when loads decrease, offering significant cost savings over the fixed capacity of on-premises systems. Additionally, cloud environments manage resources in a way that reduces the maintenance burden to a greater or lesser extent (depending on whether you choose a cloud-first or cloud-hosted solution, as described next).

Given the advantages and prevalence of cloud environments we will focus our discussion on the trade-offs associated with different cloud approaches, rather than covering the outdated on-premises approach.

An important difference in cloud-based approaches is whether they're cloud-hosted or cloud-first. A *cloud-hosted* application model is one where software designed for on-premises systems is run in the cloud. In this case you leverage cloud computing instances but assume responsibility for the software, operating system, security, and some infrastructure, such as load balancing. This is preferable to the on-premises model in that you don't pay for maintaining physical hardware, but inferior to cloud-first as you are still burdened with significant maintenance and limited in your ability to take advantage of cloud features such as scalability and elasticity.

In a *cloud-first* application model software is built specifically to take advantage of the benefits of the cloud, such as having access to virtually infinite compute and storage and enjoying true elasticity. In this scenario the provider of the software assumes the burden of ensuring the entire stack is operational and provides additional services to automatically allocate resources as needed.

Cloud-First Versus Cloud-Hosted

Cloud-first environments maximize the benefits of the cloud, such as offloading a significant portion of the maintenance burden when building and operating data applications. Because the cloud-hosted model brings an on-premises architecture to a cloud environment,

many of the shortcomings of on-premises systems exist in the cloud-hosted model as well.

These shortcomings stem from a fundamental difference in cloud-hosted and cloud-first solutions: which party takes responsibility for managing cloud resources. In the cloud-hosted model this is the responsibility of the developer, while in the cloud-first model it's up to the data platform. The following are some areas where this trade-off is particularly important to consider for data applications.

Elasticity

Cloud-first environments manage resource scaling, whereas in cloud-hosted systems resource allocation and scaling must be managed by developers. That is, in a cloud-hosted environment developers need to design processes for adding or removing compute resources as needed to service different workloads across tenants. In a cloud-first environment these resources will be automatically allocated as needed, eliminating the need to design a separate process and fully taking advantage of the elasticity of the cloud.

When modifying resource allocations it is necessary to rebalance workloads, either to distribute them to take advantage of increased capacity or to consolidate them onto a smaller set of resources. Cloud-first environments can handle load balancing automatically, even as the number of compute resources changes, whereas in the cloud-hosted approach developers have to manually adjust the load balancers or build and maintain software to automate the process.

Scaling workloads while not disrupting ongoing processes is a challenging problem. In a cloud-hosted environment not only do you have to provision instances in response to demand, but you have to do so in a way that minimizes impact to users. For example, if a machine learning workload consumes all the available resources and needs more, you will not only need to provision additional nodes but also manage redistributing your data and workloads to make use of the additional nodes. Cloud-first environments handle this process for you, again saving significant complexity and cost by managing processes you would have to design and operate yourself in a cloud-hosted system.

Availability

Major cloud platforms have the ability to deploy compute instances in geographical regions all over the world. This provides the benefits of better latency for users around the globe, and a failsafe in the event of a regional disruption. Due to the additional cost and complexity, most companies choose not to take on the task of a multi-region deployment themselves, incurring the risks associated with a single-region deployment. Data platforms that seamlessly support multi-region operation greatly reduce these costs while providing improved reliability.

In cloud-first environments availability across geographic regions can be built in such a way that if one region is experiencing service issues the platform seamlessly switches over to another region with minimal disruption to users. This kind of fallback across zones is a significant undertaking to design for a cloud-hosted environment, requiring design and maintenance of systems to detect a regional service issue and to migrate workloads to new resources.

Choice of Cloud Service Providers

One of the first questions you will be confronted with when developing data applications is what cloud service provider to use. Amazon, Microsoft, and Google are the primary providers in this space, and it can be hard to decide among them. Additionally, once you've made a choice, it is difficult to change providers or interoperate with customers using another provider without significant technical lift.

One approach is to build a data platform from scratch, using custom code that can be ported across different providers. While cloud service vendors provide basic components such as cluster compute and blob storage, there is significant work required to design, build, and maintain a data platform that will meet the needs of modern data applications. While it is possible to port the associated code for these systems across providers, the burden of managing systems across different providers remains. In addition, this approach greatly limits the cloud services you can take advantage of, as code portability requires using only the most basic cloud components.

Ideally a data platform would be cloud service provider–agnostic and enable working across cloud providers. Besides reducing the maintenance burden, this would also give you the advantage of being able to fall back to another provider if one was experiencing

an outage. In addition, not getting locked in to a single provider will enable you to onboard new customers without concern for which cloud service provider they use.

Support for Relational Databases

Relational databases are needed to support embedded BI tools, visualization workloads, and analytical users, but recently a lot of focus in the big data arena has been on NoSQL databases. In this section we will take a look at how relational databases serve data applications, with some history along the way to understand the evolution of relational and NoSQL databases.

To begin, it's important to understand the role of semi-structured data. Semi-structured data is data that does not conform to a rigid schema. JSON is an example of semi-structured data, composed of field/value pairs that enable representation of primitive values as well as hierarchical information. The flexibility of semi-structured data is important for representing machine-generated data, such as data from IoT sensors and mobile devices, whose schema evolves over time. As we discussed in "Application Health and Security" on page 7, semi-structured data is an extensive source for data applications, not just for IoT but for Customer 360 and other use cases as well.

As the prevalence of semi-structured data skyrocketed over the last decade, so did the need to process it. Fully supporting semi-structured data requires both language support for representing the data and query performance on par with that of structured systems —features relational databases didn't provide at the time. To fill this gap, NoSQL databases emerged and rapidly gained popularity due to their ability to handle fast writes for large amounts of semi-structured data.

Benefits of Relational Databases

Relational databases have since evolved to include support for semi-structured data. While NoSQL variants continue to excel at high-volume, fast writes of this kind of data, relational databases have an advantage in their ability to express analytical queries. NoSQL databases rely on procedural languages to query data. This puts the burden of optimizing queries on the programmer. SQL is declarative,

producing more concise code that is easier to write and maintain. The task of transforming SQL to something that will run efficiently is handled by optimizers, allowing users to focus on analysis without having to worry about optimizing their code.

SQL also has the benefit of decades of development as the de facto language for data analysis, boasting millions of users and a robust ecosystem. As a result, data in relational databases can be quickly leveraged by analytics professionals across a multitude of industries.

To work with data in NoSQL databases, you need to learn a first-class programming language such as Java, or a boutique language developed specifically for this purpose, such as Pig or Hive. While Hive is declarative and SQL-like, using it requires a middleware layer to translate queries to the underlying NoSQL engine. These additional complications present a barrier to leveraging data stored in these systems, limiting use to programmers.

Atomicity, consistency, isolation, and durability (ACID) transaction guarantees are another significant benefit provided by relational databases. When an update occurs in a NoSQL system it must propagate to other nodes, introducing lag. As a result, reads can return stale data, resulting in incorrect modeling and analysis. ACID guarantees in relational databases ensure that data is consistent, preventing stale data from being returned from queries.

Finally, relational databases are still significantly more widely used than NoSQL databases, making it highly likely that data applications will need to interoperate with relational data sources.[1] Given the way relational platforms have evolved to better support big data needs and their popularity, it is clear that relational data platforms that support SQL are a critical component of modern data platforms.

Separation of Storage and Compute

Historically, database systems have tightly coupled compute and storage to ensure fast transactions due to latencies in networking and storage in legacy systems. In recent years networking and storage bandwidth has increased dramatically, eliminating these bottle-

1 *https://db-engines.com/en/ranking_trend*

necks.[2] As a result, modern data platforms can take advantage of separating compute and storage for improved reliability, scaling, and cost. The elasticity of cloud-first environments enables compute and storage to grow or shrink on demand, a benefit best realized by keeping these systems decoupled.

Consider the reasons either of these resources might scale up. If your data application has a sudden spike in demand, you want additional compute capacity, but additional storage is not necessarily needed. Likewise, as the data in your system grows, you primarily want storage to scale, not compute, outside of the resources needed for ingestion. In a system where these resources are coupled and cannot scale independently you will incur additional cost for resources you do not need.

To better understand these trade-offs, let's look at some examples of cloud data architectures. Figure 2-1(A) shows an example of a data system with storage for the data and compute to access it tightly coupled.

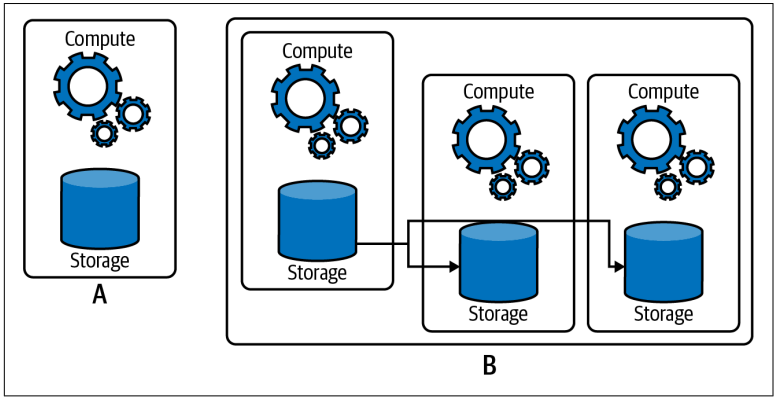

Figure 2-1. Coupled storage and compute: (A) single instance, (B) scaled to meet additional demand

In Figure 2-1(B), additional instances have been provisioned to meet an increase in compute demand. There isn't a change in the amount of storage needed, just a change in the number of users or processes accessing the data. But because storage and compute are

2 *https://transition.fcc.gov/national-broadband-plan/broadband-performance-paper.pdf*

tightly coupled, expanding compute capacity also requires expanding storage capacity. This has a few undesirable side effects:

- The additional storage is unused, resulting in spending on unnecessary resources.
- The data from the first storage instance must be replicated to the new storage instances.

