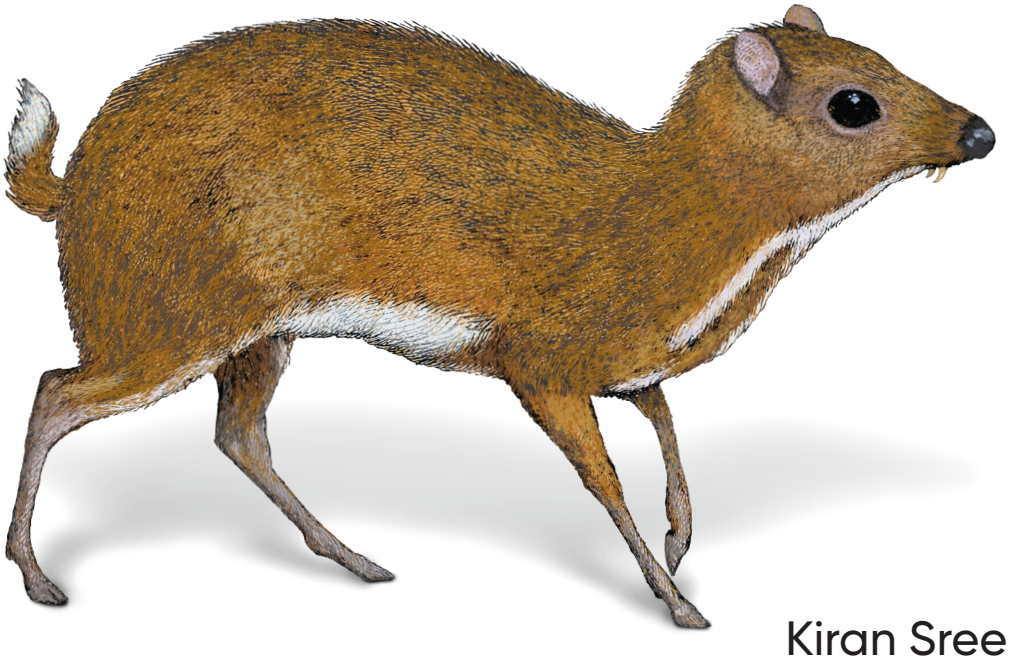
O'REILLY®

Data Governance with Unity Catalog on Databricks

Implement Data and AI Governance with Databricks Data Intelligence Platform



Kiran Sreekumar
& Karthik Subbarao
Foreword by Matei Zaharia

"The transformative approach of Unity Catalog meets the unrivaled experience of Databricks specialists Kiran and Karthik. Together, they reveal the future of data governance."

Lars George, lead product specialist, Databricks

"The authors succeed in delivering substantive technical content while engaging the reader through a compelling, real-world narrative style that is both refreshing and instructive."

Arup Nanda, managing director, head of data and AI, JPMorganChase

Data Governance with Unity Catalog on Databricks

Organizations collecting and using personal data must now heed a growing body of regulations, and the penalties for noncompliance are stiff. The ubiquity of the cloud and the advent of generative AI have only made it more crucial to govern data appropriately. Thousands of companies have turned to Databricks Unity Catalog to simplify data governance and manage their data and AI assets more effectively. This practical guide helps you do the same.

Databricks data specialists Kiran Sreekumar and Karthik Subbarao dive deep into Unity Catalog and share the best practices that enable data practitioners to build and serve their data and AI assets at scale. Data product owners, data engineers, AI/ML engineers, and data executives will examine various facets of data governance—including data sharing, auditing, access controls, and automation—as they discover how to establish a robust data governance framework that complies with regulations.

- **Explore data governance fundamentals and understand how they relate to Unity Catalog**
- **Utilize Unity Catalog to unify data and AI governance**
- **Access data efficiently for analytics**
- **Implement different data protection mechanisms**
- **Securely share data and AI assets internally and externally with Delta Sharing**

Kiran Sreekumar is a data architect with deep expertise in engineering and data and AI governance. Based in the United Kingdom, he has more than 15 years of experience in data and AI across industries and has built pioneering data products since the early days of Hadoop.

Karthik Subbarao is a specialist in data + AI platform and governance at Databricks, where he helps organizations develop secure, governed data and AI architectures at scale. He has more than a decade of experience designing and implementing solutions in the realms of big data and AI. Karthik is based in Germany.

DATA

US $69.99 CAN $87.99
ISBN: 978-1-098-17963-2

O'REILLY®

Praise for *Data Governance with Unity Catalog on Databricks*

Featuring up-to-date Unity Catalog capabilities and a fresh retail case study with Nexa Boutique, this book makes data governance on Databricks approachable. The authors methodically guide you step by step, providing clarity and practical insights you can apply immediately.

—Ashok Singamaneni, principal software engineer, Nike

With vivid examples and lucid explanations, this book is like a roadmap for implementing modern data governance in the age of the lakehouse.

—Tristen Wentling, lead solutions architect, Databricks

Governance in modern data ecosystems is often shrouded in complexity. Kiran and Karthik cut through the fog, offering actionable insights and engineering wisdom that make Unity Catalog truly accessible to every practitioner.

—Lars George, lead product specialist, Databricks

Presenting a technical topic as critical and complex as data management is a formidable challenge—particularly when viewed through the lens of a single data platform technology that is rapidly transforming the industry, all while maintaining a balanced perspective. In this book, the authors succeed in delivering substantive technical content while engaging the reader through a compelling, real-world narrative style that is both refreshing and instructive.

—Arup Nanda, managing director, head of Data and AI, JPMorganChase

Data Governance with Unity Catalog on Databricks

Implement Data and AI Governance with Databricks Data Intelligence Platform

Kiran Sreekumar and Karthik Subbarao

Foreword by Matei Zaharia

O'REILLY®

Data Governance with Unity Catalog on Databricks
by Kiran Sreekumar and Karthik Subbarao

Printed in the United States of America.

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

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: Aaron Black
Development Editor: Corbin Collins
Production Editor: Elizabeth Faerm
Copyeditor: Stephanie English
Proofreader: Andrea Schein
Indexer: Sue Klefstad
Cover Designer: Susan Brown
Cover Illustrator: Monica Kamsvaag
Interior Designer: David Futato
Interior Illustrator: Kate Dullea

September 2025: First Edition

Revision History for the First Edition
2025-09-11: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098179632* for release details.

978-1-098-17963-2

[LSI]

Table of Contents

Foreword

The ability to harness, secure, and govern data and AI assets is not just a technical requirement—it is a strategic imperative. As organizations accelerate their adoption of analytics and AI, the complexity and scale of data ecosystems have grown exponentially. The challenge is clear: how do we democratize access to data and AI while maintaining robust governance, compliance, and security?

We developed Unity Catalog at Databricks in response to this challenge, as the industry's first unified governance system for data and AI. In 2020, when we began working on the project, we were seeing more and more customers bogged down by the complexity of governing high-quality datasets than by the data analytics tasks themselves. Moreover, these same customers were starting to use unstructured data and AI in addition to tabular data, and these required a completely different governance infrastructure.

At this point, we had a choice: patch on individual governance features to our platform or design a unified governance system across the whole lifecycle of data, from unstructured files to AI models. We chose the latter, as we believed that solving this problem the "right" way would greatly simplify life for our customers. This bet paid off, with the vast majority of our workloads now running on Unity Catalog and customers reporting significant improvements from the simplicity of unified governance.

Databricks has always advocated for open source software and open formats. Unity Catalog is no different. Our vision is for Unity Catalog to be the most open and interoperable catalog for data and AI. On top of the catalog, we have also pioneered open interfaces for data sharing across companies by creating the Delta Sharing protocol, which is now one of the largest ecosystems for data delivery in the industry.

This book is a guide to understanding, implementing, and mastering the basics of data governance with Databricks Unity Catalog. Whether you are modernizing legacy systems, scaling data operations, or building a foundation for AI, the principles and best practices outlined here will help you create a unified, secure, and future-ready data platform. You will discover how you can leverage Unity Catalog to streamline governance, enhance transparency, improve data quality, and unlock the full potential of your data estates.

Embark on the next stage of data governance, where Unity Catalog centralizes clarity, control, and collaboration within your core data strategy.

— Matei Zaharia,
CTO and cofounder at Databricks

Preface

Welcome to *Data Governance with Unity Catalog on Databricks*. In the current transition from a decade dominated by the evolution of file formats, the next era is being shaped by the prominence of catalogs. Since its introduction in 2021, Unity Catalog has emerged as the foundational component of the Databricks Data Intelligence Platform. The open sourcing of Unity Catalog has unlocked new possibilities for governance innovation, overcoming traditional tool limitations.

As the industry continues its shift toward open and interoperable technologies, Unity Catalog stands out as an open and extensible catalog. Its open API and integration with leading file formats and applications in the data and AI landscape enable unparalleled flexibility. Building on this foundation, with native support for Delta and Iceberg REST catalog and tables, views, cloud storage files, AI models, and functions, Unity Catalog is a true multimodal catalog that provides a unified data-management experience.

Why We Wrote This Book and Why Now

When we first pitched the idea of this book to O'Reilly, Unity Catalog was a proprietary component of the Databricks Platform. Over the course of the book's writing, Unity Catalog became open source and incorporated numerous enhancements and new features. Despite the architecture having undergone significant changes and evolution over time, the fundamental principles and governance implementation have remained relatively consistent. Having worked closely with Unity Catalog for over three years, we recognized the importance of solidly understanding its core concepts. We determined that the time was ripe to document these foundational elements, as well as the features and functionalities that are currently publicly available.

Who This Book Is For

Our book on governance with Unity Catalog is primarily aimed at the following people:

Data architects and executives
: Decision makers responsible for shaping the data and AI strategy for entire organizations or business units will get a lot out of this book. It provides a comprehensive understanding of the industry direction on open, centralized multimodal catalogs and interoperability, enabling them to make informed strategic decisions that drive business success.

Data application teams
: This book is also for professionals who design and implement ETL pipelines, automate data processing, and make data accessible to various stakeholders, including data engineers, analysts, and scientists. It provides practical guidance on leveraging Unity Catalog to improve data management and governance, ensure data quality, and enable seamless data sharing and collaboration.

Cloud platform engineers
: Experts responsible for providing a platform for data teams and enforcing security and governance standards will benefit from this book. It provides in-depth guidance on leveraging Unity Catalog to build a robust, secure, and compliant data platform that supports their organization's data-driven initiatives.

Databricks consultants
: For seasoned professionals with experience using the Databricks Platform, this book provides expert guidance on Unity Catalog and its applications, enabling them to enhance their consulting services and deliver high-value solutions to their clients.

Machine learning (ML)/AI engineers
: As the practitioners responsible for designing, developing, and deploying AI/ML models and functions, ML/AI engineers should read this book to learn how Unity Catalog can help them secure their AI/ML assets and ensure the integrity of their models and data.

Additionally, the following group may also benefit from this book, although to a lesser extent:

Data stewards and chief data officers (CDOs)
: While the book focuses on the practical aspects of Unity Catalog and related features, the concepts and approaches discussed will help data stewards and CDOs understand the overall governance framework and how Unity Catalog can support their organization's compliance with regulatory standards.

How This Book Is Organized

To get the most out of this book, we recommend starting with Prologue: Governance by Choice, where we introduce Nexa Boutique. This fictional organization serves as the backdrop for our exploration of Unity Catalog. Although our stories and experiences about Nexa Boutique are made up, they are informed by technical accuracy and designed to illustrate key concepts and best practices. By following Nexa Boutique's journey, you'll better understand how Unity Catalog can be applied in a real-world setting.

The chapters provide a comprehensive guide to data governance, covering the essential concepts, infrastructure, and implementation considerations. They also examine the features and capabilities of Unity Catalog in depth, to provide a thorough understanding of its role in supporting effective data governance:

Chapter 1, "The Modern Governance Stack"
: We explain the fundamentals of data governance, lakehouse, and Databricks architecture, and briefly introduce Unity Catalog.

Chapter 2, "Unity Catalog Under the Hood"
: The chapter discusses the history of governance in HMS and how it led to the creation of Unity Catalog and takes a deep dive into its architecture. The chapter ends with Unity Catalog's governance model and data-management features.

Chapter 3, " Identity Management"
: Identity management is a critical foundation for data and AI governance, and we explore it in detail, starting with cloud-specific considerations and progressing to identity provisioning, single sign-on, and best practices.

Chapter 4, "Unity Catalog and Compute"
: Platform teams will find this section particularly relevant as we explore the changes to compute infrastructure in Unity Catalog. It covers standard, dedicated, and serverless access modes and their respective features and limitations.

Chapter 5, "Access Controls and Permissions Model"
: We explore Unity Catalog access controls in depth, examining the data governance models and the range of functionalities available. We dive deep into catalog layout and permissions modelling.

Chapter 6, "Governing AI"
: This chapter explores the new paradigm of AI governance practices in Unity Catalog, covering the lifecycle of AI model governance and the components and implementation of AI systems.

Chapter 7, "Observability and Discoverability"
: This section is particularly relevant for DevOps and FinOps teams, as we explore the critical aspects of data observability and discoverability in a governance platform and how Unity Catalog supports these capabilities.

Chapter 8, "Data Sharing and Collaboration"
: This chapter explores how to share data and AI assets within and outside your organization to enable collaboration. It covers different patterns and best practices with a focus on cross-metastore governance.

Chapter 9, "Open Access"
: Open access to data from external tools and engines is crucial when choosing a catalog. This section explores the various integration options for open data access and the open source Unity Catalog.

Chapter 10, "Being Compliant with Regulatory Standards"
: This chapter discusses how to comply with regulatory requirements, including General Data Protection Regulation (GDPR) and architectural best practices for handling sensitive data.

Chapter 11, "Accelerating Unity Catalog Adoption"
: Having learned how to use Unity Catalog, you can now accelerate its adoption in your organization. This chapter covers automatic Unity Catalog enablement, migration, and federation from your Hive metastore to Unity Catalog.

Chapter 12, "The Future of Unity Catalog"
: We conclude this book with a forward-looking perspective on the direction of Unity Catalog, its future, and what's coming next on its roadmap.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
141 Stony Circle, Suite 195
Santa Rosa, CA 95401
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata and any additional information. You can access this page at *https://oreil.ly/data-governance-with-unity-catalog-on-databricks*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

This book wouldn't have been possible without the tremendous work done by the Databricks engineering and product teams under the leadership of Matei Zaharia in building Unity Catalog for the Databricks Platform and eventually making it open source for the community.

We would like to extend our heartfelt thanks to Lars George, who provided invaluable guidance and offered insightful comments and suggestions throughout the writing of this book. Special thanks to fellow reviewers Andrew Weaver, Arup Nanda, Ashok Singamaneni, Bennie Haelen, Siddharth Bhai, and Tristen Wentling for their detailed reviews and suggestions, which helped us refine the book. A very special thanks to Bernhard Walter, whose knowledge and ideas have influenced the content of this book.

We are thankful to the team at O'Reilly, especially Aaron Black and Corbin Collins, for supporting our proposal to publish this book and partnering with us to make it a reality.