Replicating data to the additional storage instances can require system downtime to ensure data consistency. Every time additional compute is required, the downtime for replication is incurred.

In comparison, Figure 2-2 shows how these needs could be handled in a system where compute and storage are decoupled.

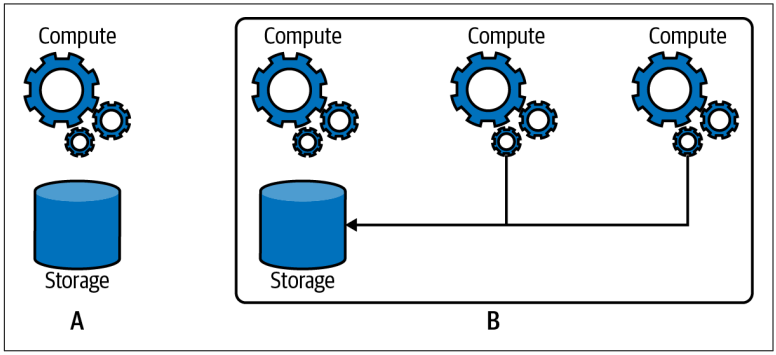

Figure 2-2. Decoupled storage and compute: (A) single instance, (B) scaled to meet additional demand

With storage and compute decoupled, compute can grow as needed. This eliminates spending on unused resources and the downtime required for replication. In Chapter 3 you will learn how multiple resources can access the same underlying data, realizing the advantages of decoupled storage and compute.

In addition to the cost and scaling benefits, separating storage from compute protects you from data loss. In systems where storage and compute are coupled, data can be lost as a result of compute instance failure. Decoupling these systems will avoid this scenario—instead of storing data on compute instances, you can store it in durable, fast cloud storage.

Data Sharing

Data sharing creates opportunities for data providers and consumers. For providers, sharing data can provide new sources of revenue and opportunities to get feedback from users. For consumers, data sharing enables enrichment of existing data to better inform business decisions and expand analysis capabilities.

Data application builders should look for data platforms that provide the ability to share data easily and avoid creating copies of data. Traditional methods of data sharing, such as file transfers, can be costly and require additional overhead to manage. Creating copies of data to share entails paying to store duplicate data, increasing costs, and once data is copied out of the data producer's system it is no longer fresh, requiring systems to repeatedly copy and handle new versions of the data to stay up to date. Sharing through data copy also carries the overhead of building and maintaining systems to incorporate copied data into the consumer's system. We'll discuss data sharing in depth in Chapter 5, covering both traditional and modern approaches.

Workload Isolation

There is an inherent contention for resources in data systems. These systems have to support a variety of workloads, including data ingestion, user queries, and analytical processes. When these processes share the same hardware they will compete for resources, resulting in performance degradation. For example, if data ingestion and user queries share the same hardware, a large data ingestion event can monopolize compute resources, resulting in poor performance for user queries. This can be costly; contention for resources can introduce reliability issues and difficulty in guaranteeing performance SLAs.

Workload isolation refers to separating computational processes to reduce or eliminate resource contention. This is a critical component for data platforms supporting data applications, especially because the challenges of resource contention expand in multitenant environments. In data applications where a variety of workloads are spread across hundreds of customers, it is important not only to isolate ingestion and analytical workloads but to isolate different customer workloads from one another as well.

In addition to the performance benefits, workload isolation enables you to provide different tiers of service, attracting both enterprise and startup customers. For example, you might offer dedicated resources to larger customers while providing a lower tier of service for self-service customers that operates on a shared resource pool. In this case workload isolation can enable low-cost options so start-ups can pilot your data application without having to make a significant investment, while also guaranteeing performance for enterprise customers.

The ability to keep track of costs per tenant is another advantage of workload isolation. In a system where resources are shared across tenants it can be difficult to determine how to split costs among customers. With workload isolation providing the opportunity to dedicate resources on a per-customer basis, billing becomes more straightforward. While you will still need to do the extra calculations to split billing if you offer a shared resource tier, this is likely to represent a smaller portion of overall costs compared to the dedicated resources for enterprise customers.

Figure 2-3 shows an example of workload isolation in a data application with two enterprise customers, E1 and E2, and three self-service customers, A, B, and C. The enterprise customers each have dedicated compute resources that are separate from those shared by the other tenants. Customer E1 has a larger volume of data and additional processing needs compared to E2, so extra resources have been allocated for this customer. Customers A, B, and C are on a shared service tier. These could be start-up customers with lower SLAs that can trade the performance of dedicated compute for the lower cost of sharing resources with other customers.

With workloads isolated as in Figure 2-3, customers E1 and E2 are protected from surges in demand arising from workloads run by other customers. It doesn't matter how many compute resources A, B, and C use; E1 and E2 each have their own dedicated resource pool to draw from.

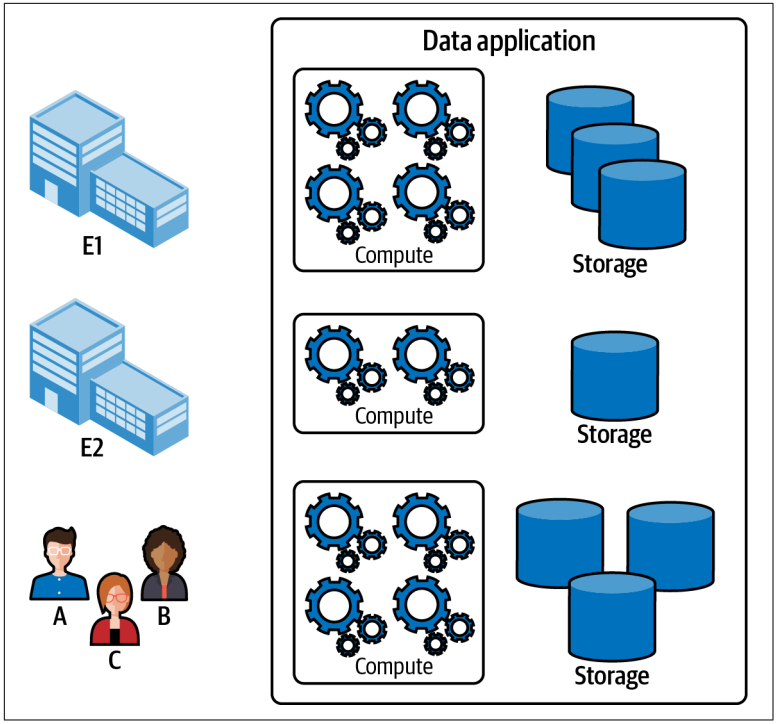

Figure 2-3. Workload isolation in a multi-tenant environment

To get a sense of how different types of workloads are isolated, let's take a look at the workloads for tenant E1, as illustrated in Figure 2-4.

The compute resources allocated to tenant E1 have been split across three different workloads: a data ingestion process, a machine learning model, and user queries. Because of workload isolation, these processes have separate, dedicated resources, eliminating the issue of workload contention.

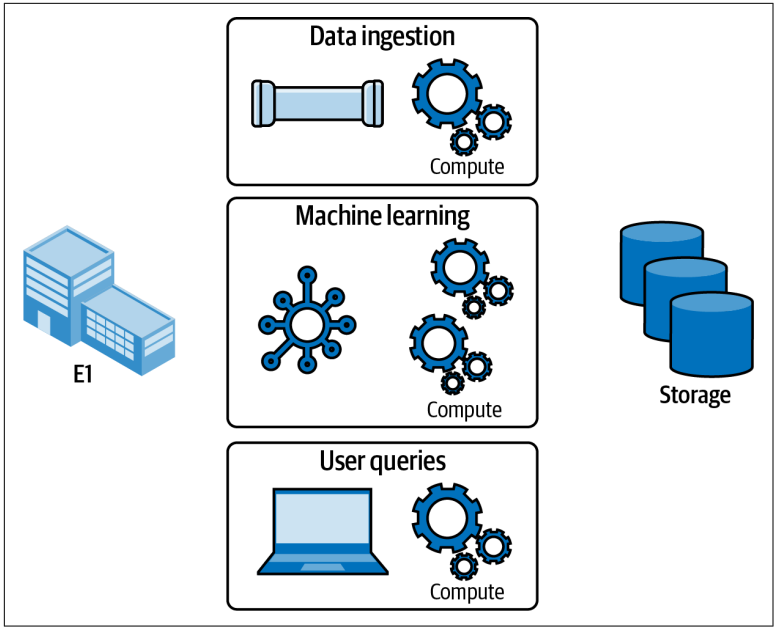

Figure 2-4. Different workloads for tenant E1

Additional Considerations

Reliability and extensibility are two additional areas to consider when evaluating data platforms. With a SaaS business, when the application goes down business grinds to a halt until service is restored. It is therefore crucial to ensure your chosen data platform has planned for potential reliability issues and provides mechanisms to keep the system online should problems occur. Additionally, to ensure evolving customer needs can be met, data platforms should provide the ability to extend their capabilities by taking advantage of third-party services and custom code.

Reliability

As with any technology, ensuring reliability is an important aspect of data platforms. Reliability issues in the cloud can include compute instances terminating unexpectedly, data lost in transit over the network, lack of availability of compute resources due to high demand, or service provider outages. Modern data platforms must be designed defensively for these issues to avoid unplanned downtimes that result in revenue loss.

Monitoring, alerting, and automatic repair are key defensive mechanisms to ensure reliable cloud systems. Cloud platforms provide services for monitoring components and alerts based on these monitors that can be configured to identify system failures. When a component fails, an alert can be dispatched to a third-party system to notify the product team of the issue. While alerting is important, a modern data platform should also enable automatic repair where possible to reduce the need for manual intervention.

A disaster recovery plan spanning multiple geographic regions and, ideally, multiple cloud providers reduces the likelihood of data loss and the length of downtime in the event of an outage in one service or region. Replication of data in different geographical regions will also help prevent data loss.