Kiran

I never thought I would be writing a book a year or two ago, although it's been a long-held aspiration on my bucket list. I would like to extend my thanks to my colleagues, who collaborated with me and the never-ending data governance discussions with Unity Catalog, as this book is a reflection of the learnings and insights gained from these interactions. I am forever grateful to my wife, Devika, for her love, support, and sacrifice throughout the writing of this book. Your encouragement and understanding helped me stay focused and motivated. To our little bundle of joy, Kedhar, who always keeps me on my toes and makes every day an adventure, I am so grateful to have you in my life.

Karthik

Writing this book has been a fascinating journey but committing to this project required significant time, energy, and focus, which inevitably meant spending less time with my family. I am deeply grateful to my wife, Pooja, for being immensely supportive of this journey and to our delightful daughter, Inchara, whose joy has been a continuous source of inspiration. I dedicate this book to my late father, Subbarao, who would have been overjoyed to see me become an author.

Prologue: Governance by Choice

Data is useful. High-quality, well-understood, auditable data is priceless.
—Ted Friedman, Gartner

In 2015, Nexa Boutique—a fictional company we'll be using as an example in this book—was incorporated as a startup in the online fashion retail industry headquartered in the United States (US). Nexa Boutique is committed to promoting sustainable fashion practices through the design, production, and sale of eco-friendly clothing and accessories. With a bold vision to revolutionize the retail industry, Nexa Boutique set out to utilize the power of data-driven insights. It leveraged the increasing popularity of data lake architectures to support business decisions and investments, which positioned the company at the forefront of the industry's transformation, driven by data analytics. In the early stages of its journey, Nexa Boutique made a strategic decision to invest heavily in its data lake platform. This proved to be a catalyst that accelerated growth by leveraging predictive analytics to optimize the supply chain and forecast market trends and customer behavior.

Definition: Nexa Boutique

Nexa Boutique, or Nexa, as we refer to it throughout the book—a name that combines "Nexa," implying next-generation innovation, with "boutique," a French term for a specialized shop—is our hypothetical example of a company that has been hugely successful in democratizing data by implementing Unity Catalog. All descriptions of its internal workings and data journey are entirely fictional and do not represent any real-world events or companies.

Definition: Data Lake

A *data lake* is a central location that holds a large amount of data in its native, raw format. Compared to a hierarchical *data warehouse*, which stores data in files or folders, a data lake uses a flat architecture and object storage to store the data.

Nexa's explosive growth propelled it into new markets, with operations expanding to the United Kingdom (UK) and Europe. Within four years, Nexa had established a presence in 40 countries worldwide. This growth created a hugely complex data platform with multiple tools and applications deployed and managed in silos. The data team faced massive challenges in coping with the increased data loads and meeting diverse regulatory requirements across geographies. For customers in the European Union (EU), the company had to comply with the General Data Protection Regulation (GDPR), which imposed strict rules on data collection, storage, and processing. For customers in the US, it had to navigate the California Consumer Privacy Act (CCPA) and similar laws, which grant California residents new rights over their personal data, including the right to know where their information is used, to delete that information at will, and to opt out of the sale of their personal information.

As a retail company operating in the ecommerce domain and handling customer credit card information, Nexa was also governed by the global Payment Card Industry Data Security Standard (PCI DSS), a globally recognized regulation that dictates the secure processing, storage, and transmission of sensitive data, including payment card data.

The data team at Nexa struggled to keep pace with these diverse regulatory requirements. It had to implement different data governance policies and procedures for each region, which added complexity and cost to operations. It also had to ensure that its data platform complies with each region's regulations, which required significant data security and infrastructure investments.

The *central data platform* (CDP) team, organized in a hub-and-spoke model, actively manages data ingestion and transformation for all the datasets within the enterprise. Figure P-1 illustrates the organizational structure of Nexa, comprising its multiple business domains and subdomains.

This model worked well for Nexa as the various domain teams did not have the necessary skill sets for data ingestion and modeling. The CDP team created standardized blueprints for data ingestion and transformation, which the domain teams could follow to ensure consistency and quality. Additionally, the CDP team took ownership of the data ingestion and transformation process, preparing the data for consumption by the domain teams. This approach enabled the data analysts from domain teams to focus on their core responsibilities, while the central data team handled the complex tasks of data preparation and integration.

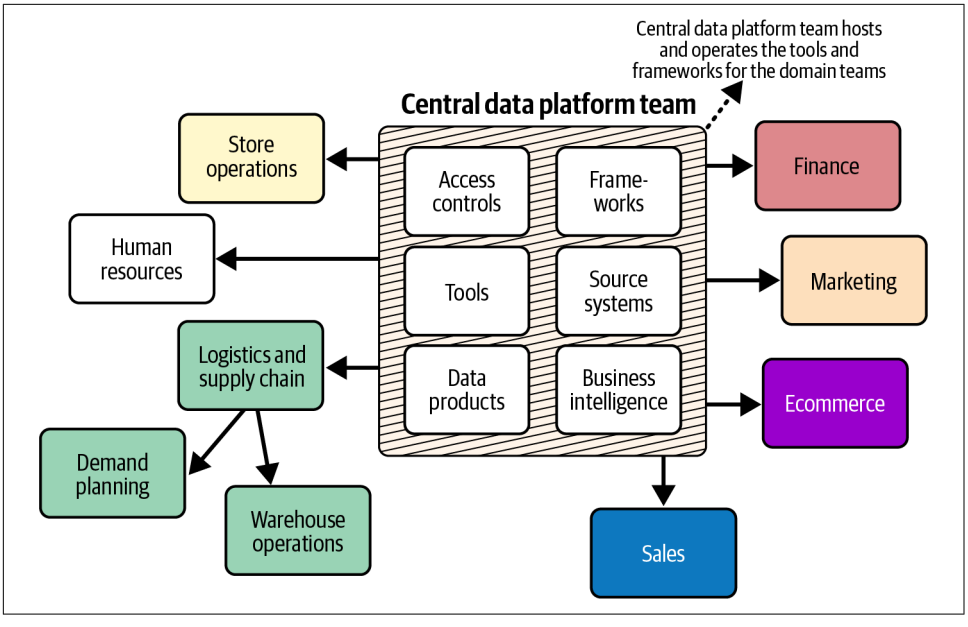


Figure P-1. The business domains within Nexa

The Journey to Being a Multicloud Data Platform

Until 2019, Nexa ran its data lake entirely on Apache Spark clusters in its on-premises data center. As part of the vision to be an entirely multicloud business, Nexa started building out its cloud data platform across different cloud vendors. Nexa undertook a massive migration program to move the on-premises data platform to the cloud. This migration enabled it to scale the data platform in line with its fast-growing business and the associated data.

Definition: Apache Spark

Apache Spark is an open source multilanguage data processing engine used for data engineering, data science, and machine learning (ML) workloads. It can handle both batches and real-time streaming analytics, SQL, and ML workloads.

However, the CDP team faced a significant challenge: a siloed data platform.

The data platform team at Nexa had to stitch together multiple cloud platform solutions to build their data pipelines, which were essential for supporting their analytics requirements. A substantial portion of their data resided in the data lake, and the reporting requirements were met by copying the same data to a cloud data warehouse. This resulted in the creation of data silos and disparate governance

solutions, ultimately complicating their data estate. Despite the data platform's complexities causing slowdowns, the business demand for rapid insights and advanced decision-making capabilities continued to accelerate, driven by the growing potential of AI to unlock new opportunities and drive competitive advantage.

In 2020, Nexa's CEO made a strategic decision to unify the company's data platform and simplify data governance by building a lakehouse platform to leverage the advancements in AI and better comply with regulatory requirements.

Databricks Lakehouse and Unification of the Data Estate

In 2021, based on its CEO's decision to unify its data estate, Nexa evaluated multiple cloud-based data platform tools to build its enterprise lakehouse and, after careful consideration, selected Databricks to build its lakehouse. The decision to choose Databricks was based on several factors, including the need for a unified platform that could support ML, data warehousing, and streaming applications seamlessly using data lakehouse architecture. Nexa's multicloud strategy was crucial to its growth because it allowed the company to adapt to regional differences in cloud provider solutions, such as varying levels of support for specific services or compliance requirements, as it expanded into new markets. Given these requirements, the ability to operate on multiple clouds made Databricks the clear choice for Nexa, as it provided the flexibility and scalability the company needed to continue its rapid expansion.

Definition: Data Lakehouse

A *data lakehouse* represents a paradigm shift in data management, combining the best of a data warehouse and a data lake. This architecture empowers organizations to unlock the full potential of their data, supporting advanced analytics, business intelligence, and ML applications across the entire data spectrum from a single source of data stored in an open file format.

By unifying its data estate with Databricks, Nexa was able to accelerate the execution of its data and AI strategy. Leveraging Databricks ML capabilities, Nexa rolled out its customer churn prediction models, allowing it to develop targeted marketing campaigns and personalized promotional offers to retain its growing customer base. Nexa invested in building a streaming platform to enhance the UX and personalization of its online store. The business grew by 35% after rolling out new capabilities powered by the Databricks Data Intelligence Platform.

As Nexa's business continued to grow, so did the complexity of its data landscape. Two problems emerged with its data platform:

Proper data governance

First, the company struggled with balancing data democratization across the organization while maintaining stringent access controls and data governance standards. It had to ensure that sensitive data was protected and complied with regulatory requirements while enabling business stakeholders to make data-driven decisions without delay. As the volume of data shared internally and externally continued to grow, it highlighted an area for improvement in the Nexa data platform. The company recognized that enhanced visibility and control over data usage, both within the organization and with external partners, would be beneficial in mitigating potential risks. This awareness sparked a desire to strengthen data management practices and ensure the secure and responsible sharing of information. The lack of transparency into available data sources in the data platform often caused users to reload existing data, resulting in redundant data duplication and inefficiencies. Furthermore, data owners faced substantial challenges in tracing the lineage of their data assets, and the lack of centralized management and auditing capabilities meant that data was scattered and unaccounted for.

Central platform team as a bottleneck

This problem was all about scalability. The domain teams relied heavily on the CDP team to make data available for consumption so that the domain teams could develop new data solutions. Any new data that could enhance their decision-making capabilities had to navigate through multiple hops, significantly slowing down their new initiatives. At this point, they discovered the *data mesh* concept introduced by Zhamak Dehghani and her book *Data Mesh* (O'Reilly), which offered a better way for Nexa to scale.

Leveraging Unity Catalog

With its continued growth, Nexa executed a series of acquisitions, merging some newly acquired businesses with the parent company while retaining others as a separate entity under the parent organization's umbrella. Tamarind Shoes (also fictional) was one such iconic brand in the sports shoe business that was acquired and retained as a separate entity under Nexa. This arrangement presented new challenges in sharing data between these entities. Until then, the entire Databricks Platform was running on a Hive metastore-based data catalog, which had several limitations when it came to governance and data sharing.

In 2022, Nexa adopted Unity Catalog to streamline its data governance process and enhance UX.

Definition: Hive Metastore

In the paper "Hive—A Warehousing Solution Over a Map-Reduce Framework", published by the Facebook data infrastructure team, the creators of Hive, Ashish Thusoo and Joydeep Sen Sarma, mention the HMS: "Hive also includes a system catalog, Hive-Metastore, containing schemas and statistics, which is useful in data exploration and query optimization."

Definition: Unity Catalog

Unity Catalog is a data governance solution from Databricks that unifies data and AI asset governance and enables enhanced data sharing and access control capabilities.

Unity Catalog became generally available in the Databricks Platform in 2022 and has evolved quite a bit since. Nexa was an early adopter of the technology and underwent a journey of experimentation, exploring various architectural patterns before ultimately arriving at its current state-of-the-art solution. The CDP team's role as a single point of dependency for data availability created a constraint, slowing the pace of data delivery and introducing a risk that Nexa's consumer experience might be impacted by delays or inefficiencies. By adopting a decentralized platform strategy, Nexa was able to rapidly deploy the data mesh architecture, which provided domain teams with the autonomy to manage their own data and work independently.

This change enabled teams to innovate and respond to changing business needs with greater speed and agility. With Unity Catalog adoption, Nexa migrated its data architecture to a harmonized data mesh, as depicted in Figure P-2.

Throughout this book, we will guide you through the story of how Nexa implemented Unity Catalog and democratized its biggest asset: data. You will see how Nexa's CDP team built a sophisticated and well-governed data platform and navigated the challenges of its growing data assets.

We'll begin in Chapter 1 by stressing the importance of data governance and the shift in the tech industry toward a unified data platform that caters to all data and AI use cases and how this eventually led to the genesis of Unity Catalog.

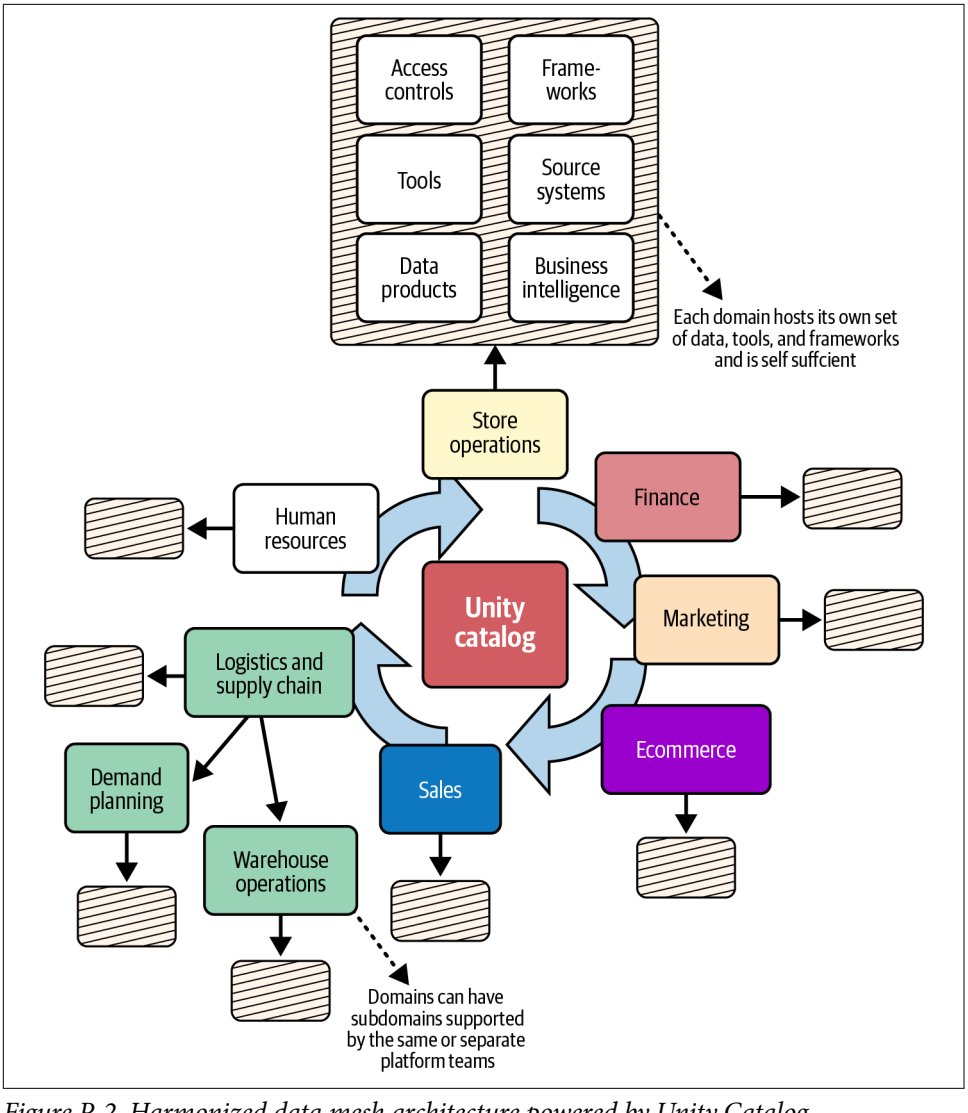


Figure P-2. Harmonized data mesh architecture powered by Unity Catalog

CHAPTER 1

The Modern Governance Stack

Data is the new oil.
—Clive Humby

In this chapter, we'll start with an introduction to data governance and why it's important. If you're already familiar with data governance in general and convinced that it is truly important in the analytics space, you may skip to "The Dawn of the Lakehouse" on page 8. If you're already familiar with the lakehouse paradigm and the Databricks Platform, then jump right into "Databricks Unity Catalog: Enabling Unified Governance" on page 22.

Introducing Data Governance

On October 28, 2021, a cyber security team at SafetyDetectives discovered an unsecured Amazon S3 bucket that contained over a million files. Among other things, the content of the bucket included personally identifiable information (PII) of employees and sensitive company data of some airports in Colombia and Peru. Although the contents of the buckets dated back to November 2018, no one knew how long the buckets had been publicly exposed. Apparently, the S3 bucket belonged to a well-established and prominent security services company called Securitas, headquartered in Stockholm, Sweden, with branches around the world.

This was not a standalone incident. In July 2019, Capital One suffered a data breach where over 100 million customer records were accessed due to a cloud firewall configuration vulnerability. Yet another incident related to Amazon S3 bucket misconfiguration led to the exposure of a significant amount of sensitive data in March 2022. The data exposed belonged to Pegasus Airlines, a low-cost carrier in Turkey with bases at several Turkish airports. The exposure was said to be due to human error. If this data, which included almost 23 million files, were to land in the wrong

hands, thousands of passengers and the flight crew would have been affected. For instance, such personal information could be used to commit identity theft and financial fraud. Moreover, exposing personal data is a breach of Turkey's Law on Protection of Personal Data (LPPD) and could have resulted in fines being imposed.

Unfortunately, such events are not uncommon. Multiple studies have shown "human error" to be the main cause of cyber security incidents. According to Verizon's Data Breach Investigations Report, published in 2022, 82% of breaches involved a human element. Gartner predicts that in the near future, lack of talent or human failure will be responsible for over half of significant cyber incidents.

Given this trend, we need to work on security processes and practices with a human-centric approach. In other words, humans will make mistakes, so we need to make it much harder for them to make mistakes and put in guardrails to minimize the impact of those mistakes when they inevitably happen. This requires implementing several measures, such as educating and enabling people on security topics, securing the network, implementing a zero-trust security model, and having robust data governance practices.

Data governance, at its core, is a comprehensive approach to managing an organization's data to ensure its availability, usability, quality, integrity, and security throughout its lifecycle.

Key components of data governance include the following:

- Policies and standards
- Data quality management
- Metadata management
- Access controls and security measures
- Data lifecycle management and lineage tracking
- Auditing and compliance monitoring

Benefits of Effective Data Governance

Proper data governance practices play a crucial role in reducing the risk of data breaches and security incidents through several key mechanisms, such as:

Establish clear policies and controls
: Define clear and comprehensive policies and controls for managing data throughout its lifecycle, which includes the following:

  - Implement strong access controls and authentication procedures to ensure only authorized personnel can access data and related assets.

- Enforce data classification and risk assessment to identify and prioritize the protection of high-risk data assets.
- Establish data handling policies or guidelines that specify how data should be stored, processed, transmitted, and disposed of securely.

Enhance data security measures
: Support and strengthen the overall security posture of an organization by doing the following:

  - Require or promote the implementation of robust security measures like encryption, use of firewalls, and intrusion detection systems.
  - Implement data minimization practices to reduce the overall attack surface and potential impact of a breach.
  - Enforce regular security audits and risk assessments to identify and proactively address potential vulnerabilities.
  - Establish incident response plans to address and mitigate damage from security breaches quickly.

Foster a culture of security awareness
: Go beyond technical controls to cultivate a workforce that becomes the strength of an organization by being human firewalls rather than the weakness that leads to security incidents through the following:

  - Organize regular employee training and awareness programs on data security best practices and the importance of protecting sensitive information.
  - Create accountability for data security across the organization by defining clear roles and responsibilities for data management.

Additional benefits of implementing proper and effective data governance include the following:

- Improved data quality and reliability
- Enhanced regulatory compliance
- Better decision making based on trusted data
- Increased operational efficiency
- Reduced data management costs
- Greater trust from consumers

Let's start by delving deeper into the lifecycle of data to comprehend the challenges of data governance.

The Lifecycle of Data

The generic data lifecycle in an organization has four stages, as illustrated in Figure 1-1:

Onboarding
: This is when the data is collected, generated, or ingested:

- *Collected:* For example, users signing up to a website
- *Generated:* For example, signals from machines
- *Ingested:* For example, a product catalog from a supplier

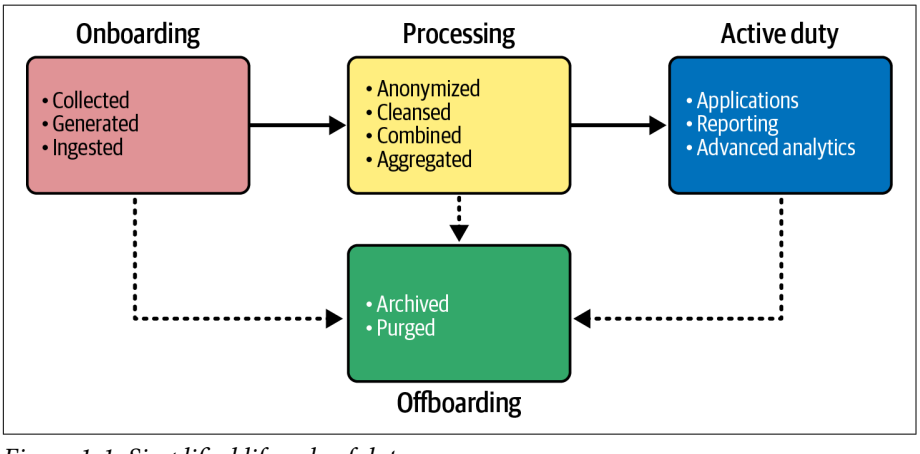


Figure 1-1. Simplified lifecycle of data

This is a very simple and high-level view of the data lifecycle. The real world is typically much more complicated.

Processing
: It is often the case that, when data comes into existence in your organization, it needs to be processed in some form to make it more usable. This stage is called *processing*. Of course, this depends on how good the process of data onboarding is. In other words, the first step you take with data is to make sure it is of good quality, and to achieve this, you might need to take additional steps, for example, filtering some data out, filling in the unavailable values, and so on.

  Other forms of processing include combining or merging the data and aggregation. For example, you might have different data points representing the same entity in the real world. This means you need to first identify which records belong to the same entity and merge them. This process is termed *entity*

resolution, not to be confused with *data harmonization,* which is the process of unifying or consolidating data from various sources into a coherent and standardized format. You might also need to combine different datasets to create new ones to make them more valuable, a simple example being *table joins.*

Aggregating data is also a very common processing step, which is usually done to extract information, in other words, to gain insights. Sometimes the value of data lies in the aggregated form, and one can get rid of the source data once the aggregated results are computed and stored. An example of this is sensor data from *Internet of Things (IoT)* systems, where applications and devices generate large amounts of granular data points, which are aggregated into hourly or daily summaries, and the detailed data points are purged to save storage space. Another example is a financial system, such as high-frequency trading platforms, that generate massive amounts of data, which are aggregated into daily price summaries. Individual trade records are purged after regulatory retention periods. Purging data after the intended use might also be a requirement from the compliance perspective in cases where datasets can be used only for generating aggregated results.

Another measure, to be compliant with regulatory requirements, is to anonymize data. The extent of anonymization can vary depending on the intended use of the data in the corresponding use cases. Sometimes parts of the data, for example, personally identifiable information (PII) or personal health information (PHI), need to be completely anonymized. In other cases, you might need to anonymize it in a way that you could still use it for a data product yet without the risk of completely disclosing the data. Later chapters delve deeper into this topic.

In simple terms, PII is any information that can identify an individual directly or could do so when combined with other information.

PHI is a subset of PII specifically associated with healthcare contexts. PHI is related to an individual's health status and healthcare services and is protected under the Health Insurance Portability and Accountability Act (HIPAA) in the US.

Active duty

The data, thus processed or unprocessed, can be used either in applications, for reporting, or for advanced analytics purposes such as training a predictive machine learning (ML) model. This stage, when the data is being used for a specific reason, is termed *active duty*.

Offboarding

Once the purpose of the data is fulfilled and the data is no longer needed, it should be either purged or archived, depending on the organizational policies and regulatory requirements. The purging or archiving of data can also be done after any stage. Consider the data that is used for some specific purpose, say, verifying the correctness of the information provided. In that case, you need to store the data only until the verification is done. This last stage of the data lifecycle is termed *offboarding*.

To dive deep into the concepts of data governance in general, *Data Governance: The Definitive Guide* (O'Reilly) by Evren Eryurek, Uri Gilad, et al. is a great resource.

Governing the Ungoverned

Until recently, data governance solutions were designed and implemented for managing data that is well-defined and well-structured. In other words, data governance typically has been applied to data in the form of tables or views with well-defined schemas. The reason is that in the analytics space, in most cases, the business needs were fulfilled by data ingested from online transaction processing (OLTP) systems that are usually structured in nature. However, most data (80% to 90% per some analyst estimates) generated or collected by organizations is *unstructured*. Some examples include emails, text files, logfiles, media files, and so forth. Due to various challenges, despite being abundant, the unstructured data is mostly underutilized. These challenges include lack of the right tools and technologies, concerns around regulatory compliance, and general lack of expertise. Hence, the unstructured data, although it may have great potential, ends up as archives in so-called *data swamps*.

However, this situation is now changing rapidly, thanks to the rise of AI. Many organizations are very interested in AI and have realized that unstructured data leveraged by ML and other AI techniques can give them significant advantages in data-driven decision making. AI is leading to improvements in existing business outcomes and is also creating new opportunities and new revenue streams. For example, AI can analyze market trends, customer feedback, and other data to identify opportunities for developing new products and services.

We use the term *AI* to cover all advanced analytics concepts, such as predictive and prescriptive analytics, traditional ML, and generative AI.

Increased use of unstructured data and rapid development in the field of AI have given rise to new challenges in data governance practices. The unstructured data in organizations that was not used was also not governed. Now, with this change, the scope of data governance has increased significantly to cover the data lifecycle of unstructured data. The policies and governance models used for structured data may or may not apply to unstructured data, and there might be a need to define new policies and build new governance models.

Moreover, you can create assets apart from data. For example, an ML model trained on your datasets is an asset with intrinsic value derived from the data and needs to be stored, secured, and governed. Therefore, data governance has to include the assets in addition to data, which we like to term *AI assets*. These AI assets have also not been on the radar of data governance, and we would not be exaggerating to say that these have been more or less ungoverned for a long time.

Complying with Increasing Regulations

ChatGPT, the AI-powered chatbot, made AI popular among the general public. Before ChatGPT, AI was some sort of complex system that most of the public wasn't really aware of or interested in. In a way, ChatGPT democratized the awareness and the use of AI-powered applications. This shift in the perception of AI had its network effects. Even the regulators, who usually do not understand technology and often are behind the curve in caring about it, have taken notice and have started working to regulate AI. The European Union (EU) has already come out with the EU AI Act, published in July 2024, which aims to establish a common regulatory and legal framework for AI within the EU.

Even before this significant rise in the popularity of AI, companies had to comply with data privacy regulations such as the EU's General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA), which aim to protect the privacy and security of personal data. This means the burden of making sure that you are compliant with the existing and upcoming regulations for ever-increasing data and related use cases affects the data strategy and innovation possibilities of an organization. Hence, a proper data governance setup is necessary. Moreover, these regulations are often quite ambiguous and not specific to technology, meaning every enterprise has to interpret and apply them as required. There is a lot of pressure on organizations to get this right and not leave themselves open for litigation.

With the newly expanded scope and tightening regulatory standards, the modern data governance solution should be able to cover all data and AI assets while being able to adapt to the latest compliance requirements. Before we go more into the details of a governance solution, we would like to walk you through a brief history of data warehouses, data lakes, and the dawn of the data lakehouse.

The Dawn of the Lakehouse

We increasingly spend most of our time in front of one or the other screens. Work, networking, leisure, and hobbies are increasingly moving online. We are always interacting with a myriad of digital gadgets that are constantly collecting data. Even when you're not looking at a screen, your smartphone knows your location, your smart watch collects your health data, and your car stores data on your driving behavior. Now, imagine the amount of data being collected and processed for billions of people worldwide. With the increase in human population, rapid digitalization, widespread adoption of technology, and our new way of life, where we increasingly choose to coexist with gadgets rather than nature, the volume of data flowing into data platforms has exploded.

Data being collected from nonhuman entities, both tangible and intangible, has also increased significantly. We are increasingly monitoring and measuring everything we can, including machines, digital and physical products, and even nature and its phenomena.

From the perspective of an organization, more data is always better—provided the organization puts it to proper use. The gist of how data helps organizations is that it provides insights and enables them to make better decisions, resulting in significant savings and improved growth opportunities. Savings are usually reflected in productivity boosts and reduced resource wastage. Growth opportunities are due to improved strategies based on the insights derived from data. Therefore, most companies, especially those with a strong focus on technology, have realized how valuable data is, and this has led to a significant improvement in the tools and techniques used to collect data. Moreover, data has now evolved into a product on its own and sometimes acts as raw material that fuels new tangible products—for example, smart thermometers and smart speakers.

Deriving Value from Data

Once the data is collected, the next step is to process it and bring it to a form that is easy to consume and derive insights from. The data collected usually first lands in one of the systems attached to the application that is providing a service. For example, if you register with an ecommerce website, the data that you entered is stored in an OLTP database associated with the web application that you're interacting with. This OLTP database lives either on premises or on the cloud, depending on how technologically advanced the ecommerce company is. If the data is streaming in nature—that is, if the data is flowing in continuously from various sources such as IoT systems—the data might be landing on a disk, cloud object store, or message broker system such as Kafka.

In any case, to process data—especially in large volumes—for analytical purposes, it should be handled on a platform independent of the customer-facing application. This helps avoid overloading the application and prevents delays or downtime that could negatively affect the customer experience. Additionally, you should have the capability to efficiently and securely store, process, and publish the derived data and related assets (see Figure 1-2).