Extensibility

Another important aspect of a modern data platform is the ability to leverage third-party resources and custom code for working with data. SQL is a powerful tool for analysis, but there are uses for which other languages or tools may be better suited. In addition, there are many third-party libraries offering packaged functionality that customers may want to use, such as for financial modeling and machine learning. Sending a request to an external system to acquire additional data or perform an analysis is another frequent need when working with data. In this section we will look at a few ways cloud platform functionality can be extended to meet these needs.

User-defined functions (UDFs)

A UDF is a function written in either SQL or a procedural language such as Python, JavaScript, or Java. These functions are created by programmers and registered with the data platform, which enables the functions to be used by others. UDFs are called as part of a SQL statement to produce a value or a relation given a set of inputs. An example of a UDF is a function that calculates mortgage interest given an interest rate, loan term, and amount. The UDF encapsulates this code so it won't have to be rewritten every time the application needs to perform the calculation. UDFs also support performing operations on tables, such as create, read, update, or delete (CRUD).

Stored procedures

Stored procedures are another vehicle for extending platform capabilities. These are SQL subroutines that are stored in relational databases, allowing them to be accessed by applications. Stored procedures are particularly useful if you want to dynamically generate SQL or if you need to perform CRUD operations.

External callouts

Third-party services offer an abundance of tools that can be leveraged when working with data. For example, an application may make a request to a geocoding API to convert an address to a latitude/longitude coordinate, or call out to a machine learning model hosted outside of the data platform to generate a prediction.

Summary

In this chapter we highlighted essential elements of a modern data platform that take advantage of advances in data systems and the benefits of cloud services. Taken together, these elements reduce the burden on product teams building data applications, improve the customer experience, and streamline costs.

A modern data platform should include:

- A cloud-first environment that is cloud platform–agnostic
- Support for semi-structured and structured data
- ACID guarantees and SQL support
- Separation of storage and compute
- Data sharing without copying data
- Workload isolation
- Extensibility
- Robust disaster recovery and resiliency mechanisms

Advances in computing have enabled a revolution in the design of data systems, significantly reducing barriers of cost and scalability. With an understanding of how to evaluate modern data platforms, in the next chapter we'll move on to discussing how to take advantage of these platforms to build scalable data applications.

CHAPTER 3

Building Scalable Data Applications

Scalability is a requirement for successful data applications. A product that scales well can quickly onboard new customers, enable customers to run new workloads without impacting the performance of others, and take advantage of the elasticity of the cloud to keep costs in check. By thinking about scalability from the beginning you can avoid bottlenecks and costly redesign efforts that can blunt product growth.

In Chapter 2 you learned about important features of a modern data platform. In this chapter you will learn how to best leverage those features to design scalable data applications. We will begin with an overview of the key design considerations for building data applications that scale. The rest of the chapter will dive into best practices and real-world examples to support these considerations. At the end of this chapter you will understand how to make the best use of Snowflake's features for designing scalable data applications.

Design Considerations for Data Applications

As discussed in Chapter 2, support for multiple tenants is a foundational requirement for data applications. Underlying this requirement are three components: storage, compute, and security. In this section we will present design patterns and examples covering each of these areas. Data application customers will be referred to as

"tenants" and individual users associated with a customer will be referred to as "users."

This section will include examples using Snowflake's *virtual warehouses*—clusters of compute resources that provide resources such as CPU, memory, and temporary storage to perform SQL operations.[1] We will also introduce Snowflake's multi-cluster warehouses, which offer the capabilities of virtual warehouses with a larger pool of compute resources, including built-in load balancing.[2]

Design Patterns for Storage

Data applications need to ensure that customers can't see each other's data, but the details of how this is accomplished depend on a variety of factors that will vary by application. In this section we will discuss different methods to isolate data between tenants and processes and provide recommendations on when to use each method.

Multi-tenant tables

Multi-tenant tables combine all tenants into a single set of database objects. In this scenario, all tenants belong to the same table, with row-level security applied to ensure isolation. This greatly reduces the number of objects you have to maintain, which can make it easier to support many more tenants without increasing your operational burden.

Snowflake implements multi-tenant tables as illustrated in Figure 3-1. Source data is exposed in a single table shared across all tenants, with an application layer that ensures each tenant can only access the data it has permission to access.

1 *https://docs.snowflake.com/en/user-guide/warehouses.html*

2 *https://docs.snowflake.com/en/user-guide/warehouses-multicluster.html*

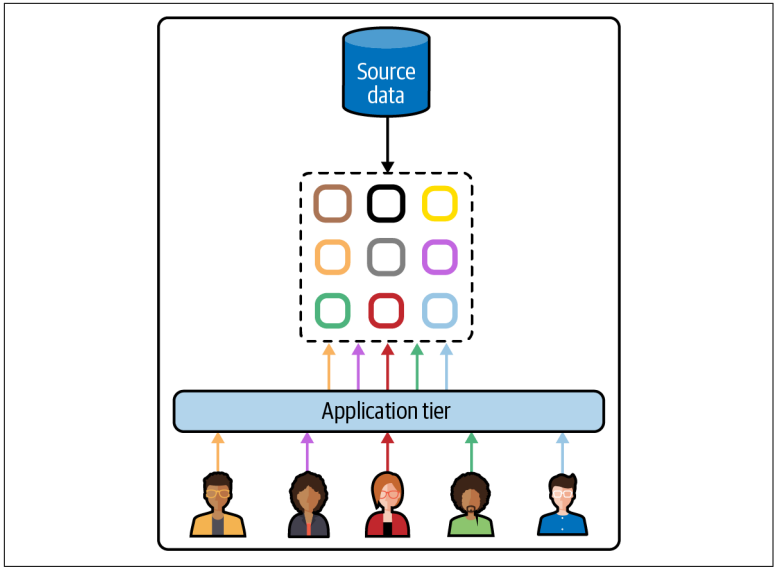

Figure 3-1. Snowflake multi-tenant table

An important consideration for multi-tenant tables is ensuring performance does not degrade as the tables grow to significant size. Without optimization, table growth will result in slower access times for interacting with data. To address this issue, it is recommended that you cluster data in multi-tenant tables based on the tenant ID rather than by date. This table clustering scheme ensures that as each tenant queries its slice of the data, Snowflake has to scan as few files as possible, resulting in increased performance.

Object per tenant

In an object per tenant model, underlying database instances are shared but database objects are allocated to separate tenants. For example, tenants may have their own databases, schemas, and tables but be commingled in a single database instance. Role-based access control (RBAC) is used to isolate tenants to their respective objects. While the overhead is lower than with an account per tenant model, discussed next, this approach also can quickly become unwieldy to manage as the customer base grows.

The object per tenant model is commonly used when the shape of the data for each tenant is different. In cases where requirements demand greater data separation than row-level security, object per

tenant can be an effective choice. Figure 3-2 depicts how this option is designed in Snowflake.

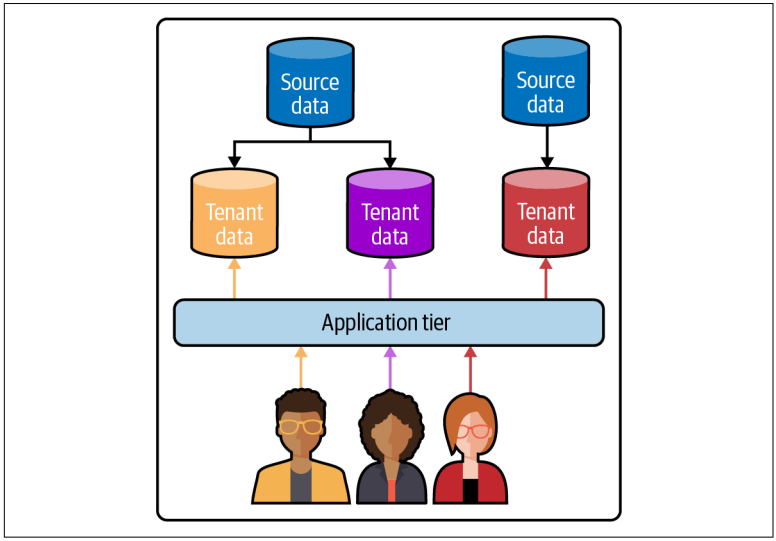

Figure 3-2. Snowflake object per tenant design pattern

Account per tenant

Another way to provide data isolation is to create a new database instance for every tenant. This ensures complete separation between tenants, which is important for applications with contractual or regulatory requirements.

While this approach guarantees complete isolation, it comes with higher overhead and maintenance costs due to additional administrative objects to maintain for every instance. For example, for every tenant you will need to maintain a separate database instance. As the number of tenants increases, so will the number of resources needing to be upgraded, monitored, and debugged, leading to significant maintenance and support costs. Therefore, consideration should be given to the number of tenants you will need to support with this method.

Snowflake makes the creation of instances simpler, because an *instance* in Snowflake is just a logical Snowflake account which can be created with a SQL statement. There is still some administrative overhead with this approach, but not as much as with a physical database instance. Figure 3-3 shows the Snowflake account per ten-

ant design pattern. In this case, each tenant has a dedicated database instance associated with its account.

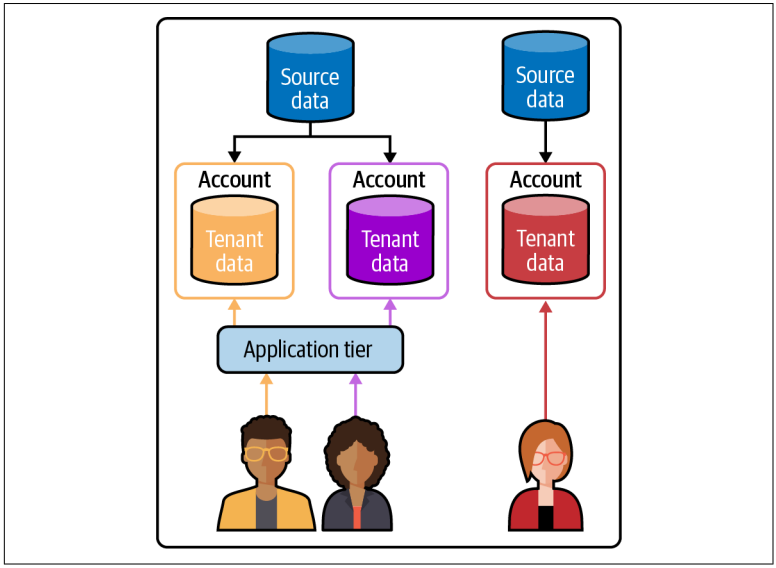

Figure 3-3. Snowflake account per tenant design pattern

Design Patterns for Compute

Suboptimal design for compute in multi-tenant environments can lead to poor query performance, delays in ingesting new data, and difficulties servicing the needs of different tiers of customers. In this section we will discuss different methods for allocating compute resources to meet performance requirements.

Compute scaling is discussed along two axes: vertical and horizontal. Vertical scaling refers to the ability to provide more powerful resources to perform a task. If a customer runs a complex task, vertical scaling can improve the runtime. In cloud platforms, this involves provisioning different types of compute instances with more powerful specifications.

While vertical scaling can be used to make a task run faster, horizontal scaling can increase the number of tasks that can be run simultaneously, such as when many users access the platform simultaneously. Another example is a data ingestion process onboarding a large dataset, where adding additional nodes by scaling out horizontally improves parallelization, resulting in faster data ingestion.

Horizontal scaling is implemented by changing the number of compute instances available under a load balancer.

Overprovisioning

One approach to keeping up with compute demand is to provision an excess of compute resources in anticipation of increased or variable demand. Having additional compute resources available means you can quickly scale out capacity without having to wait for a new instance to be provisioned.

Overprovisioning relies on the ability to predict demand, which can be tricky. Consider the impact of COVID-19 resulting in sudden, enormous demand for video conferencing. The impossibility of predicting such an event could lead to disruption in services for customers.

Additional drawbacks to overprovisioning include cost and load balancing. If demand is lower than forecast, it's costly to pay for idle resources. If demand is higher than forecast, there is a poor customer experience as performance will degrade. Furthermore, when scaling out capacity, existing jobs need to be reorganized to balance the load across the additional resources. This can be challenging to accomplish without impacting existing workloads.

Autoscaling

Instead of trying to predict demand, *autoscaling* will increase the amount of compute available as demand rises and decrease it as demand subsides. Snowflake's multi-cluster, shared data architecture manages this for you by autoscaling within a configurable number of minimum and maximum clusters and automatically load balancing as the number of users changes.

When it comes to scaling, consideration needs to be given to tradeoffs in cost, resource availability, and performance SLAs. With multi-cluster warehouses you can choose to provide dedicated compute resources for tenants that need them, and for others a pool of shared resources that can autoscale horizontally when more tenants are on the system, enabling you to easily provide separate tiers of service. For example, customers who pay more could each be given their own more powerful warehouse, while lower-paying customers could be pooled onto a smaller, cheaper warehouse that can autoscale up to a maximum number of clusters when more customers

are simultaneously using the application. In this case, balancing loads across the scaled compute capacity must be managed as well. Snowflake has built-in load balancing that handles this out of the box with a simple configuration.

Workload isolation

As discussed in Chapter 2, workload isolation is important to ensure good performance in multi-tenant data systems. Isolating workloads also helps protect against runaway processes or bad actors. For example, if a single compute instance were shared among several tenants, a rogue process could disrupt all the tenants. With separate instances, the instance with the rogue process could be shut down without disrupting other workloads.

Different workloads have different compute needs, making workload isolation attractive for separating synchronous workloads from asynchronous ones, isolating simple workloads from more complex workloads, and separating data processing tasks from analytical tasks.

The virtual warehouse model provided by Snowflake achieves workload isolation by enabling multiple compute instances to interact with the same underlying dataset, as shown in Figure 3-4. Workloads for ingestion, transformation, and different tenants operate within separate compute environments, enabling resources to be sized independently. This allows workloads to consume as many or as few resources as required, while also ensuring the different workloads don't impact the performance of others. Tenants aren't affected by continuous data ingestion and transformation, and synchronous and asynchronous workloads can be isolated.

Separate virtual warehouses can be provisioned for data processing and analytical workloads to allow these processes to work in parallel without impacting each other's performance. All processes interact with the same data, with guaranteed consistency provided by Snowflake's cloud services layer.[3]

3 *https://www.snowflake.com/blog/snowflake-challenge-concurrent-load-and-query*

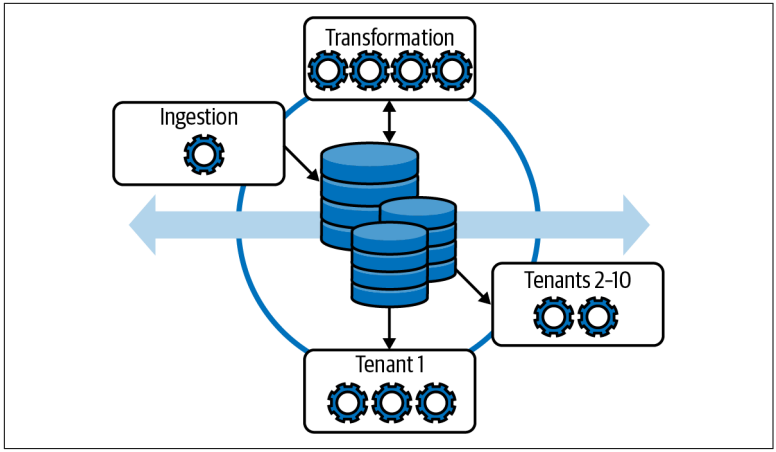

Figure 3-4. Workload and tenant isolation in Snowflake

However, with multiple compute instances comes the need to manage access across tenants. Snowflake provides configurable compute privileges, enabling data applications to determine which compute instances customers have access to and the types of operations they are allowed to perform. We'll discuss this and other security considerations in the next section.

Design Patterns for Security

With security breaches frequently in the news, you should expect customers to have concerns about the security of their data. Security features in a multi-tenant data platform should include guarantees for regulatory and contractual security requirements, as well as managing access to data and compute resources.

Access control

Within a multi-tenant system it is important to have some way of granting and restricting access for different users. *Access control* broadly refers to the mechanisms systems put in place to achieve this goal. In this section we will talk about two types of access control: role-based (RBAC) and discretionary (DAC).

It is useful to think of access management in terms of relationships between users and objects in the system. In a data application, objects include databases, tables, configuration controls, and compute resources. Relationships between users and objects are set by

privileges generally falling into the categories of *create*, *view*, *modify*, and *delete*. As it is typical for a user to have more than one of these privileges, the grouping of multiple privileges into a role used to control access is common. This is the RBAC model, shown in Figure 3-5.

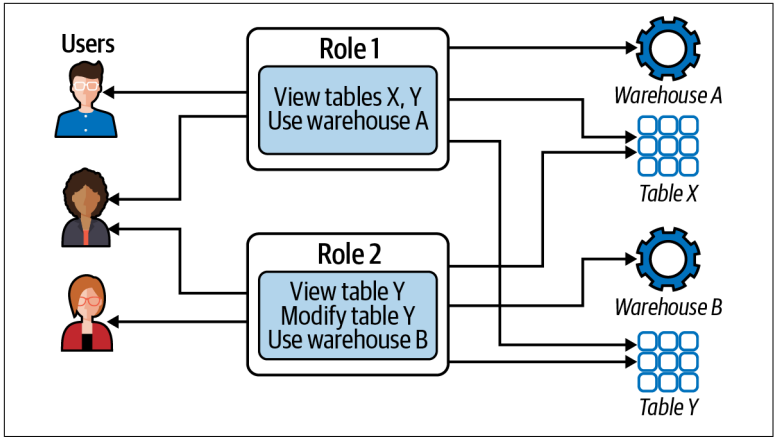

Figure 3-5. RBAC example in Snowflake

As depicted in Figure 3-5, Role 1 and Role 2 each encapsulate multiple privileges for interacting with the available assets, and each individual user may be assigned one or both roles.

In addition to typical access controls for database objects, Snowflake includes warehouses and other entities in its set of *securable objects*, or elements that can have access constraints applied to them.[4] This can save data application developers significant overhead when setting up permissions for tenants. The alternative is a patchwork of permissions across different services, such as database grant permissions for relational components and specific cloud-based controls for granting permission to interact with blob storage. Encapsulating these lower-level permissions in this way streamlines permission management and reduces the chances of omissions and mistakes.

In a multi-tenant system where data can be shared, it is helpful to enable data creators to specify who should have access to their objects. This is where DAC comes in, where object owners can grant access to other users at their discretion. Snowflake combines these

4 *https://docs.snowflake.com/en/user-guide/security-access-control-overview.html*

models, resulting in object owners providing access through roles that are assigned to users.

One aspect to keep in mind when handling permissions is that it's best to minimize the spread of privileges to a given object across several roles. Limiting access to a given object to a single role reduces overhead if there is a need to modify that permission in the future. A role hierarchy can then be used to create combinations of privileges, as in the example depicted in Figure 3-6.

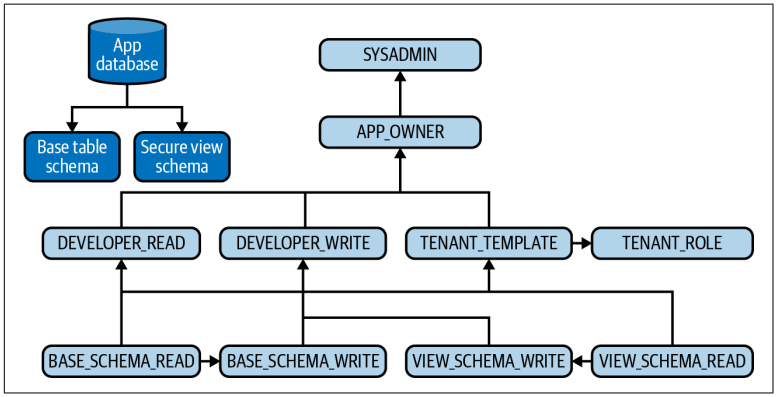

Figure 3-6. Role hierarchy and privilege inheritance

In the upper-left corner of Figure 3-6 you can see that the application database is organized with two different schemas: a schema of base tables and a scheme of secure views. It's good practice with the multi-tenant table approach to isolate secure views into their own schema, and this is also important for defining the role hierarchy.