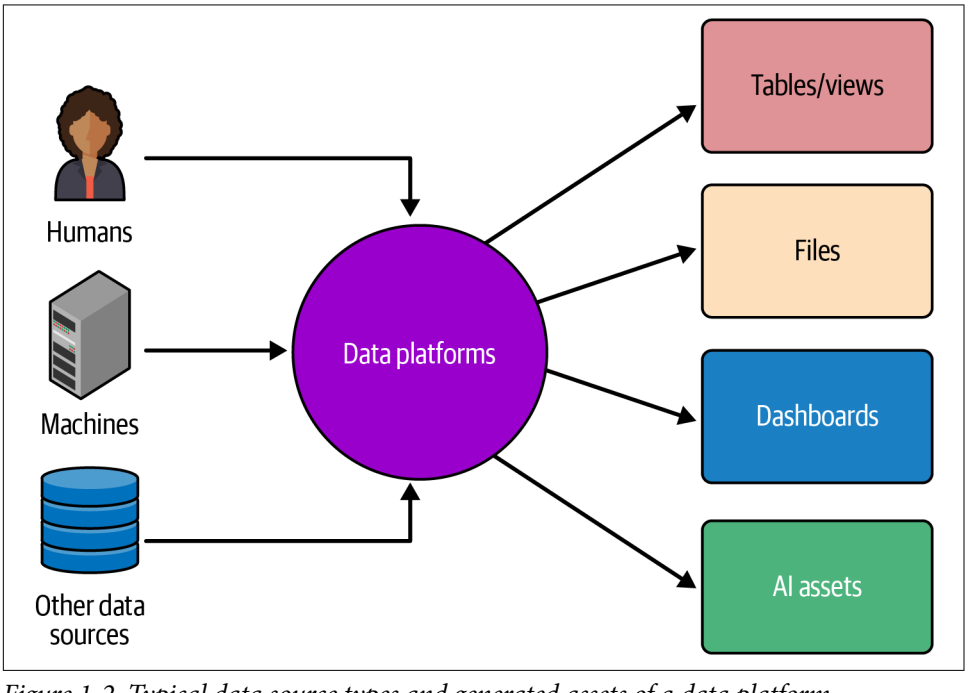


Figure 1-2. Typical data source types and generated assets of a data platform

On a very high level, you would ingest the data from humans, machines, and other data sources and create valuable assets, namely files, tables/views, dashboards, and AI assets. AI assets include feature tables for training ML models, trained models, and so on.

Business intelligence and advanced analytics

Now, if we look a bit deeper, we see that in the analytics space, there are two major sets of use cases:

- Reporting or business intelligence (BI)
- Advanced analytics

The use cases associated with BI usually help organizations gain insights from the past and present to plan for the future. The data that needs to be ingested and processed for these use cases is typically in the form of tables with well-defined schemas.

Advanced analytics, on the other hand, is often used in predictive and prescriptive use cases—the focus is usually on the future. To achieve this, organizations employ AI techniques and technologies. Many common sources of data for these use cases are semi-structured or unstructured in nature. *Semi-structured* data refers to formats such as logfiles, which have consistent patterns and predictable formatting but allow for flexible and variable content. In contrast, *unstructured* data can be any file—such as media files (including MP3 or MP4), images, or plain text—that lacks a predefined organizational structure. This does not mean that you won't use or want to use semi-structured or unstructured data for BI and structured data for advanced analytics. In an ideal world, you would have a platform that enables you to work with all types of data to fuel all kinds of use cases, as shown in Figure 1-3.

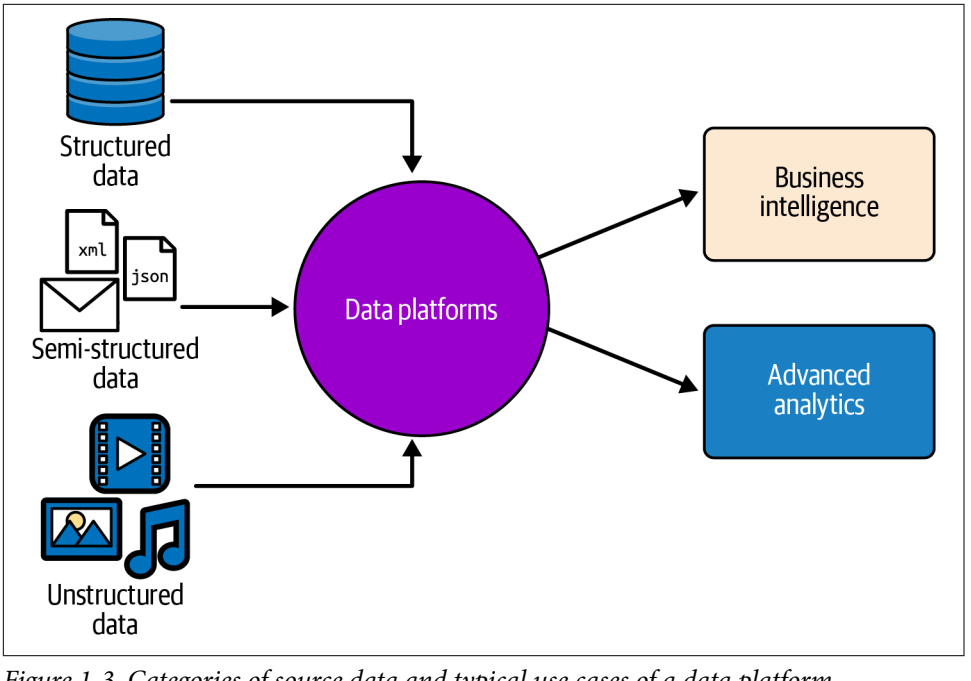


Figure 1-3. Categories of source data and typical use cases of a data platform

Big data begins

The continuous increase in the volume and variety of data has resulted in what we term *big data*.

> Big data is high-volume, high-velocity, and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.
>
> — Doug Laney (Gartner)

If you pay attention to this definition, you can see that it talks about processing the data and the value it creates, in addition to its properties. We believe that when Doug Laney was defining the term *big data* back in 2001, he was convinced that the tools and technologies that existed at that time were clearly not enough for the efficient processing of big data and therefore embedded the need for "innovation" within the definition. He couldn't have been more right!

The three Vs of big data—volume, velocity, and variety—have paved the way for innovation in the tools and technologies required for storing, processing, analyzing, and managing the data. Initially, we believed *volume* to be the major contributor driving this innovation. However, we eventually realized that *variety* is the most challenging one to deal with. That's because many organizations realized that they could derive great insights from data that lives in text files, logfiles, media files, binary files, and so on, which were often never used for any analytical purpose and were either being purged or archived. Moreover, the growth and innovation happening in the field of ML led to the increased demand for these raw data files from data scientists, ML engineers, and the like. Therefore, the tools and technologies that constitute the data platform to be used for dealing with big data had to cater to the variety, which mandated the need for new ways of doing things. This was a paradigm shift in how we deal with data for analytical purposes.

Big data is truly big for only a very few organizations. The majority deal with heterogeneous data that is actually not that large. Moreover, with the platform as a service (PaaS) and software as a service (SaaS) offerings and unlimited scaling offered by the cloud, at least in theory, the *volume* and *velocity* have been truly commoditized. Hence, the time to address the *variety* had arrived.

Data Warehouses and Data Lakes

The trend in the tech industry for the last couple of decades has been to have two types of data platforms: one catering to BI and the other to advanced analytics use cases. The platform type primarily used for BI and reporting is known as the *data warehouse*, and the one primarily used for advanced analytics is the *data lake*. Accordingly, a data warehouse deals with structured data and a data lake often deals with semi-structured and unstructured data.

A *data mart* is a curated database that includes a set of tables designed to serve the specific needs of a single business unit, department, or team—such as the marketing, sales, or engineering department. It is normally smaller and more focused than a data warehouse and generally exists as a subset of an organization's larger enterprise data warehouse.

> If you think of a data mart as a store of bottled water—cleansed and packaged, and structured for easy consumption—the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples.
>
> —James Dixon (former CTO of Pentaho)

On a high level, both data warehouse and data lake solutions have four main aspects:

Ingestion
: Data is collected from various sources.

Storage
: Data is stored for processing and consumption.

Processing
: Data is transformed and prepared for consumption.

Serving
: Data is made available to other systems for reporting, analytics, and more.

Popular examples of data warehouse solutions include Microsoft Azure Synapse Analytics, Amazon Redshift, Google BigQuery, and Snowflake. Each solution offers strengths in one or more aspects such as scalability, performance, ease of use, and integration into the existing ecosystem. However, almost none of them are strong in all the aspects mentioned. These popular, modern data warehousing solutions are cloud native and are gradually replacing legacy solutions such as IBM DB2 Warehouse and Teradata. The main reason is that modern data warehousing solutions provide greater flexibility, scalability, and ease of use. One major factor that enables these improvements is the separation of compute and storage, which allows you to scale and optimize each component separately, resulting in a significant increase in performance and a reduction in costs. One more thing to notice is that some solutions, such as Snowflake, are venturing into the area of data lake with support for processing semi-structured data. In other words, modern data warehouse solutions are trying to cater to advanced analytics use cases as well.

Hadoop: A False Dawn

Back in 2010 or 2011, when Karthik started his career in tech as a software engineer in a fintech startup, the company was adopting a new technology for processing financial data. This new technology promised cost-effective and scalable processing of big data by leveraging commodity hardware with its distributed file system, *Hadoop Distributed File System* (*HDFS),* and processing technique, *MapReduce*. The technology was Hadoop, and Karthik was one of the few lucky ones in the company to have the opportunity to work with the latest and greatest at the time.

Hadoop represented the much-needed paradigm shift for working with data for analytical purposes. Hadoop provided an affordable way to store and process files and to enable advanced analytics with ML. The open source version of Hadoop became so popular that new companies such as Cloudera, MapR, and Hortonworks spun up to provide Hadoop-as-a-Service, with a focus on on-premises setup. For Karthik, at that time, it felt like Hadoop was a technology that was going to last for decades, and the brains behind these companies that centered around Hadoop were geniuses to have capitalized on this opportunity.

Hadoop was introduced at a time when there wasn't really any feasible solution that catered to low-cost big data processing. You could set up Hadoop on commodity hardware, which meant you could put together a bunch of nonpowerful compute resources that you already have and use them as a Hadoop cluster. The data replication and fault tolerance features Hadoop offered made it possible to process big data on these clusters. Moreover, the Hadoop setup is scalable. You could add or remove machines to the cluster and still get the job done. Hence, more and more companies began to find it appealing because they did not need a huge investment to derive insights from big data.

Hadoop MapReduce is a divide-and-conquer technique that leverages the file system. In the early 2010s memory was expensive, so using the file system for processing data was genius. However, this meant the performance wasn't great. Nevertheless, Hadoop at least provided an option to process big data, which was better than nothing. But then things began to change. Data volume continued to grow significantly and memory became cheaper. This meant using memory for affordable processing of big data was becoming a reality.

Fast-forward to today and Hadoop is a legacy system that everyone who adopted it wants to move on from, and for really good reasons. Hadoop was and is still very difficult to use. It is neither performant nor reliable. Moreover, it lacks security and governance. One major good thing that came out of Hadoop's journey was the creation of Apache Spark, which was developed to address the limitations of the MapReduce paradigm.

Cloud data lake

Along with the evolution of analytical data platforms, the adoption of cloud services has been growing significantly in the past decade. The consumption-based model (you pay for only what you use) offered by cloud providers has been a game changer for many organizations, especially startups that cannot write large checks for acquiring on-premises resources.

Cloud data lakes allow you to have all your structured, semi-structured, and unstructured data in raw or native formats. The cloud providers have specific storage services, such as Amazon S3 and Microsoft Azure Data Lake Storage, where you can simply dump all types of data in large volumes. An important aspect here is that the compute is not tied to the storage. This means you can scale storage and compute independently, and you get the flexibility of using diverse compute types with the same data store. Although cloud data lakes native to cloud providers offer a plethora of benefits, there are some challenges related to metadata management, data governance, and security.

Data silos and fractured governance

Karthik lived in Bengaluru, India for three years. It is a great city with lots of career opportunities, a diverse population, and great weather. In the past few decades, it has grown rapidly with a large influx of people from all over India. It grew in area and population without any sort of long-term plans for managing basic infrastructure such as roads, public transport, drainage systems, water supply, and so on. Today, Bengaluru has become a geographically large city with a huge population, terrible traffic, water shortages in some areas, and some areas being flooded during heavy rains.

If you look back on the data landscape of organizations, most of them look like an unplanned city similar to Bengaluru. You see a multitude of systems and data all over the place, without clear access controls or security measures. A large organization may have one or more data warehousing solutions that are used for reporting and BI, with yearly costs amounting to millions of dollars. You will see data lakes used as a dumping ground for data that people don't need or don't know how to use. If you dig deeper, you might come across sensitive information such as PII being stored or accessed in a way that is noncompliant with internal policies, local laws, or even regulations with global reach, such as the GDPR. There might be cases where the engineers, scientists, and analysts using the data have no idea where it is coming from or whether they can rely on the quality of the data provided to them.

You might be thinking, how would organizations end up in such situations? There are several reasons. In companies where things have to move fast, decision makers have little or no time to thoroughly evaluate tools and technologies and might have a limited budget, so they end up using multiple tools often tied together to somehow get things working with minimal costs. This results in *technical debt,* a concept that refers to the future costs and consequences of choosing quick and expedient solutions over more robust and long-term approaches. Everyone plans to pay back the technical debt at some point in the near future, but that rarely happens.

In large organizations with multiple business units, each business unit might have its own choice of tools and technologies, and often similar use cases. In some companies, especially in the regulated industries, it might be quite a lot of work to get approvals for new tools and technologies, making it harder to adopt the latest ones. Moreover, the pace at which the technologies change has significantly increased, and many organizations are not able to keep up. They end up using the latest tools for new use cases and continue with the old ones for the existing setups.

Yet another reason for untidy data landscapes that we often come across is the decision to build versus buy. Sometimes, for the decision makers, it might seem like a good idea to build stuff from scratch instead of using an off-the-shelf solution, either catering to a niche use case or on the often-mistaken assumption that doing so will reduce costs. Moreover, developers and engineers love to create stuff on their own. But eventually, they realize that it is really hard to maintain, scale, and customize such handmade solutions. On the other hand, vendors might influence companies' decision-making process and sell stuff that might not be best suited. Sometimes organizations have no choice but to use different systems to cater to their analytical use cases, as not all of them can be delivered by a single platform.

Let's take the case of data warehouses and data lakes. Two different solutions are employed depending on the nature of the use case and source data. This might also lead to duplication of data, as there might be some datasets that fuel use cases on both systems. Having different solutions means organizations need different governance and security models, simply because the data that they need to govern and secure is of a different nature. In a data warehouse, you need access controls on the tables, and in a data lake, you would need to apply access controls at the file level. The result is an increase in operational efforts and costs, as Figure 1-4 illustrates.

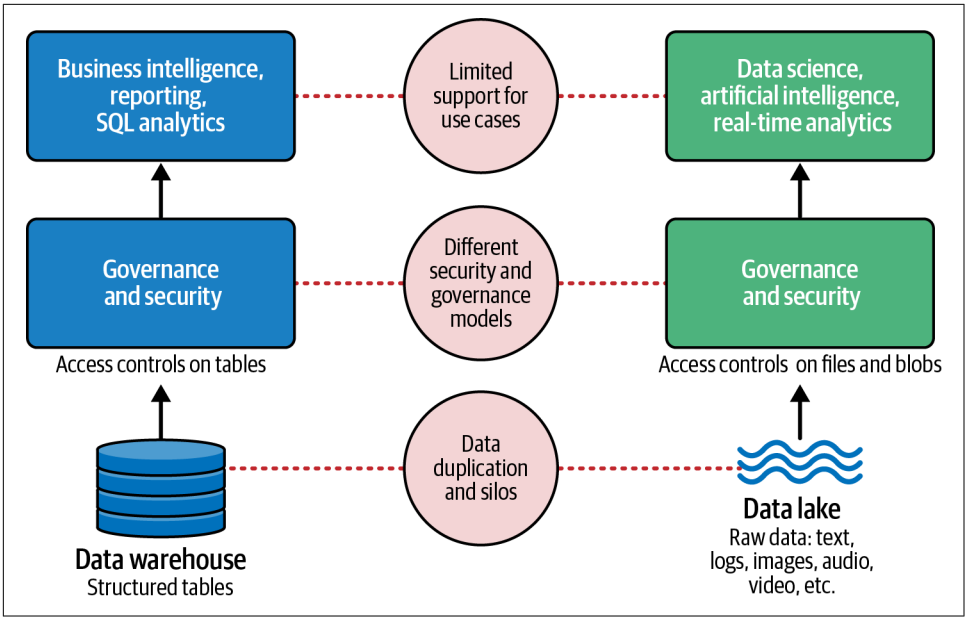


Figure 1-4. Challenges of having disparate systems for BI and advanced analytics

The Lakehouse Paradigm

The issues highlighted in the previous section clearly call for a single solution that caters to all use cases, one that can deal with all types of data along with a unified security and governance model. The *lakehouse* paradigm precisely addresses this need and fulfills these requirements.

Definition: Data Lakehouse

An open data management architecture that combines the best elements of data lakes and data warehouses while enabling unified governance and security models.

The lakehouse paradigm aims to provide a single platform to support the following:

- All the data-related workloads, including BI, advanced analytics, and others
- Storing, processing, and analyzing various types of data
- Unified security and governance models

Figure 1-5 illustrates the data lakehouse idea.

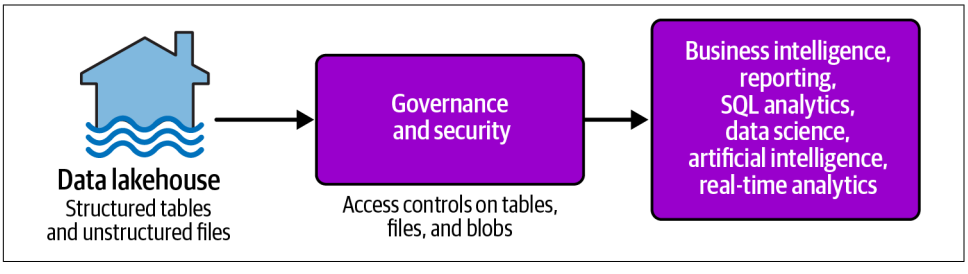


Figure 1-5. Data lakehouse caters to all analytical use cases with a unified governance and security model

The lakehouse architecture

Although the first documented use of the word *data lakehouse* was in 2017, similar architectures have been trending for some time. Projects such as Impala by Cloudera, Presto by Facebook, Spark SQL, and more have attempted to provide a fast SQL experience on top of data lakes. This is essentially an effort to enable using a data lake as a data warehouse with SQL capabilities. Nevertheless, the governance model kept these separate from data warehouse solutions.

Implementing the lakehouse architecture is not trivial. If you start from scratch, where would you begin? You'd begin with a storage that can store a variety of data in large volumes, for example, HDFS, Amazon S3, and Azure Data Lake Storage.

In other words, a data lake that can store files in native formats such as CSV, JSON, MP3, MP4, Parquet, and so on. This form of storage uses the BASE consistency model, and for data warehousing-like capabilities, you need the ACID consistency model. In other words, you need a solution that can provide data warehouse or database-like transactional capabilities to the data lake.

ACID (atomicity, consistency, isolation, durability) and BASE (basically available, soft state, eventually consistent) are two different consistency models used in database management systems. They represent contrasting approaches to handling data consistency and availability in distributed systems:

- ACID prioritizes consistency, while BASE prioritizes availability.
- BASE systems are generally easier to scale horizontally compared to ACID systems.
- ACID is better suited for applications requiring strict data integrity, while BASE is more appropriate for systems that can tolerate temporary inconsistencies.

Relational databases such as MySQL and PostgreSQL are examples of ACID-compliant databases. AWS Redshift is an example of an ACID data warehouse. NoSQL databases, such as Cassandra and Amazon DynamoDB, tend to conform to the BASE model.

This void is filled by the open-table formats, namely Hudi, Apache Iceberg, and Delta Lake. These formats add ACID transactions to data lakes and ensure data consistency and reliability across operations. They build upon and extend the Parquet format's capabilities. A more suitable term for these formats is *data management framework,* as they offer a lot more than just being table formats. For instance, they support *schema evolution*, which is changes to the data structures without disrupting the existing data; *time travel*, which is the ability to query historical data and rollback to previous versions; and *efficient metadata handling* to optimize query performance and reduce latency for large datasets.

Then comes the compute layer, a high-performance and massively scalable query engine such as Apache Spark for efficient data processing. The query engine runs on compute resources that are decoupled from the storage. On top sits a data processing layer that supports batch processing, stream processing, ML, and other advanced analytics workloads. Finally, an adjacent security and governance layer, as it is necessary to ensure data protection and compliance. A few key aspects of this layer include access controls, authentication mechanisms, data encryption, data lineage, and auditing capabilities. Figure 1-6 illustrates the lakehouse architecture.

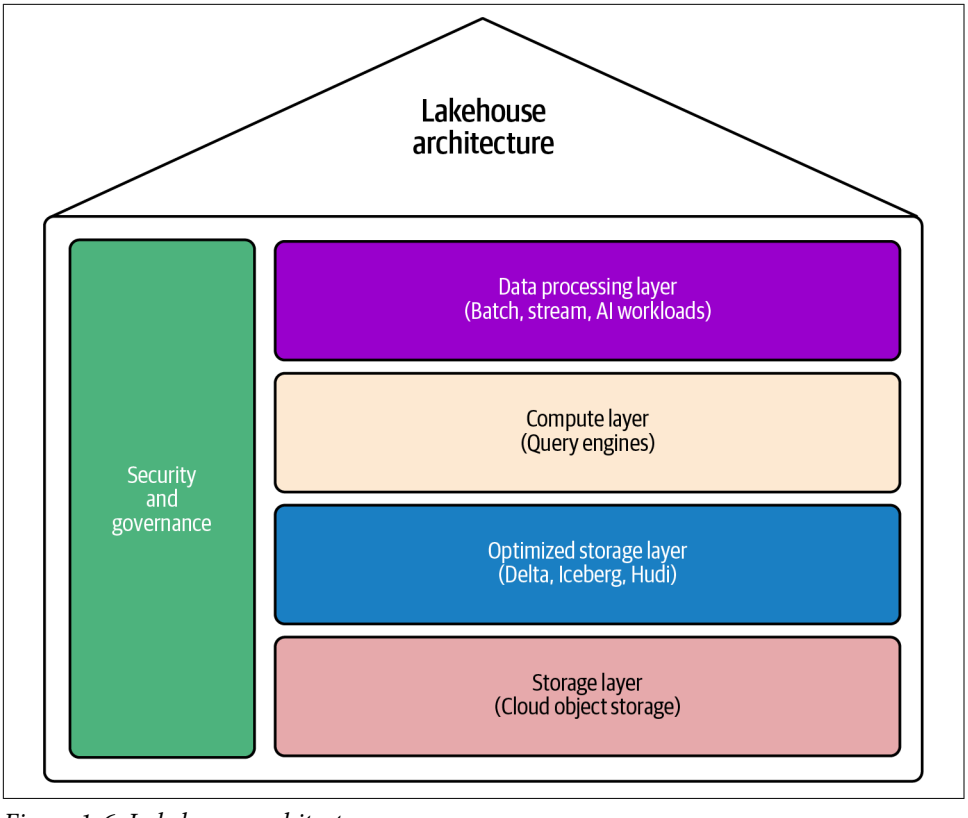


Figure 1-6. Lakehouse architecture

The best example of a lakehouse implementation is probably the cloud-based platform offered by Databricks, a data and AI company founded in 2013. Databricks has helped thousands of organizations adopt the lakehouse architecture. Therefore, we are going to delve into the concepts of lakehouse architecture in the context of the Databricks Data Intelligence Platform.

Medallion lakehouse architecture

Medallion architecture, also known as multihop architecture, describes a layered pattern for processing the data. Data goes through layers of processing during which it is incrementally improved and enriched. This architecture is preferred for online analytical processing (OLAP) workloads for the following reasons:

- It guarantees ACID for the data as it passes through different processing layers (validation, transformation, etc.).
- Data quality and usability increase as the data undergoes different stages of transformation. An example implementation would include terms such as bronze (raw), silver (cleansed/validated), and gold (curated/enriched) that indicate the quality of data.
- The processed data (usually termed the gold layer) is stored in a read-optimized layout for efficiently running analytical workloads.
- It results in streamlined workflows and enhanced data governance.

Figure 1-7 illustrates medallion lakehouse architecture.

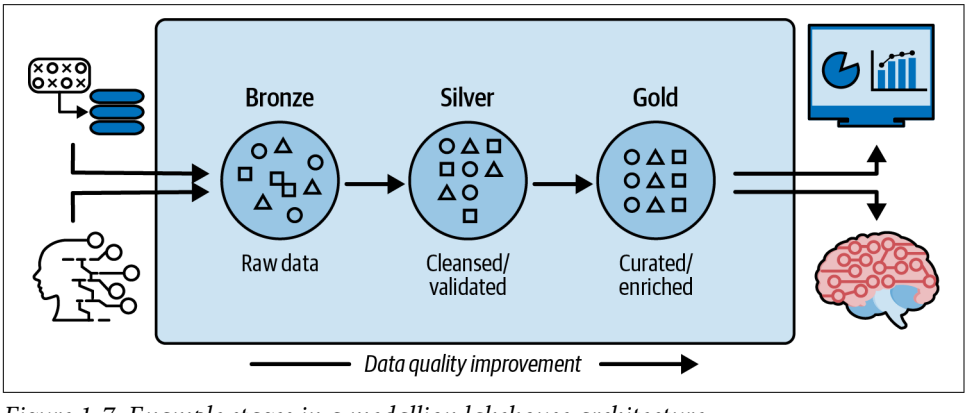


Figure 1-7. Example stages in a medallion lakehouse architecture

Databricks Data Intelligence Platform

Databricks has its origins in Apache Spark, with a focus on data engineering workloads. Eventually, it expanded its scope to include data science and traditional ML. Recently, it has ventured into data warehousing and generative AI, covering all the BI and advanced analytics use cases.

Databricks is a unified, open, and intelligent data and AI platform that enables the lakehouse architecture to address all data—and AI-related use cases.

To elaborate on this definition:

- Databricks, powered by the lakehouse paradigm, addresses the limitations of data warehouses and data lakes by providing a unified platform for all data and AI-related use cases.
- The term *unified* also refers to the single governance model applicable to all data and AI assets.

- Databricks is built on open source technologies, namely Apache Spark, Delta Lake, MLflow, and Unity Catalog.
- Databricks is boosted by the contextual intelligence generated using organization-specific information.

Figure 1-8 shows the core components of the Databricks Data Intelligence Platform.

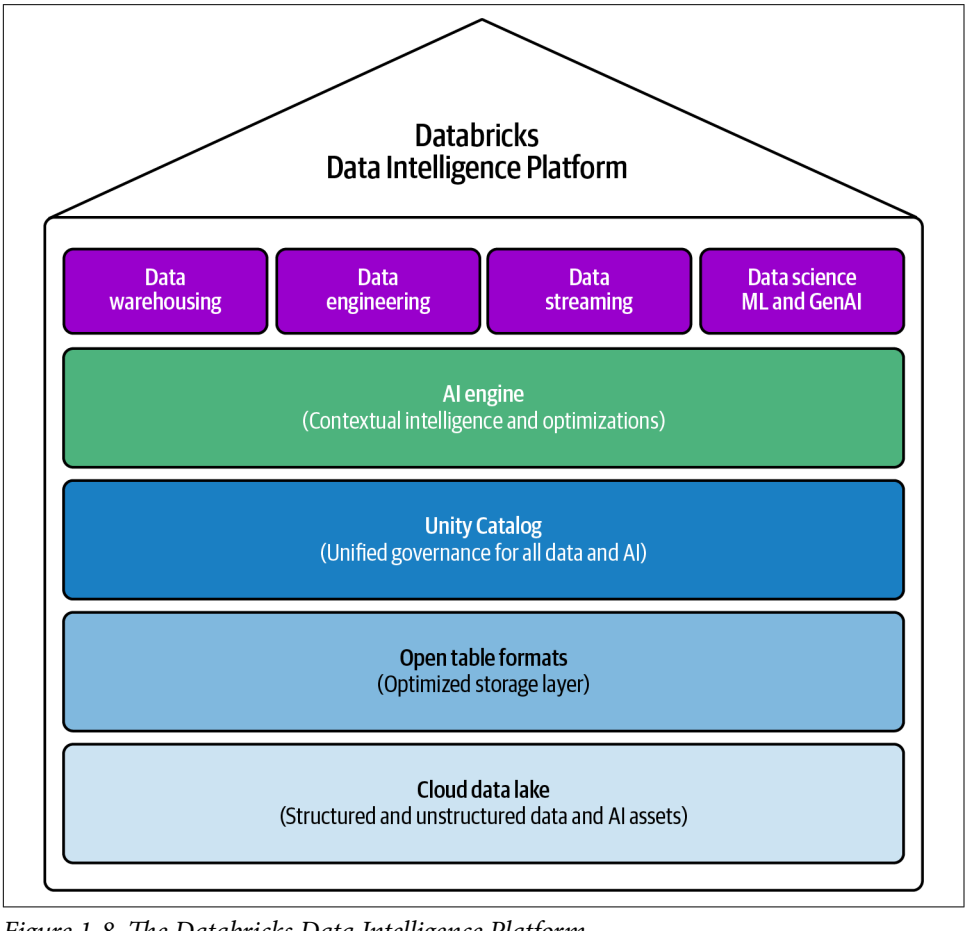


Figure 1-8. The Databricks Data Intelligence Platform

Here's a closer look at these core components:

Cloud data lake
: A cost-effective object storage for all structured and unstructured data and AI assets.

Open table formats
: An open source and optimized storage layer on top of the cloud object storage that supports ACID transactions, scalable metadata handling, and unification of streaming and batch data processing. Databricks started out with Parquet, then developed Delta Lake, and it is gradually extending its support to Apache Iceberg and Hudi.

Databricks Unity Catalog
: A unified catalog with governance features built into the Databricks Platform. Unity Catalog features include a metastore, search, and discovery of data and AI assets, access controls, lineage, audit logs, and other advanced governance features.