You begin by creating the roles associated with the database schema permissions, shown in the bottom row. Functional roles, shown in the middle row, associate a user's function with the appropriate database permissions. For example, the `DEVELOPER_READ` role includes the `BASE_SCHEMA_READ` and `VIEW_SCHEMA_READ` roles.

Roles can be granted to other roles to create a hierarchy of inherited privileges, as we saw with `DEVELOPER_READ`. Notice also that `BASE_SCHEMA_WRITE` inherits `BASE_SCHEMA_READ`, further simplifying the permissions hierarchy by including read access with write access.

Because tenants should only have access to the secure views, the `TENANT_TEMPLATE` role is only granted the `VIEW_SCHEMA_READ` role. The

TENANT_ROLE inherits the permissions of TENANT_TEMPLATE and can be assigned to system tenants. With this inheritance, any future changes made to the TENANT_TEMPLATE role will automatically propagate to all tenants.

All functional roles are granted up the privilege chain to the APP_OWNER and finally the SYSADMIN, to ensure the administrator role has access to everything in the system.

Auditing

The ability to audit changes in access controls is critical for assessing system vulnerabilities, tracking down the source of corrupt data, and complying with regulations such as the European Union's General Data Protection Regulation (GDPR). In addition, many industries require auditing capabilities to do business; for example, if you hope to market your data application to healthcare or finance organizations, this is a critical requirement in a data platform. Snowflake meets this need with robust auditing of user permissions, logins, and queries, which can be accessed through logs.

Access and authorization controls

Another important area of security is ensuring the connections between the application tier and the underlying data platform are secure. Considerations in this space include authentication, encryption management, and secure network design.

To guarantee a secure connection between the application and the Snowflake Data Cloud, you can use AWS[5] or Azure PrivateLink.[6] These services allow you to create direct, secure connections between Snowflake and your application tier without exposure to the public internet. Snowflake allows connections from any device or IP by default, but you can limit access using allow and block lists.

Snowflake provides a variety of options for user authentication, including OAuth, multifactor authentication (MFA), key pair authentication and rotation, and single sign-on (SSO). Having these services built in removes a significant burden from product teams,

5 *https://docs.snowflake.com/en/user-guide/admin-security-privatelink.html*

6 *https://docs.snowflake.com/en/user-guide/privatelink-azure.html*

as providing support for even one of these methods is a substantial engineering undertaking.

As an example, consider a data application using OAuth to generate a secure token to access Snowflake. With OAuth, credentials do not have to be shared or stored, eliminating the need to build secure credential sharing and storage into your data application. Key pair authentication is another option for authentication where username/password credentials do not need to be explicitly shared; instead, a private key pair retrieved from a key vault can be used to control access.

Summary

Ensuring your approach to storage, compute, and security will meet the demands of the ever-changing data landscape is fundamental to building successful data applications. In this chapter you learned how to take advantage of the virtually infinite storage and compute resources of cloud platforms to create scalable data applications. We discussed different approaches to storage, including the multi-tenant table, object per tenant, and account per tenant models. You also learned about approaches to optimizing compute resources, including using autoscaling to provision resources in response to demand instead of attempting to predict demand. With an understanding of the trade-offs and use cases for each approach, you can make informed design decisions.

Taking advantage of the scalability of the cloud requires an approach to security that will scale as well. With Snowflake, creating a role hierarchy to manage permissions and coupling DAC and RBAC result in robust and flexible access control while keeping permissions management manageable. To control access to Snowflake a variety of secure user authentication modes are provided, as is support for securely connecting with application tiers on Azure and AWS.

CHAPTER 4

Data Processing

Data applications provide value by processing large volumes of quickly changing raw data to provide customers with actionable insights and embedded analytical tools. There are many ways to approach data processing, from third-party tools and services to coding and deploying bespoke data pipelines. A modern data platform should support all of these options, giving you the power to choose which best meets your needs. In this chapter you will learn how to assess the trade-offs of different data processing methods, providing the necessary understanding to make informed choices about working with the tooling provided by data platforms.

We will start with an overview of design considerations for this space, highlighting the elements you should consider when architecting data processing pipelines as part of a data application. Then we'll cover best practices and look at some real-world examples of implementing these practices with Snowflake's Data Cloud.

Design Considerations

Data processing is a sizable task that needs to be done in a way that is very low latency, low maintenance, and does not require manual intervention. A data platform that can meet this challenge will enable product teams to focus on application development instead of managing ingestion processes, and will ensure that users get insights as quickly as possible. The considerations presented in this section will guide you as you consider how to approach data processing.

Raw Versus Conformed Data

Raw data is information stored in its original format. JSON stored as a document is an example of raw data. As mentioned in Chapter 2, relational systems can now store and query this kind of raw, semi-structured data, so it is already in a format that is usable without having to be transformed into a tabular structure. While it is possible to query raw data, the queries are less performant than those run against conformed data.

Conformed data is information that fits a specific schema, requiring transformation of raw data. The motivation to conform data comes from the performance improvements and interoperability advantages of tabular structures. Data that is organized is easier for query engines to work with, resulting in better speed and performance than that achieved by queries run against raw, semi-structured data. In addition, many data analysis tools require structured, tabular data.

The trade-offs of conforming data include time and cost. Translating data from a raw state into a tabular structure requires data pipelines to be developed and maintained. If the data source changes frequently, this will require recurring work to accommodate the new schema. In addition, raw data can be acted on immediately, whereas transforming raw data introduces the delay of data processing.

Understanding how data will be used will help guide the decision of what to conform and what to leave in a raw state. For example, if you launch a new feature for your application that requires a new data source, you might initially choose to leave the data in its raw state. Then, if the new feature proves popular, you could invest the resources to conform the data for better performance.

Data Lakes and Data Warehouses

A key difference between how data warehouses and data lakes handle data is that data warehouses store only conformed data, whereas data lakes contain data in its raw format. That is, a data warehouse transforms all data to a set schema as it is written, while a data lake performs the transformation on an as-needed basis, when the data is read by users. Because data warehouses conform data during ingestion, they can perform additional tasks on the data, such as validation and metadata extraction. For example, JPEG and video files

themselves would not undergo an ELT process (described in "ETL Versus ELT" on page 41), but information about the content, source, and date of creation could be important to capture.

In legacy data warehouses, data was limited to that which could conform to a rigid schema, owing to the lack of support for unstructured data. If data could not be made to conform to the data warehouse schema, it couldn't be used. This also meant users were limited to the data that had been curated inside the data warehouse. But with the large amount of variable structured data available today, the industry has made moves to loosen these restrictions to take advantage of the modern data landscape.

Data lakes are at the other extreme. A data lake can ingest data in any form, and it can be tempting to take in data from a broad range of sources without accounting for the complexity of the subsequent transformation process. Because data is transformed when it is read, different users can request different transformations of the same data source, resulting in the need to maintain many different transformations for a single source. This complexity may not be well understood from the outset, so attempts at creating a data lake can result in a large store of data that isn't usable, giving rise to the moniker "data swamp."

Data warehouses trade the cost and complexity of transforming the data on load with the benefits of conformed data. Data lakes trade the complexity of managing transformation when the data is accessed for the benefits of being able to quickly onboard new data sources.

Schema Evolution

Changes in the structure of data sources are an inevitability, and data pipelines must be built to address this reality. These changes are most often additive: additional columns available for a structured data source, or new fields in a JSON input.

For conformed data, adherence to a rigid schema makes supporting input data structure changes a heavy lift. Not only do data pipelines and table schemas need to be modified, but there is also a need to handle schema versioning. When data is accessed in its raw format, the issue of schema evolution is significantly reduced.

Far less often, changes can also be destructive, eliminating columns or splitting a single data source into multiple sources. Any such alterations in data sources require downstream schemas to be updated to accommodate the new data format. Such changes require careful planning and rollout regardless of schematization because they affect existing data and consumers of that data.

Other Trade-offs

When setting up a data pipeline for data applications, architects need to consider trade-offs among latency, data completeness, and cost.

With virtually infinite availability of compute resources, cloud data platforms can scale out to quickly complete data processing and analysis tasks. But this performance doesn't come for free: additional, more powerful (and more expensive) compute resources are required. Architects need to consider how to manage this, potentially creating different service tiers to offer latency/cost trade-offs to a wide variety of customers.

Another way to improve latency is by conforming data to enable faster analytical performance. In the cloud, compute resources are more expensive than storage, so it may be a worthwhile trade-off to spend the compute resources to conform the data.

A similar issue arises when considering how often to run data pipelines. For example, running a pipeline every five minutes will provide fresher data than running the pipeline once a day, but depending on the data application use case, it may not be worth the additional compute expense.

Data retention is another important consideration, as the amount of data retained will impact cloud storage costs. Retaining historical data is important in data applications supporting machine learning for model building and testing, and some use cases, such as healthcare or government services, may have strict requirements for keeping historical data.

Another aspect of data completeness relates to how the data is processed. In some cases, applications may want to retain the original data alongside transformed versions, such as for auditing purposes. As we've discussed previously, it may not be necessary to conform all fields in a dataset, depending on what is of interest to data users,

so there is opportunity to save on costs here while sacrificing completeness.

Best Practices for Data Processing

In this section we will present best practices for designing data processing pipelines when building data applications, highlighting how to take advantage of modern data platforms to build them.

ETL Versus ELT

Extract, transform, and load (ETL) is the process by which legacy data warehouses ingest and transform data. Data is first extracted from its source, then transformed to conform to the data warehouse schema and finally loaded into tables where it is accessible to end users. The conformed data enables performant queries and use of analytical tools requiring structured data.

ETL systems require an investment in development, maintenance, and compute resources to service the underlying data pipelines that perform this process. With many different data sources this can become a significant overhead and often requires staff dedicated to creating and maintaining these systems.