AI engine
: An intelligence engine embedded into the Databricks Platform that leverages contextual information to accelerate development with support for coding and debugging, and drive optimizations in terms of storage layout, compute configurations, and so on.

The Databricks Platform, with all its features, can house all data—and AI-related workloads. In other words, it caters to all the personas in modern data teams, namely data engineers, data scientists, data analysts, ML engineers, and AI engineers.

Databricks Unity Catalog: Enabling Unified Governance

Databricks announced Unity Catalog at the Data and AI Summit in 2021. A gated public preview of Unity Catalog on Azure and AWS was announced in April 2022. We joined Databricks in 2022 when Unity Catalog was still in its infancy in terms of enterprise adoption. Core governance features such as the metastore, integration with identity management systems, coarse-grained access controls, and more were available, but a lot of advanced features were still in development. As field engineers, we got to experience the journey of Unity Catalog adoption across thousands of organizations, some of which we directly contributed to. And we got to witness the rapid evolution of Unity Catalog. Today, it is perhaps the only solution that enables unified governance for all data and AI assets. Let's now double-click on Databricks Unity Catalog and see what it has to offer.

Introducing Unity Catalog

Karthik remembers a conversation with a key decision maker from one of the largest retail organizations that had adopted Databricks as their core data platform. Let's call him Bob. Prior to Unity Catalog, Hive Metastore (HMS) was used as a metadata repository for all the data assets stored in Databricks. Bob had convinced all the stakeholders in his department to migrate to Unity Catalog from the legacy HMS.

He mentioned that most of his colleagues from the data teams initially did not understand what Unity Catalog was. They thought it was just another catalog that replaced HMS and believed that the effort to migrate all their data assets to Unity Catalog was not justified. Ultimately, Bob had to clarify that Unity Catalog is much more than just a catalog. It comes with solid governance features that simplifies data security and access and is a necessary component of the Databricks Platform. Bob told them, "Unity Catalog is not just a catalog, but an evolution of Databricks."

Definition: Apache Hive

Apache Hive is a distributed data warehousing and SQL query language for Hadoop. It allows users to process large datasets stored in cloud storage (AWS S3, ADLS, Google Cloud Service [GCS]) or on-premises HDFS using standard SQL queries.

Bob was absolutely right. Unity Catalog simplifies governance for the data stored in or accessed via Databricks. Moreover, it's a core part of the Databricks Platform and is required for its operation. Therefore, it can be referred to as an *operational* catalog. It can be further described as a *unified* and *open* solution. The reasons for using these terms are as follows:

Unified

- Unity Catalog, as part of the Databricks Platform, can be used on any of the major cloud providers. Therefore, organizations aiming for a multicloud strategy can continue to have unified governance across all their data platform(s), provided Databricks is part of their multicloud strategy.
- Unity Catalog supports multiple open source lakehouse formats: Delta, Apache Iceberg, Hudi, CSV, JSON, Parquet, and so on, thereby providing a unified governance for all data formats.
- Unity Catalog provides a single governance model for all the data and AI assets.

Open

- Unity Catalog being open source prevents vendor lock-in.
- The credential vending feature allows for temporary access to data and AI assets governed by Unity Catalog from external engines using Unity Catalog REST APIs, which paves the way for interoperability.
- The catalog federation feature further enhances interoperability by supporting metadata exchange with other catalog solutions in the industry, such as AWS Glue.

It is important to note that Unity Catalog is open source and can be installed and used on any cloud or on-premises infrastructure independent of Databricks. Databricks offers a managed version of Unity Catalog, which is a core part of the Databricks Platform. Moreover, additional components within the Databricks Platform make it much more feature-rich. In simple words, the *Unity Catalog open source software (OSS)* is a unified data and AI catalog, and *Databricks Unity Catalog* is a unified governance solution for data and AI assets on Databricks.

Databricks Unity Catalog is a governance solution in the context of the Databricks Platform and a bit beyond. It does not include some of the features, such as data stewardship workbenches and compliance reporting, that you might find in traditional data governance tools. However, some of these features, for example, compliance reporting, can be developed on top of Unity Catalog by leveraging its features and the metadata it offers. Unity Catalog integrates well with other tools in the data ecosystem. Many organizations use enterprise data catalogs such as Collibra, Alation, Atlan, etc. to fulfill organization-wide data governance requirements.

Access control for data and AI assets

The governance model of Unity Catalog includes access controls to data and AI assets—in other words, objects that are secured by Unity Catalog, as shown in Figure 1-9.

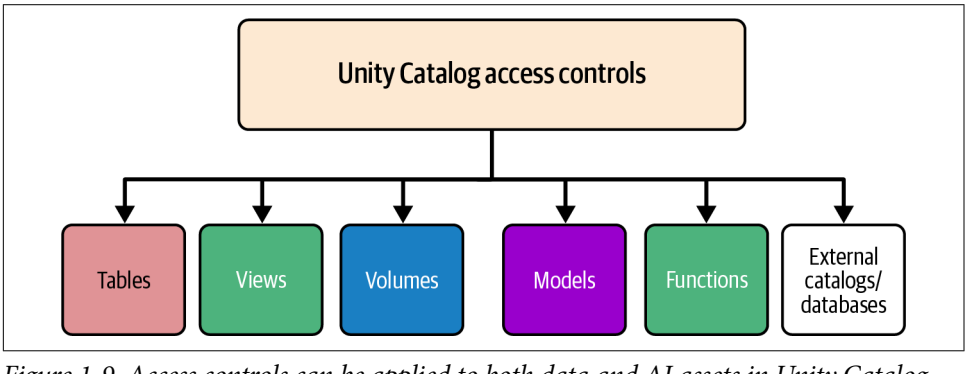


Figure 1-9. Access controls can be applied to both data and AI assets in Unity Catalog

Data and AI assets include the following:

Tables
: Collection of data organized as rows and columns.

Views
: Read-only objects created from one or more tables and views.

Volumes
: Logical entities consisting of arbitrary files.

Models
: AI models packaged using the MLflow framework.

Functions
: Unit of saved logic defined by a user that returns a scalar value or a set of rows.

External catalogs/databases
: Bring your own databases, data warehouses, or catalogs, such as AWS Glue.

Core governance features

Let's first look into the core features offered by Databricks Unity Catalog that enable the governance of data and AI assets:

Authentication
: Unity Catalog integrates with external identity providers such as Microsoft Entra ID, Google, Okta, and others for authentication.

Data cataloging
: Unity Catalog offers a metastore where all the data and AI assets are cataloged.

Authorization
: It allows for the authorization of identities to control access to data and AI assets.

Data discoverability
: The data and AI assets cataloged within Unity Catalog and metadata, such as tags, are searchable with natural language search.

Audit logs
: Operations on objects cataloged under Unity Catalog are automatically logged and made available.

Lineage
: All data and AI assets, i.e., tables, views, volumes, and models, have lineage automatically generated and accessible within the Databricks Platform.

Advanced governance features

The following are the advanced governance features within the Databricks Platform that are unlocked by Unity Catalog:

Fine-grained access controls (row-level security [RLS]/column-level masking [CLM])
: With Unity Catalog, you can apply row-level filtering or CLM to tables, which is usually required for use cases involving sensitive data.

Attribute-based access controls (ABAC)
: ABAC enables conditional access control mechanisms based on certain attributes such as identity, resource, request, and so on. ABAC empowers organizations to fulfill complex governance requirements by providing enhanced capabilities that can scale.

Quality monitoring
: Unity Catalog enables you to monitor the statistical properties and quality of the data at the table level in an automated way. The respective stakeholders could be notified of any anomalies or issues with the datasets and help rectify them on time.

The next chapters go deep into every feature that Unity Catalog provides and discuss how best to use them. We will also provide relevant examples and best practices.

Databricks Platform Architecture

We described Databricks Unity Catalog as a governance solution, but to comprehend what it is in terms of system architecture, we need to first explain the Databricks Platform architecture.

On a very high level, the Databricks Platform has four main components. The most important component is *you*—that is, the users and applications that access the platform. The platform itself is then divided into the second and third components, the *control plane* and the *compute plane*. The final component is the storage where all your data and AI assets are persisted. Figure 1-10 illustrates the architecture.

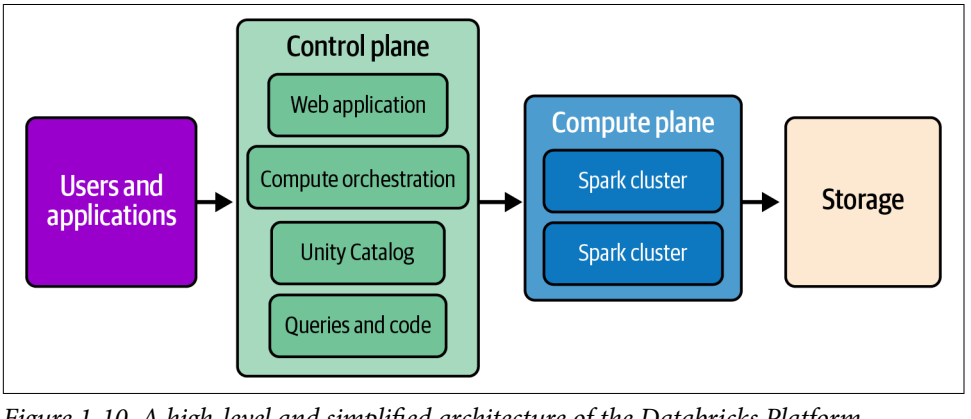


Figure 1-10. A high-level and simplified architecture of the Databricks Platform

Control plane

The first interface that you interact with whenever you talk to the platform is the *control plane*. The control plane is where all the crucial services of the platform run. These include the web application, orchestration service for the compute instances, the Unity Catalog service, etc. This is the layer that does the authentication and authorization of users. The control plane is the first point of contact, even when you talk to the platform via the APIs.

Compute plane

The compute plane is where all the processing happens. In other words, all the workloads are run in the compute plane, most of which are on Apache Spark clusters. These clusters are a collection of compute instances with one driver node and zero or more executor nodes that employ the Apache Spark framework for executing workloads. Databricks uses a curated execution environment known as Databricks Runtime (DBR) on all its compute instances.

The Apache Spark framework has been a core part of the Databricks Platform offerings for quite some time. However, some of the latest Databricks offerings, like model serving or fine-tuning services, do not use Apache Spark.

Unity Catalog components

Let's now dissect Unity Catalog and examine its anatomy within the Databricks Platform. When we refer to Databricks Unity Catalog as a governance solution, it can be thought of as a combination of three main components: the UI, the Unity Catalog service, and the DBR:

UI
: The Unity Catalog UI is embedded within the Databricks UIe, where you, as a user, can perform various actions such as discovering data and AI assets, creating these assets, controlling access, tagging, monitoring, and more.

Unity Catalog service
: The Unity Catalog service, which runs in the control plane of the Databricks Platform, is the main component that performs the actions that materialize as governance features. For instance, when a user executes a query against an asset cataloged under Unity Catalog, the Unity Catalog service verifies whether the user has access to that asset and responds accordingly. The Unity Catalog REST API is also part of the Unity Catalog service.

Databricks Runtime

The DBR is a set of core software artifacts that run on all the compute instances managed by Databricks. It is designed and regularly updated to significantly enhance the performance and security of workloads. Some aspects of Unity Catalog governance features are embedded within the DBR—for example, fine-grained access controls are enforced at the compute level, and hence, the DBR plays a crucial role. A certain version of the DBR is required to even use Unity Catalog within Databricks, and some of the latest features require upgrading the DBR version used in the compute instances.

DBR version 11.3 or higher is required to use Unity Catalog on Databricks compute instances. However, it is recommended to use at least DBR 13.3 or a later version, as many significant Unity Catalog features have been added in that Long-term Support (LTS) release. In general, always choose the latest DBR LTS version to benefit from the most up-to-date supported features.

Data Sharing and Collaboration

One of the important aspects of modern-day governance is that you have to consider requirements beyond your borders. The biggest challenge with governance that we saw earlier was to break down silos within the organizations and foster collaboration. The reasons for silos/borders can be organizational structures, different geographies, and other things inherent to the data strategy, such as a multicloud setup. Other reasons include regulatory requirements, historical reasons, or even because of internal politics. Whatever the reason, you should design the governance approach by considering and catering to the needs of individual entities, which can be teams, departments, or business units, and find a way to unify and standardize at least some aspects of it.

Imagine that all the different entities that are independent in their operations have a common language that they can speak with other entities. In other words, imagine they get a single interface and similar processes based on a common technology stack to share and consume data and AI assets across the organization and even beyond. Moreover, what if they could continue to maintain the quality and govern the shared assets even if they are being shared across borders? This would enable organizations to materialize a standard yet collaborative governance approach. It would make it possible to implement data architecture such as a data mesh even in large organizations.

This is possible with *Delta Sharing*, an open protocol for secure data sharing. Because Delta Sharing is open source, you can set up and use it as a standalone solution. With Databricks, it is available as a managed solution that requires Unity Catalog. In other words, on the Databricks Platform, Unity Catalog enables secure data sharing

via Delta Sharing. The Delta Sharing protocol is the common language that different entities within an organization can adopt and leverage for data sharing and collaboration. Based on this, each entity can define similar processes, with Databricks being the common platform across the board.

Delta Sharing is the foundation for other collaborative product offerings from Databricks, namely the *Databricks Marketplace* and *Databricks Clean Rooms*. Databricks Marketplace is an open forum for publishing and consuming data products with a business-user-friendly interface. Databricks Clean Rooms are secure and private spaces for collaboration among different entities within or across organizations. Clean rooms provide a secure environment for combining datasets with sensitive information without compromising privacy or security. Chapter 8 dives deeper into Delta Sharing, Databricks Marketplace, and Databricks Clean Rooms.

Summary

This chapter showed why governance matters, and as the data volume and variety keep increasing along with their use cases, the importance of securing and governing them also increases. Modern day problems need modern data solutions, and we saw how the desire to unify the data warehousing and data lake capabilities led to the genesis of the data lakehouse. The Databricks Platform, the best example of the lakehouse implementation and Unity Catalog as a core part of it, simplifies and unifies the data and AI governance.

The following chapters cover various aspects of Unity Catalog and some aspects of the Databricks Platform, and we will use our fictional organization, Nexa Boutique (NB), to make them more comprehensible. So, let's begin our journey to explore the lakehouse paradigm, Unity Catalog, and Delta Sharing in the context of the Databricks Platform.

CHAPTER 2
Unity Catalog Under the Hood

In 2021, while the data architects at Nexa Boutique (Nexa) were analyzing data governance strategies for the firm, Databricks announced Unity Catalog. This native data governance solution for the Databricks Platform provided a more capable alternative to HMS. Unity Catalog unifies governance for all Databricks Platform assets, including tables, views, ML and AI models, vector tables, and files in Unity Catalog Volumes across workspace boundaries.

Organizations typically work with multiple data formats across their data estates, generated by disparate source systems, which warrant multiple data governance tools—a challenge that Nexa also faced. Using numerous bespoke data governance tools within your data estate leads to fragmented governance, which is often not a deliberate choice. Unity Catalog addresses the issue of fragmented governance by providing a unified and cohesive product for governing your data and AI assets.

This chapter covers Unity Catalog's quintessential architecture. It explores what is under the hood and the features that make it an ideal catalog for meeting your governance needs.

The story of Unity Catalog begins with the history of governance in Databricks, specifically the limitations of the HMS in addressing data governance needs. As we explore the evolution of governance in Databricks and the challenges posed by HMS, you'll begin to appreciate the need for a more robust and scalable governance solution, ultimately leading to Unity Catalog as a central governance tool in Databricks. We provide an overview of the architectural pillars that underpin Unity Catalog and the guiding principles that shape its design. This foundation will help you understand the inner workings of Unity Catalog and how it enables effective governance.

We explore how to build a well-governed data platform using the Unity Catalog model. We also share a quasi-real-world example of how Nexa successfully adopted

the Unity Catalog model in its organization, highlighting the benefits and challenges the company encountered during its journey.

We cover Unity Catalog's key data-management capabilities, including securable and access controls. You'll learn how to use these features to ensure secure and controlled access to your data assets.

The Governance Story So Far

Before diving into the architecture of Unity Catalog, it's essential to understand the motivations behind its creation. What limitations in existing data governance features on top of an HMS-based data catalog led to the creation of Unity Catalog? To answer this, let's take a step back and examine how data governance worked in the traditional HMS setup and how HMS interacted with Apache Spark, the data-processing engine at the heart of Databricks. The table access control list and credential passthrough features worked in tandem to close most of the gaps in governance. We will dive into the details of this integration below. We will also examine how the Databricks architecture looked at Nexa before onboarding to Unity Catalog.

Here are a few key governance milestones on the Databricks Platform:

- 2015: Databricks Platform integrated with HMS
- 2017: Table Access Control Lists (TACL) introduced
- 2018: Azure Data Lake Storage (ADLS) Gen1 credential passthrough introduced for Azure
- 2019: ADLS Gen2 credential passthrough for Azure and Identity and Access Management (IAM) passthrough for AWS (Amazon Web Services)
- 2021: Unity Catalog introduced
- 2024: Unity Catalog became open source

Definition: Apache Spark

According to the Apache Spark website, "Apache Spark started as a research project at the UC Berkeley AMPLab in 2009 and was open-sourced in early 2010. Many of the ideas behind the system were presented in various research papers over the years. After being released, Spark grew into a broad developer community and moved to the Apache Software Foundation in 2013."

Hive Metastore as the Default Catalog

The year 2010 saw the release of Apache Hive, and the HMS service followed in 2013 when on-premises Hadoop clusters dominated the data landscape. As mentioned in Prologue: Governance by Choice, Nexa Boutique also began its data journey with Hadoop clusters powering its central data platform (CDP). The CDP team had accumulated and managed nearly 10 petabytes (PB) of data in the company's on-premises Hadoop clusters. HMS played a crucial role in Nexa Boutique's initial architecture, serving as a central repository for metadata. The data engineers, analysts, and scientists at Nexa were well-versed in the HMS way of working and had grown accustomed to its capabilities over the years. When these users started onboarding to Databricks, they found themselves in familiar territory.

As a result of its early arrival, preceding the widespread adoption of cloud-based solutions, HMS gained broad acceptance and support from all the major data processing tools and applications, including Apache Spark. This era was also marked by the *file format controversy*, with Apache Parquet, Apache ORC, and Apache Avro competing for supremacy in the big-data storage format space. However, more recently, the industry has witnessed a convergence toward open lakehouse formats like Delta, Iceberg, and Hudi, which have gained popularity due to their improved performance, scalability, and transactional features.

These storage formats only differ in their metadata and use Parquet as the underlying file format. Databricks also released the UniForm compatibility, eliminating the requirement for separate file formats and transaction logs, unifying it all with Unity Catalog, and ultimately aiming to end the ongoing file format controversy. Figure 2-1 illustrates how UniForm automatically generates metadata, allowing Delta Parquet files to be accessed as either Hudi or Iceberg, without requiring duplicate files for different data consumers.

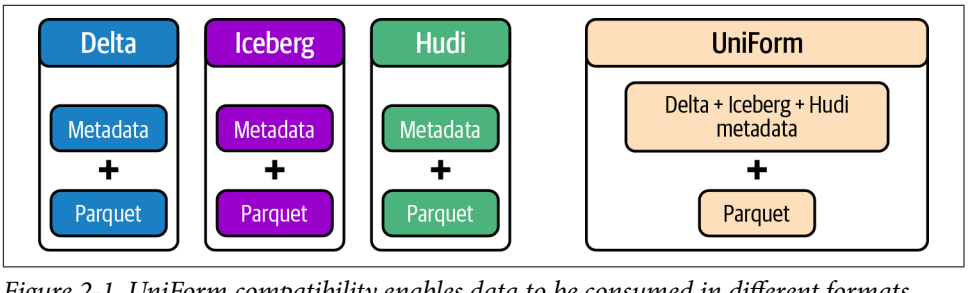


Figure 2-1. UniForm compatibility enables data to be consumed in different formats

In its most simplified form, HMS can be considered a centralized registry of tables and their schema within a data warehouse, mapping them to the physical locations of the underlying files that store the data, as illustrated in Figure 2-2. Although HMS efficiently maps table metadata to storage locations, enabling faster query performance, it falls short of providing robust data governance capabilities over the data. When deploying an HMS in Databricks, you can opt for the Databricks-managed Relational Database Management System (RDBMS) instance or configure your own external metastore, a self-hosted RDBMS instance. Using the self-hosted HMS, you can share metadata across multiple Databricks workspaces, overcoming the limitation of the Databricks-hosted version but adding overhead of maintenance, configuration of compute resources, and cost.

Additionally, for users on the AWS cloud, it is also possible to connect to the AWS Glue catalog as an external HMS in a Databricks workspace. As a multicloud enterprise, Nexa had Databricks workspaces deployed across multiple cloud providers, including Microsoft Azure, AWS, and Google Cloud Platform (GCP). Azure and GCP workspaces used the managed HMS, whereas AWS workspaces utilized the AWS Glue catalog as its HMS service.

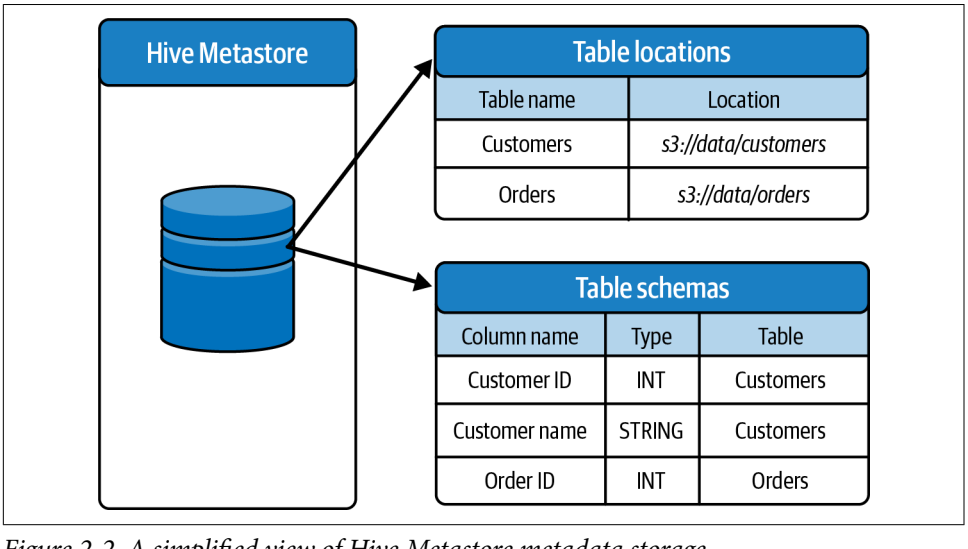


Figure 2-2. A simplified view of Hive Metastore metadata storage

At Nexa, the CDP team managed the deployment and administration of the Databricks workspaces for all their business domain teams. The Logistics and Supply Chain (LSC) domain was the primary user of the platform and alone accounted for up to 25% of the entire platform expenditure. The LSC domain is a complex and multifaceted area comprising the following five distinct subdomains, each with unique challenges and opportunities:

- Demand planning
- Inventory management
- Warehouse operations
- Purchasing
- Transportation

Within each subdomain, various applications supported specific business use cases, which were distributed across different systems. These included on-premises transactional systems that utilized MySQL and Oracle databases, as well as cloud-based SaaS applications, such as Salesforce. To support their business, the CDP team deployed over 100 workspaces for the LSC domain under multiple global cloud regions. Nexa's sales strategy relies heavily on historical data analytics and predictive modeling, enabling the company to identify trends, optimize pricing, and forecast demand.

Managing user onboarding and HMS across these workspaces was adding additional operational overhead even though the CDP team automated most of their infrastructure deployments using infrastructure as code (IaC) tools like Terraform. The CDP team managed user provisioning by integrating their identity provider, Microsoft Entra ID, through the System for Cross-domain Identity Management (SCIM) API integration. SCIM integration needed to be done individually across all workspaces, as there was no provision for identity federation from a central location.

Definition: System for Cross-Domain Identity Management

The SCIM specification simplifies user identity management in cloud-based applications. It builds on existing schemas and deployments, prioritizing ease of development and integration while leveraging established authentication, authorization, and privacy models. The goal: fast, affordable, and easy user management in the cloud.

Figure 2-3 illustrates the logistics domain architecture utilizing HMS in the US, highlighting the intricacies of their data infrastructure. We have focused on the US's demand planning and purchasing subdomains to simplify the representation. However, the actual architecture becomes significantly more complex when considering all domains and subdomains across multiple regions, encompassing 500 Databricks workspaces.

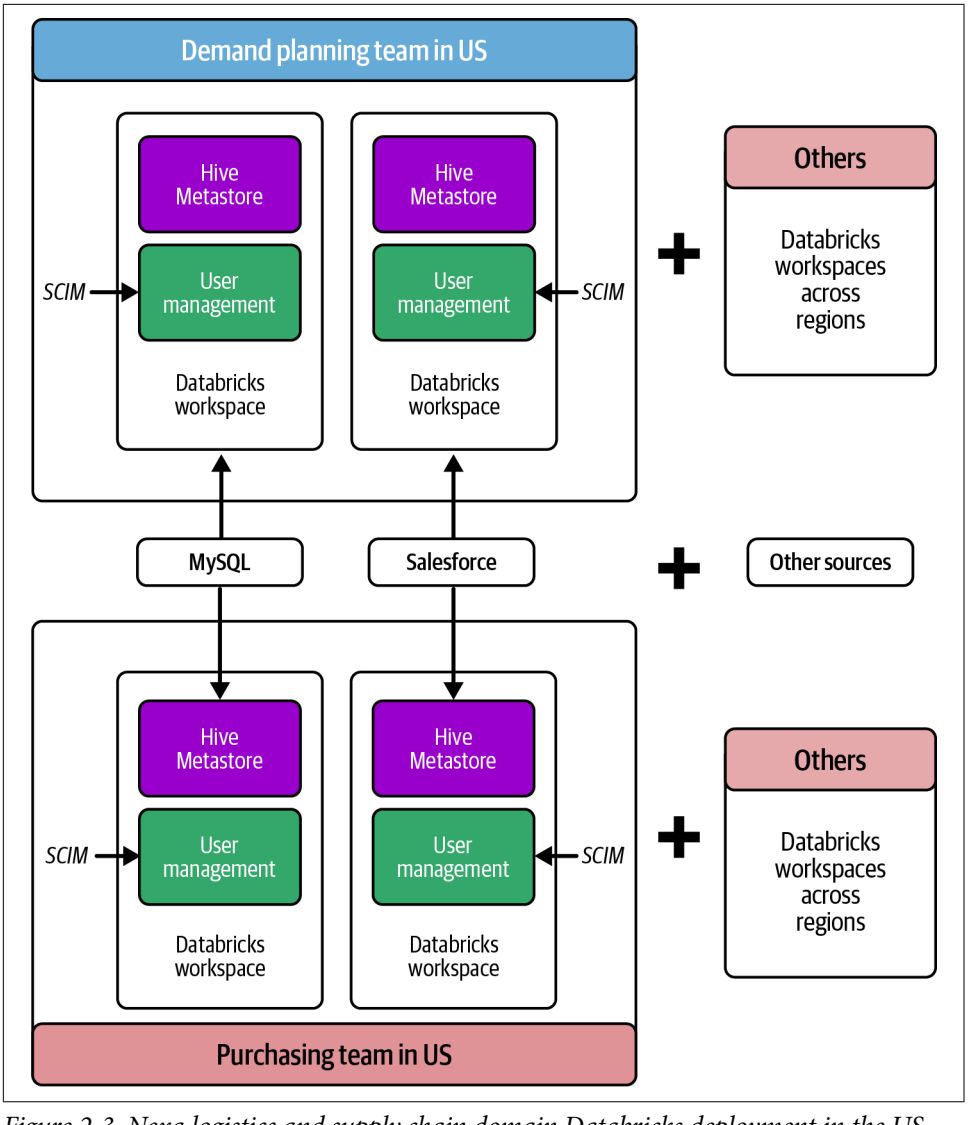


Figure 2-3. Nexa logistics and supply chain domain Databricks deployment in the US

The Dilemma of Governance in Hive Metastore

The HMS has evolved significantly in its mission to deliver robust data governance to its users. Administrators attempted to meet data governance requirements in the early stages of HMS by segregating users and applications at the cluster level and assigning each their own dedicated cluster. However, this approach had inherent limitations in the on-premises Hadoop clusters and major cloud data processing platforms, including the initial versions of Databricks that used HMS, which led

to increased spending and operational burden. The introduction of credential passthrough in Databricks mitigated some of these limitations, allowing users to share a cluster when accessing file-based data. Nevertheless, Spark and Python-based access continued to pose challenges. The TACL on top of HMS overcomes these limitations. This section provides an in-depth examination of these advancements and explores how they have collectively contributed to more comprehensive data governance in HMS over the years.

Isolation at cluster level

Historically, Spark enforced access control at the cluster level, using the Spark cluster as the boundary for data isolation within the HMS. By default, the HMS attached to a Spark cluster would grant access to all the metadata it managed. Additionally, in Databricks on AWS, an instance profile associated with a Spark cluster could be used to access S3 buckets, allowing for read or write permissions to the files. Azure users authenticate to an ADLS account with a service principal, whereas GCP users authenticate to a GCS with a service account.

Coupling credentials to a Spark cluster can become a bottleneck in a multitenant or multiapplication environment. Assigning a dedicated cluster to each user or application is expensive and wasteful, as it can lead to underutilization of resources. The approach of data isolation at the cluster level poses a significant security risk, as it allows anyone with access to the cluster to read sensitive data stored in cloud storage. When credentials are attached to the cluster, they can be used to access the data, regardless of the user's role or permissions. As a result, there is a clear need for more granular access control. Administrators mitigated this risk and achieved data isolation by provisioning separate Spark clusters for individual users or applications, as Figure 2-4 depicts.

Provisioning a separate Spark cluster for each user or application, with customized access to data, introduced several challenges, including the following:

- Cost of maintaining complex infrastructure.
- Access is very coarse-grained, as there is no way to provide row-level or column-level access.
- To prevent toxic combinations (see below), administrators must plan and control the resources accessible to a cluster.
- Increased complexity in managing multiple clusters, leading to higher operational overhead.