A modern approach to data processing is to move the transformation step to the end, as in data lakes. An extract, load, and transform (ELT) process loads raw data from the source without transforming it, preserving the original format. This not only removes the need to transform all source data, with the associated compute and maintenance costs, but also prevents information loss when converting data from its raw format to a conformed structure. Transformation can then be performed selectively: for example, if a conformed structure is needed to improve query performance or for data visualization tools. In addition, transforming only a subset of the extracted data reduces potential points of failure compared to ETL systems that transform all extracted data.

ELT represents a shift in boundaries in a data warehouse, enabling additional use cases over ETL systems. In legacy ETL systems, data is only loaded after it is transformed, limiting applications to the conformed version of the data. With ELT, applications can access either raw or transformed versions of the data. For example, exploratory data analysis on raw data is a first step in designing machine

learning solutions. This provides important insights around potential data issues and their downstream impacts that are not available in ETL systems.

The flexible nature of ELT provides opportunities for performant queries on conformed data alongside less performant raw data queries, enabling a choice in trading off performance for compute expense. In the next section we will discuss how to assess what subset of data to conform.

Schematization

Underlying the trade-offs of ETL versus ELT systems is a difference in when the raw data is schematized. *Schema on read* is the paradigm of ELT systems, where raw data can be queried in its native format. The schema is applied only when the data is accessed, hence "on read." *Schema on write* is the ETL paradigm, where the schema is applied when data is written into the data platform.

The schema-on-read approach offers significantly more flexibility. Because the raw format is always available, it is possible to create a number of derived, conformed versions to meet different business needs while only conforming the subset of data that is of interest.

Snowflake's `VARIANT` data type enables schema on read, allowing semi-structured data to be acted on without the delays of transforming data.[1] This not only gets data quickly into use, but also provides an opportunity to determine what fields customers are using to identify data that should be conformed.

Schema on read is also preferred by data vendors as it reduces the burden of handling changes in a data source. For example, if a data vendor adds additional fields, a schema-on-write system would need to modify the transformation step to handle these additional columns. In contrast, schema-on-read systems would immediately make this new data available.

1 *https://docs.snowflake.com/en/sql-reference/data-types-semistructured.html*

Loading Data

Loading is the process by which data is moved from a data source into the data platform, typically into a temporary staging table where it can be further processed. To support different types of data sources, the Snowflake Data Cloud provides two options for loading data: bulk loading and continuous loading.[2] Deciding which process is appropriate will depend on the type of data you need to load and the frequency at which the application requires it.

Serverless Versus serverful

The bulk copy approach is a *serverful* process—that is, one that requires deployment of dedicated compute resources to run the copy job. This is in contrast to Snowflake's ingestion service, now known as Snowpipe, which is a *serverless* process. In serverless processes, the compute resources are managed by the platform, not the data application. Serverless processes run immediately, whereas serverful processes incur delays waiting for compute resources to come online.

Keeping an eye on costs is important when leveraging serverless processes. It can be tempting to hand over all your jobs to a serverless process, letting the data platform figure out what resources are needed to run the job, but in some cases this can be costly. It is important to understand the workloads that rely on serverless processes, experimenting with running a few jobs and assessing the cost and performance trade-offs compared with a serverful approach.

Batch Versus Streaming

The frequency at which a data pipeline runs will depend on the needs of the data application, considering the trade-off of fresher data for the increased cost of running pipelines more frequently. Underlying data pipeline scheduling is the continuum of batch to stream processing.

Batch processes operate on bulk data, processing a chunk of data on a scheduled basis, such as loading historical transaction data once a day. Streaming processes operate continuously on as little as one

2 *https://docs.snowflake.com/en/user-guide/data-load-overview.html*

event at a time, or at an interval, processing micro-batches of events. Processing clickstream data in real time, as each event is being generated from the source, is an example of stream processing. You can think of stream processing as batch processing with a batch size of 1 (or, in the case of a micro-batch, a small number).

Batch processing

Batch processing is best suited for ingesting large amounts of data from sources that update on a regular schedule and to which the application does not require access in (near) real time. For example, transaction data that is loaded once a day as an input to a marketing recommendation application is well suited for batch processing.

With this approach, batches of data are copied from a data source into the data platform. In Snowflake this is accomplished using a `COPY` statement in SQL, issued from a virtual warehouse that developers must set up and size appropriately to service the job. This job can be run manually or set up to run on a schedule.

With the copy process, architects need to consider the trade-off of frequency and cost. How often data should be loaded will depend on both how often the data source publishes new data and how soon after this users need this information. More frequent copies will result in users having access to fresh data more regularly, but at the higher cost of more compute resources to schedule the copy jobs.

Appropriately sizing the virtual warehouse to service the copy process is another consideration. Understanding the amount of data in each batch is important to ensure sufficient compute resources are allocated.

Stream processing

Snowflake provides two methods for handling stream processing. For working with data in the cloud, continuous loading with Snowpipe is an alternative method to `COPY`.[3] Snowpipe processes data in micro-batches to surface new data quickly after it becomes available. This enables near-real-time access to data, such as would be required for IoT applications or making real-time recommendations

3 *https://docs.snowflake.com/en/user-guide/data-load-overview.html#continuous-loading-using-snowpipe*

based on clickstream data. Unlike with `COPY`, Snowpipe manages the compute resources for you, autoscaling as needed.

Change Data Capture

Detecting and responding to changes in source data is a key operation of a data platform. Rather than reingesting an entire data source when a change is detected, it is desirable to apply only the changes that occurred—that is, the delta between when the information was previously captured and its current state. *Change data capture* (CDC) aids in this process by tracking the data manipulation language (DML) changes made to a table (i.e., updates, deletes, inserts).

When processing data from streaming sources such as Apache Kafka,[4] Snowflake Streams and Tasks detect data changes and perform transformation on new and updated data. Snowflake `STREAM` objects provide CDC for underlying tables, enabling stream consumers to take action based on the changes that occurred since the consumer last queried the stream. For example, a `STREAM` object monitoring a staging table will reflect updates to the table as it receives new data from the source.

Streams can be used as an input to `TASK` objects—scheduled jobs that can be used to automate data processing steps with SQL statements or stored procedures—and Tasks can be chained, enabling the creation of multistep data pipelines.[5]

4 *https://docs.snowflake.com/en/user-guide/kafka-connector.html*

5 *https://medium.com/slalom-data-analytics/how-to-automate-data-pipelines-with-snowflake-streams-and-tasks-2b5b77ddff6a*

To understand the batch-to-streaming continuum, consider the architecture in Figure 4-1.

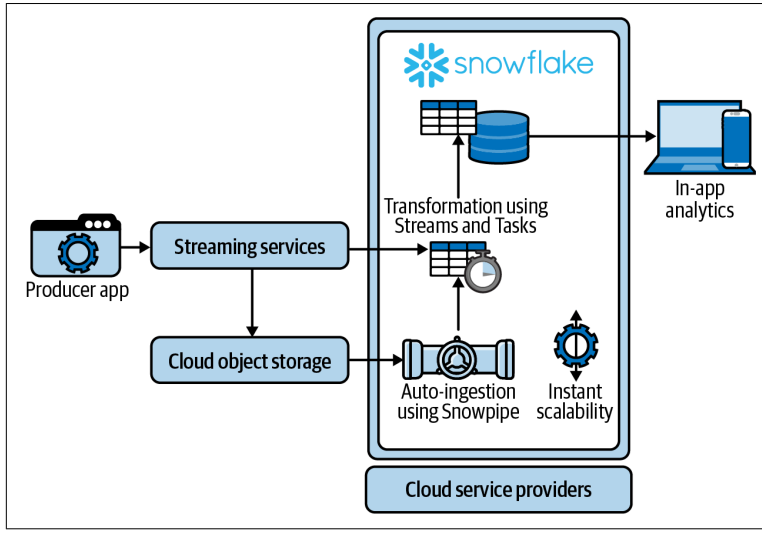

Figure 4-1. Snowflake streaming architecture

Producer applications generate continuous data that is surfaced by streaming services. Depending on data application needs, Snowflake ingests data either directly from the streaming service, processing it in near real time, or via batches published to cloud storage by streaming services.

For example, clickstream events would be ingested directly to enable action to be taken in near real time, such as choosing an ad to display within the application based on customer behavior. In this case data freshness is imperative, making the cost of processing the data continuously worthwhile. Snowflake Streams and Tasks process the data, surfacing it for use by the data application.

On the other hand, application transaction data used to update ad recommendation machine learning models is not needed in near real time and instead can be collected in batches in cloud storage and processed less frequently, such as once a day, by Snowpipe Auto-ingest. The results of the auto-ingest are handed off to Streams and Tasks to run the data pipelines on the bulk data source.

Summary

In this chapter you learned how data pipelines are critical to the success of data applications, which provide value through the ability to deliver fresh insights. From structured, historical data sourced from transaction systems to semi-structured data from devices and applications, data applications must be capable of processing a wide variety of data types.

Scalable, performant data processing leverages both conformed and raw data, selectively conforming data for performance and tool support and using raw data to evaluate new application features, determining if the cost and latency of conforming the data is worthwhile.

A hybrid approach combining the performance of conformed data warehouse tables and the raw data support of data lakes enables data processing systems to be tailored to customer needs while minimizing cost and maintenance overhead. Additionally, providing customers with access to raw data enables new use cases, such as machine learning, and ensures information is not lost as a result of transformation.

To realize these benefits ELT is the preferred approach for most data applications, enabling those applications to deliver fresh insights from both conformed and raw data. ELT systems offer significant flexibility and scalability gains over legacy ETL approaches by being more resilient to changes in data sources and supporting raw data.

When designing data processing systems it is important to consider the batch-to-streaming continuum. Streaming approaches process data in near real time, which is critical for applications such as responding to clickstream data with relevant ad recommendations. Batch approaches trade-off data freshness for lower costs, which is appropriate for cases where data can be consumed less frequently (such as incorporating historical transaction data). In this chapter you learned how Snowflake enables data processing for batch and streaming use cases through `COPY`, Streams and Tasks, and Snowpipe.

CHAPTER 5

Data Sharing

As we saw in Chapter 2, data sharing is an important requirement for data applications. In this chapter we will take a deep dive into this subject and how this impacts data applications.

We'll start with a discussion of different approaches to sharing data, then move on to design considerations in data applications. Next, you will learn about Snowflake's architecture, which eliminates the storage costs and overhead of traditional approaches.

In addition to sharing data among different parties, the ability to discover data is also an important element of data applications. For data consumers, this means knowing what data is available and how to get it. For data providers it means ensuring potential customers know about their offerings. You will learn how Snowflake Data Marketplace solves the data discovery problem, building a global data network to drive the data economy.

To provide an example of how data sharing in the Snowflake Data Cloud benefits data application builders, we will conclude with an overview of how Snowflake partner Braze leverages data sharing to drive their business.

Data Sharing Approaches

In this section we will discuss two different approaches for data sharing: sharing by creating copies of the data and sharing references to the data.

Sharing by Copy

The legacy approach to data sharing is to create copies of data to distribute to consumers, as illustrated in Figure 5-1.

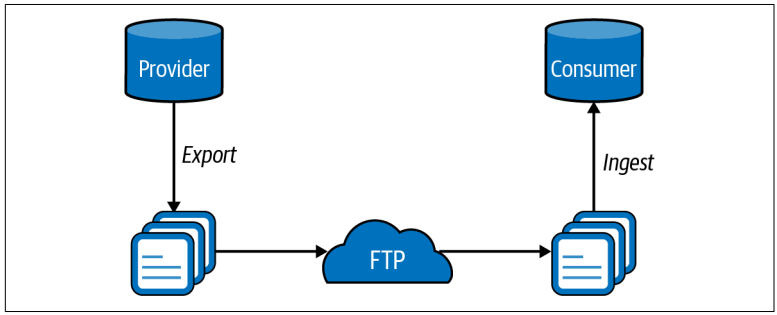

Figure 5-1. Sharing through data copy

Data providers export a copy of the data, which is transferred to the consumer. This transfer can take place over the web, using specialized file transfer software, or, in some cases, by mail (sending a disk to the consumer). Data providers and consumers share the burden of ensuring that data is copied securely and regulatory policies are followed.

With large amounts of data, this process can incur significant delays owing to the complexity and the time required to transfer large datasets. And once data is received, consumers must create and manage data ingestion processes to incorporate the copied data into their systems.

In our age of constantly changing data, providing a point-in-time data export results in the need to acquire new, updated copies as the data changes. Data consumers may ask for these exports frequently, putting a significant burden on data providers to maintain export systems and respond to requests.

Storage costs associated with data copy are also a significant drawback. With large data it is very costly to have to pay for storage. Furthermore, multiple versions of the data may need to be retained, multiplying this cost several times.

Another aspect of dealing with point-in-time data copies is the need to maintain version information. What version of data is presently in the consumer's system? When was it last refreshed? These details can fall by the wayside, leading to ambiguity as to data freshness.

This laborious process often results in data consumers working with stale data, not only because the data may already be stale upon export but because of the expense in time and money of importing a new version.

Sharing by Reference

A modern approach to data sharing, shown in Figure 5-2, replaces copying data with *referencing* data. Without the cost and overhead of copying data, the velocity increases dramatically, enabling data to be shared immediately.

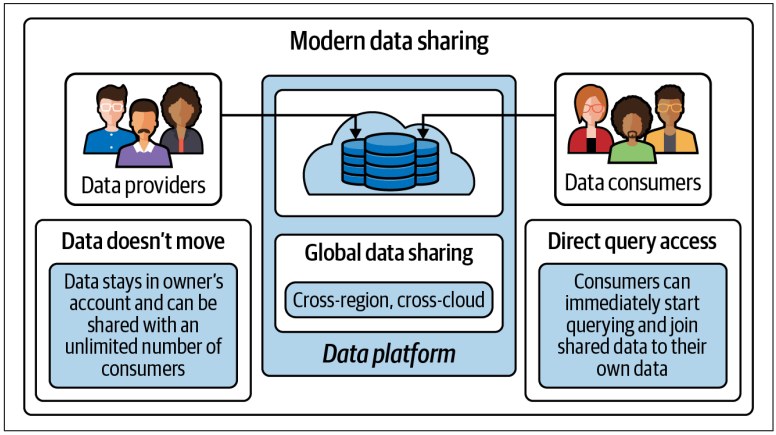

Figure 5-2. Data sharing in a modern data platform

As opposed to moving data between providers and consumers, in this scenario data remains fixed. Instead of managing a copy process and multiple versions of data, the data platform manages live data access in a transactionally consistent manner. Live data access requires providing transactional consistency to ensure consumers have a current view of the data at all times.[1] To make this seamless for data providers and consumers, a data platform should ensure transactional consistency of the data is guaranteed outside of the provider's system boundaries.

1 See the "Cloud Data Platform for Dummies" whitepaper.

Design Considerations

The more barriers there are to data sharing, the less likely it is to occur, reducing opportunities for both data providers and consumers. Building on what you've learned about data copy versus data sharing, in this section we will take a deeper look into the underlying mechanisms of data sharing to further help you assess the trade-offs.

Sharing Data with Users

In the modern age of quickly changing data, providing near-real-time, secure access to data is critical to providing value through data sharing. Similarly, it's necessary to be able to quickly revoke access to data, both to ensure appropriate business use and to comply with regulatory requirements.

In the data copy model, you lose control over data once it has been exported. Consider a scenario where you need to revoke access to data that has been shared to comply with regulatory requirements, such as the GDPR. In a data copy model you don't have control over the files that have been exported, so your only recourse is to trust the data consumers will delete the data when requested.

As well as region-specific laws about data transfer and access, in data copy systems there is the additional overhead of setting up global file transfer systems. With a worldwide customer base, you need to consider how data transport systems will be set up (region-specific FTP endpoints, centralized hub-and-spoke models). Furthermore, if you provide software for managing data transfer, you must also provide worldwide deployment and support.

Other important considerations when sharing data with users are cost and complexity. In the data copy model users must figure out what subset of data they need from providers, requiring investment in data pipelines just to perform this evaluation. And as we have discussed, data evolves over time, leading to inevitable changes requiring data pipelines to be reworked. Finally, there is the expense of storing large data files to consider. In the data copy model, large files must be transferred and stored, incurring additional expense.

Getting Feedback from Users

To create revenue from data sharing, a provider needs to understand what part of their dataset has value for different user groups. Feedback from data consumers is needed to determine this, but it's difficult to obtain. One option is to survey customers on data use, but this relies on creating a survey that meaningfully captures the information you need, and on customers to voluntarily respond.

Data Sharing in Snowflake

There is significant overhead and cost involved in developing, maintaining, and deploying the infrastructure for sharing copies of data. Data application builders shouldn't have to spend time managing this process. In this section, we will present Snowflake's low-friction approach to data sharing that eliminates many of the headaches of the data copy model.

With data applications built on the Snowflake Data Cloud, consumers and providers access the same copy of data. Because of Snowflake's multi-cluster, shared-data architecture, data is fully decoupled from compute, enabling data to be accessed across tenants. Customers can easily explore the data to determine what they need by executing SQL queries against shared tables.

With Snowflake Secure Data Sharing, there are no copies of data, eliminating the cost and complexities of data copy systems. There are no file transfers to manage or additional software needed, and there's no additional storage to pay for to host data copies. The main cost to consumers is the compute resources used when accessing the shared data.

Data access is controlled by Snowflake's services layer and metadata store, enabling access to be granted or revoked immediately and providing near-real-time access to data.

Figure 5-3 shows an example of using Snowflake's data sharing architecture.

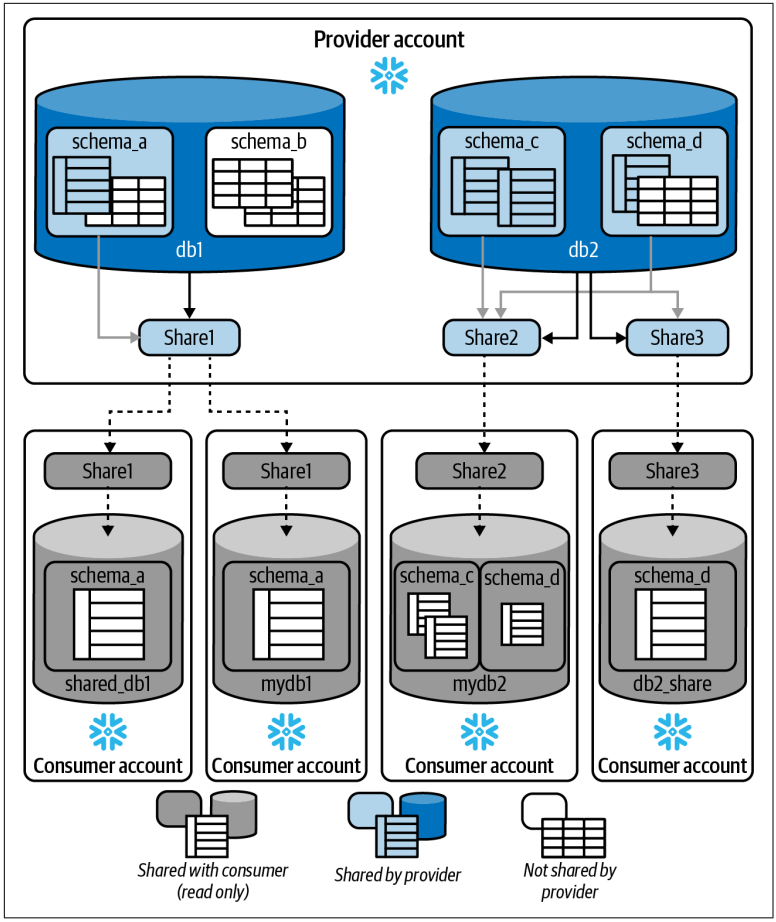

Figure 5-3. Snowflake data sharing example, adapted from Snowflake documentation[2]

As shown here, data providers can identify objects to share with read-only access. For example, in `db2` all objects in `schema_c` are shared, but only some objects in `schema_d` are shared. As new data in the shared objects become available on the provider side it is immediately accessible to the consumer, eliminating the issue of stale data with data copy systems. If the provider no longer wants to share data with a consumer, they can immediately revoke access.

2 *https://docs.snowflake.com/en/user-guide/data-sharing-intro.html*

In addition, Snowflake's cloud-first environment provides support for multi-cloud, cross-region data sharing. Because Snowflake manages handling data access across cloud providers, data application developers simply need to grant access to datasets without worrying about the underlying details. This enables Snowflake customers to have data replicas colocated with customers worldwide, ensuring reliable, low-latency data access.

Snowflake Data Marketplace

With low-friction, cloud-first data sharing, the Snowflake Data Cloud has a unique opportunity to transform data discovery. For data consumers, traditional data discovery involves searching for providers that might have the data of interest. For example, a consumer seeking demographic data would have to search for and evaluate third-party data providers, investing time and money in researching providers and ingesting sample datasets. For providers, the problem is one of being discovered—how do you make sure prospective consumers know about your datasets? And once you've connected with prospective customers, you must deal with the overhead of managing custom exports for each one.

Snowflake Data Marketplace brings together data providers and consumers, providing live, ready-to-query dataset access.[3] Consumers can instantly integrate data from providers on the marketplace with their own, realizing value as soon as the providers grant access.

In addition, Snowflake customers can share their data with a Snowflake Data Marketplace partner, who will enrich it and provide the resulting dataset back to the customer. Snowflake customers can use these services to expand their data processing capabilities without having to build these systems themselves.

As mentioned earlier, understanding how data is being used is important for data providers to provide more value to users. Snowflake offers this information through *data sharing usage views*, database views that provide information on queries consumers have run on the data and telemetry information such as inventory tracking.[4]

3 *https://www.snowflake.com/data-marketplace*

4 *https://docs.snowflake.com/en/sql-reference/data-sharing-usage.html*

For data providers, Snowflake Data Marketplace provides an opportunity to generate new revenue streams from data sharing. For example, Braze provides customer engagement benchmarks that enable ad tech companies to compare their marketing engagement data to this dataset and identify areas of improvement, or to run what-if projections.[5]

Snowflake Secure Data Sharing in Action: Braze

Snowflake partner Braze, a comprehensive customer engagement platform, provides a real-world example of how to leverage the Snowflake Data Cloud to make the most of data sharing in the customer 360 domain.[6]

Braze provides sophisticated tools for brands to orchestrate and deliver cross-channel campaigns. With live views of customer data and in-the-moment campaign optimization, the ability to quickly manage large volumes of data is paramount to the Braze experience.

Braze identifies the following as being key wins of data sharing in Snowflake:[7]

- Instant, secure, governed access to data
- Reduced time to insights by eliminating ETL and data copy
- Breaking down data silos
- No charge to consumers to store shared data

Braze integrates Snowflake Secure Data Sharing into its platform, providing customers with the ability to leverage their data within the application. Recall from "Application Health and Security" on page 7 that a challenge in Customer 360 use cases is creating insights from many different datasets. With Snowflake Secure Data Sharing, Braze customers can join their Braze message engagement and behavior data with all their other Snowflake data to create a holistic

5 *https://www.snowflake.com/datasets/braze-customer-engagement-benchmarks*

6 *https://www.braze.com*

7 *https://www.braze.com/docs/partners/data_and_infrastructure_agility/data_warehouses/snowflake/#what-is-data-sharing*

view of their users. With a single source of truth, Braze customers can create richer insights without the overhead of ETL processes or the cost of storing data copies.

Integrating the Snowflake Data Cloud also helps Braze attract prospective customers through Braze Benchmarks, an interactive tool enabling Braze companies to compare their metrics against the Braze customer engagement benchmarks available in Snowflake Data Marketplace.[8]

Summary

Data sharing provides both data application builders and their customers with opportunities for growth. To best realize these opportunities, a data platform should provide the following:

- Secure data transport and access
- The ability to share data across geographic regions
- A feedback loop between providers and consumers
- Timely access to data while keeping costs and maintenance burden in check

Snowflake Secure Data Sharing leverages the advantages of cloud-first systems by providing secure, read-only access to data through access controls, eliminating the risk, cost, and maintenance overhead of data copy systems.

Building on this seamless data sharing experience, Snowflake Data Marketplace improves data sharing by greatly reducing the issue of data discovery. It provides opportunities for data applications to get feedback from users, data processing providers to sell access to their products, and data providers to connect with consumers, democratizing access to the global data economy.

8 *https://www.braze.com/resources/benchmarks*

CHAPTER 6
Summary and Further Reading

Data applications are uniquely positioned to drive customer success and create new sources of revenue by taking advantage of modern data platforms. After decades of being held back by legacy technology, business needs are now in the driver's seat, providing fertile ground for innovation.

Data applications make data actionable through processing large volumes of complex, fast-changing data and providing customers with analytics capabilities to harness their data directly within the application. These applications need to handle all types of data and be flexible enough to accommodate changes in data sources while surfacing new data as quickly as possible to customers in a scalable compute environment.

Traditional data platforms lack the flexibility of modern, cloud-first approaches, making it difficult to use them to build successful data applications. In this report you've learned what to look for in a modern data platform to offload the data management burden from product teams so they can focus on building applications. Key components of these platforms include:

Cloud-first architecture
: Cloud-first data platforms maximize the advantages of modern technology, providing near-infinite storage and compute resources to support multiple tenants and workloads, and elasticity that guarantees SLAs in times of peak demand and keeps costs low when demand is low.

Separation of storage and compute
: Decoupling storage and compute maximizes the benefits of the cloud, enabling these resources to scale independently of one another.

Autoscaling
: A platform that will scale resources to match demand and handle load balancing will offload significant burden from data application teams in managing compute resources across tenants.

Workload isolation
: Ensuring customer workloads will not impact one another is critical in a multi-tenant environment. In addition, workload isolation enables the different processes within a tenant's environment to not compete for resources. This allows data pipelines to run while providing application services, such as real-time ad recommendations, without performance degradation for either.

Support for standard SQL
: The wide user base and mature ecosystem of SQL makes this a necessary feature for data platforms. SQL enables users to focus on expressing analytical queries while handing off optimization to the underlying engine.

Native support for semi-structured data
: Semi-structured data is a significant data source for modern data applications, including data from IoT, mobile, and web devices. The ability to store and query semi-structured data enables data applications to quickly pilot new features based on new datasets, without incurring the cost of developing, maintaining, and paying for resources to conform the data before it can be used.

Schema-on-read support
: Applying schema on read enables data applications to simplify data pipelines and shorten time to insight, even as the schema changes.

Data sharing by reference
: The benefits of immediate access and elimination of storage costs make live data sharing a must for modern data platforms.

For near-real-time applications live data sharing is essential, making traditional data copy approaches a nonstarter.

Throughout this report we've provided real-world examples using the Snowflake Data Cloud, a modern platform powering hundreds of data applications, purpose-built to optimally leverage modern data technology.

From here we recommend the following resources for more information about the topics covered in this report:

Snowflake virtual warehouses

- Virtual Warehouses

Building data applications

- The Product Manager's Guide to Building Data Apps on a Cloud Data Platform

Data pipelines in Snowflake

- Change Tracking Using Table Streams
- Introduction to Snowpipe
- Introduction to Data Pipelines

Security in Snowflake

- Managing Security in Snowflake

Data sharing

- Introduction to Secure Data Sharing

Data marketplaces

- What Is a Data Marketplace?

About the Authors

William Waddington is a distinguished engineer at Snowflake, where he has worked since 2015. During that time he has focused on the service components of the system, including service stability and overall system performance.

Prior to joining Snowflake, William was an architect at Oracle Corporation, where he worked in the RDBMS group particularly focusing on SQL execution and parallel and distributed SQL execution. William holds a Master of Science degree in Computer Science from Stanford University and a Bachelor of Science degree in Computer Engineering from Carnegie Mellon University.

Kevin McGinley is currently a technical director on the Customer Acceleration Team (SnowCAT) at Snowflake, focusing on taking customers to the next level using Snowflake's most strategic and bleeding-edge features, especially when building data applications.

Before joining Snowflake, Kevin developed data warehouses and business intelligence systems for Fortune 500 companies dating back to the mid-1990s. He has spoken at popular conferences, hosted a data-centric podcast, authored a vlog, and written many blog posts, and was also recognized in the Oracle space with the ACE Director distinction.

Pui Kei Johnston Chu is a senior engineering manager at Snowflake, where he leads the data sharing and data exchange team. His mission is to build the data economy for Snowflake's customers, making it secure for providers to distribute and monetize their data, and easy for consumers to discover and use it.

Before joining Snowflake, Johnston held engineering roles at Domo, Goldman Sachs, and Microsoft. He is a Hong Kong native, grew up in Toronto, Canada, and now lives with his family in San Mateo, CA.

Gjorgji Georgievski is an engineering manager at Snowflake, responsible for building data ingestion and pipeline technologies with low latency and high throughput characteristics. His goal is to give customers of the Snowflake Data Cloud the ability to compose and use comprehensive data processing pipelines for complex transformations.

Prior to joining Snowflake, Gjorgji was a principal engineering manager at Microsoft, working on query optimization and query execution for SQL Server and Azure SQL DB. He is a native Macedonian and lives with his family in Seattle, WA.

Dinesh Kulkarni is a principal product manager at Snowflake driving data ingestion and data pipelines. His goal is to continuously simplify and extend ways to get insights from data with efficient and effective data pipelines.

Before Snowflake, Dinesh was a PM for Google Cloud, where he launched Google Cloud Machine Learning—the first service to make hosted TensorFlow available to all with graphics processing units and Tensor Processing Units. Prior to Google, he worked at Microsoft as a PM for SQL Server, C#, and Azure App Services. Dinesh holds a Ph.D. in computer science and engineering from the University of Notre Dame, Indiana, and a Bachelor of Technology from IIT Bombay.