- Processes running on each cluster often fail to fully utilize the available resources, leading to inefficient resource utilization.
- Difficulty in scaling, as adding new users or applications requires provisioning new clusters.

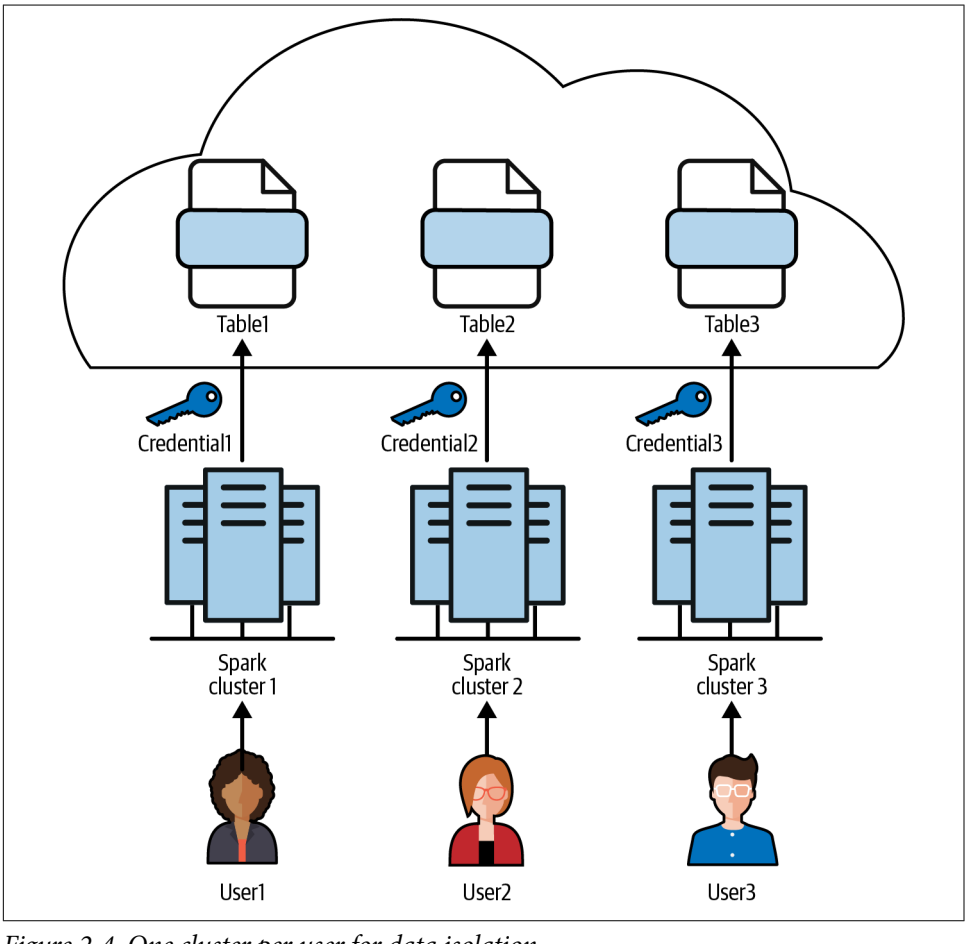


Figure 2-4. One cluster per user for data isolation

Definition: Instance Profile

An instance profile is a container for an IAM role that passes role information to an EC2 instance at startup. When attached to a Spark cluster, it grants users with cluster access the same permissions as the instance profile, potentially exposing all accessible AWS resources to those users if not properly managed.

Definition: Toxic Combination

A toxic combination happens when several harmless-looking elements, like vulnerabilities or weak settings, join forces to create a bigger and more serious security problem. This can occur when different systems, apps, or services work together to create a weak spot, making it a hidden danger that attackers can exploit.

By installing additional tools like Apache Sentry or Apache Ranger, administrators can provide fine-grained access controls on top of the HMS tables, allowing for more granular data access and permissions management. However, these tools had a significant limitation: they could enforce access control only for SQL-based access out of the box, leaving a gap in the security of data accessed through DataFrames or Python-based file access. Custom plug-ins enable file-based access controls, including DataFrame APIs, for systems such as the HDFS or the NameNode in a Hadoop cluster. However, implementing custom plug-ins can be highly invasive and is often not feasible for opaque, cloud-hosted SaaS solutions. Furthermore, relying on custom plug-ins can make the overall architecture brittle and prone to fragility, as changes to the plug-in can have unintended consequences on the system as a whole. With the pace at which Databricks upgrades the Databricks runtime, having plug-ins could hamper new feature adoptions due to compatibility issues. Plugging in all the security controls gets complex, especially regarding cloud-based SaaS applications and cloud storage. SQL-only enforcement meant that users could bypass access controls using non-SQL methods to access sensitive data.

Definition: Apache Sentry

Apache Sentry (retired) enforces fine-grained role-based authorization for data and metadata stored on a Hadoop cluster.

Definition: Apache Ranger

Apache Ranger is a framework that enables, monitors, and manages comprehensive data security across the Hadoop ecosystem.

Table Access Control List

To address the governance limitations of HMS, Databricks introduced TACLs to provide fine-grained access control and governance for HMS tables within a Spark cluster. TACLs came in two variants: one that supported only SQL-based access and another that supported a broader range of access methods, including SQL, Python, and PySpark. This implementation utilized a metadata table to store predefined user permissions, which were subsequently validated against all incoming data requests, regardless of whether they originated from SQL queries or DataFrame API calls. The permissions were enforced at runtime, whether the user accessed the data using SQL or via the DataFrame API in Spark, providing a robust layer of access control and governance. Figure 2-5 illustrates the implementation of TACL in Databricks.

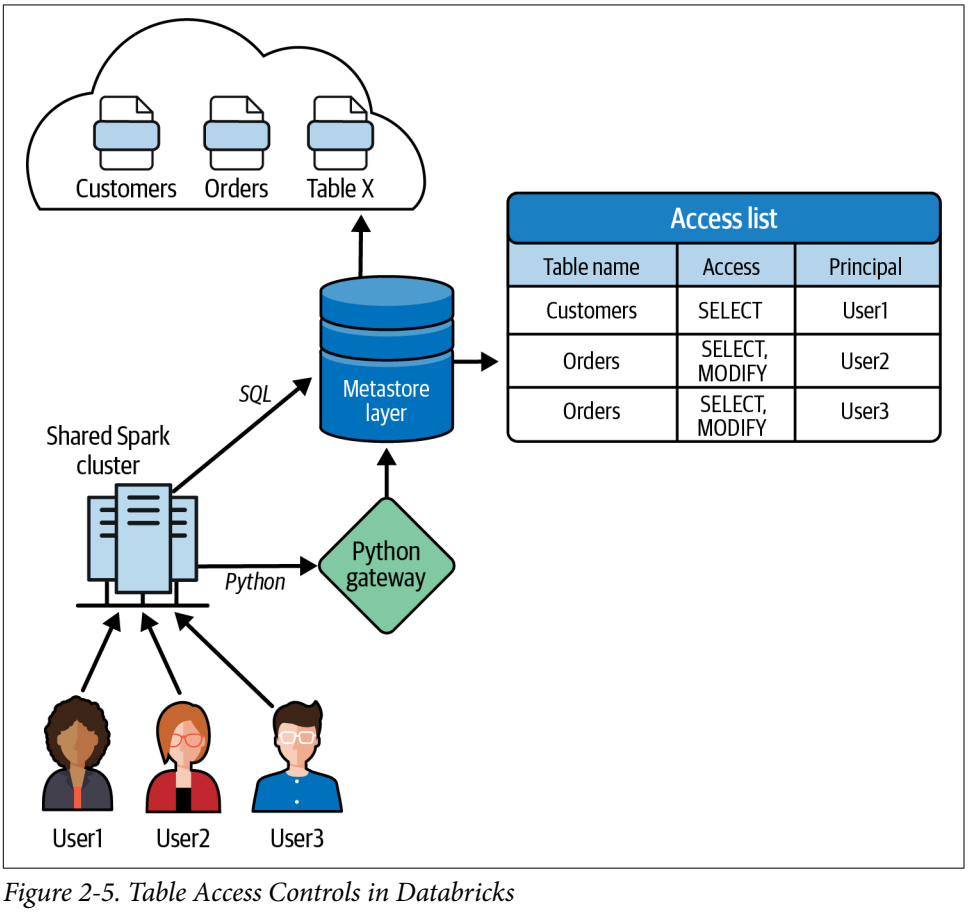


Figure 2-5. Table Access Controls in Databricks

Nexa leveraged TACL for its governance in HMS using Python and SQL table access control. By enabling TACL on their workspace, Nexa could restrict users from directly querying tables using Apache Spark DataFrames. Applications that cater to multiple domains require data to be filtered at the domain level while consuming from the data lake. PII that gets captured requires masking and is visible only to users with elevated privileges. Before TACL, developers replicated data to tables, one with PII and another without PII. The administrator then granted table access via group membership, and only the privileged group users could access the data.

Dynamic views in TACL solved the data replication problem by enabling row-level filter and column masking functionality. TACL provides two functions that help implement fine-grained access controls on data in HMS: `current_user()` and `is_member()`. The function `current_user()` returns the logged-in user to the Databricks workspace. The `is_member()` function validates whether the current user has the appropriate membership in the group. The following code validates whether the current user is a valid user of the demand planning team with Nexa:

```
SELECT
 current_user as user,
 is_member("demand_planning") as admin
```

TACL addressed four key challenges in providing robust access control and governance in Apache Spark:

- TACL enforces SQL-based Access Control Lists (ACLs) to the Spark DataFrame API by checking the metadata table for the correct grants for each user. TACL enforcement ensures that users can access only data to which an administrator has granted permissions.
- TACL blocks access to unsupported Spark APIs, such as Resilient Distributed Dataset (RDD), which could read underlying files directly by enforcing an allow listing mechanism at the gateway that Python uses to communicate with the Spark driver. TACL checks each Spark API invocation against an allowed list of supported APIs and only allows those that verify permissions before accessing data.

Definition: Resilient Distributed Dataset

Since its inception, Spark's primary user-facing API has been the RDD. RDD provides an immutable distributed collection of your data elements that Spark partitions across nodes in your cluster. Spark's RDD API operates on this collection in parallel, offering low-level transformations and actions that users can apply to their data.

- TACL's approach to addressing the issue of network access involves two key measures. First, clusters with Table Access Control are prohibited from connecting to cloud provider metadata services that contain cloud provider credentials, preventing users from using Python to circumvent ACLs. Second, incoming and outgoing connections are restricted to ports 80 and 443, preventing users from connecting to other potentially sensitive cluster services.
- Lastly, TACL solved the problem of processes accessing intermediate paths where Spark writes temporary data during data processing. Spark's tendency to spill data to disk during intermediate operations like shuffles and joins posed a risk of data leakage to other users. Furthermore, users could write data directly to local files, allowing unauthorized access. To mitigate this, TACL requires each Databricks user to execute their code as a separate low-privilege OS-level user on the worker machines. These users are forbidden from reading any files written by Spark or others, ensuring that sensitive data remains secure.

AWS Instance Metadata Service version 1 (IMDSv1) is a good example of why connecting to cloud providers metadata services are restricted in TACL and was spotted by Colin Percival, a Canadian computer scientist and computer security researcher, in his blog titled "EC2's Most Dangerous Feature":

"IAM Role credentials are exposed to EC2 instances via the EC2 instance metadata system: In other words, they're available from *http://169.254.169.254/*. (I presume that the 'EC2ws' HTTP server which responds is running in another Xen domain on the same physical hardware, but that implementation detail is unimportant.) This makes the credentials easy for programs to obtain...unfortunately, too easy for programs to obtain."

AWS later fixed this by releasing the more secure version, IMDSv2.

Databricks Table Access Control empowered administrators to share a Spark cluster with users who leverage Python and Spark DataFrame APIs while controlling data access at varying levels. SQL-based access was possible only in a shared cluster before TACLs, which limited compute resource sharing. With TACL enabled, users could leverage the full power of Spark and Python with support for fine-grained access controls on their data.

Credential passthrough

Considering many user's workflows, mainly those running machine learning workloads, who often work directly with files rather than tables, it's reasonable to ask: why not rely on the cloud provider's file system ACLs to manage data access and governance? Typically, cloud engineers within these teams would have already established

file-based access controls on their storage layer, so it was simply a matter of extending these controls to allow access through Databricks Spark clusters. Cluster-level governance has limitations, and delegating access control to the cloud provider's file system ACLs is often more effective, mainly when working with files rather than tabular data. Cloud providers already provide robust and mature IAM capabilities leveraging IAM roles, IAM policies, and IAM users.

In Databricks on AWS, administrators register AWS instance profiles within the workspace and associate them with a Spark cluster. This instance profile enables the cluster to assume specific roles and generate the credentials for accessing cloud storage. When relying on an instance profile attached to a Spark cluster, all users inherit the same level of access, as defined by the instance profile, without the ability to enforce user-specific access controls. When users use an instance profile to access file systems, administrators cannot track which specific user accessed what resources, as native logging tools like CloudTrail do not provide a clear audit trail. The lack of precise auditing and tracking is where credential passthrough comes into play, enabling automatic authentication to S3 buckets from Databricks clusters using the same identity used to log into Databricks.

AWS IAM credential passthrough offers two key advantages:

- It enables multiple users with varying data access policies to share a single Databricks cluster. Credential passthrough is in contrast to using an instance profile, which is limited to a single IAM role and forces all cluster users to share the same role and associated data access policies.
- IAM credential passthrough assigns a unique identity to each user, enabling more granular S3 object logging via CloudTrail, where the system directly attributes all access to resources to the user's Amazon Resource Name (ARN) in the logs.

Figure 2-6 shows that an administrator can enable credential passthrough during Spark cluster creation in a Databricks workspace by enabling a checkbox. The Spark cluster handles all of the complexities internally.

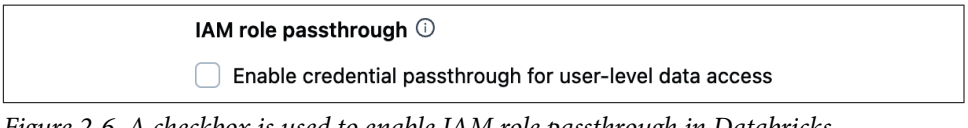


Figure 2-6. A checkbox is used to enable IAM role passthrough in Databricks cluster creation

While Databricks offers various flavors of credential passthrough, they all share a common foundation despite differences in implementation across cloud providers and variants. One implementation in AWS uses the Security Assertion Markup Language 2.0 (SAML 2.0) federation-based passthrough. When you log into your Databricks workspace through an identity provider, an AWS Security Token Service

(STS) call is initiated, and a credential for you against the IAM role assigned to you is issued. When you attempt to read files by attaching a notebook to a Spark cluster, the notebook uses the generated credential to authenticate your access to files in cloud storage. Upon successful authentication, you are granted access to files in cloud storage for which you have read or write permissions, as specified by the cloud storage's access control policies. Figure 2-7 illustrates the workflow of credential passthrough in Databricks on AWS workspace.

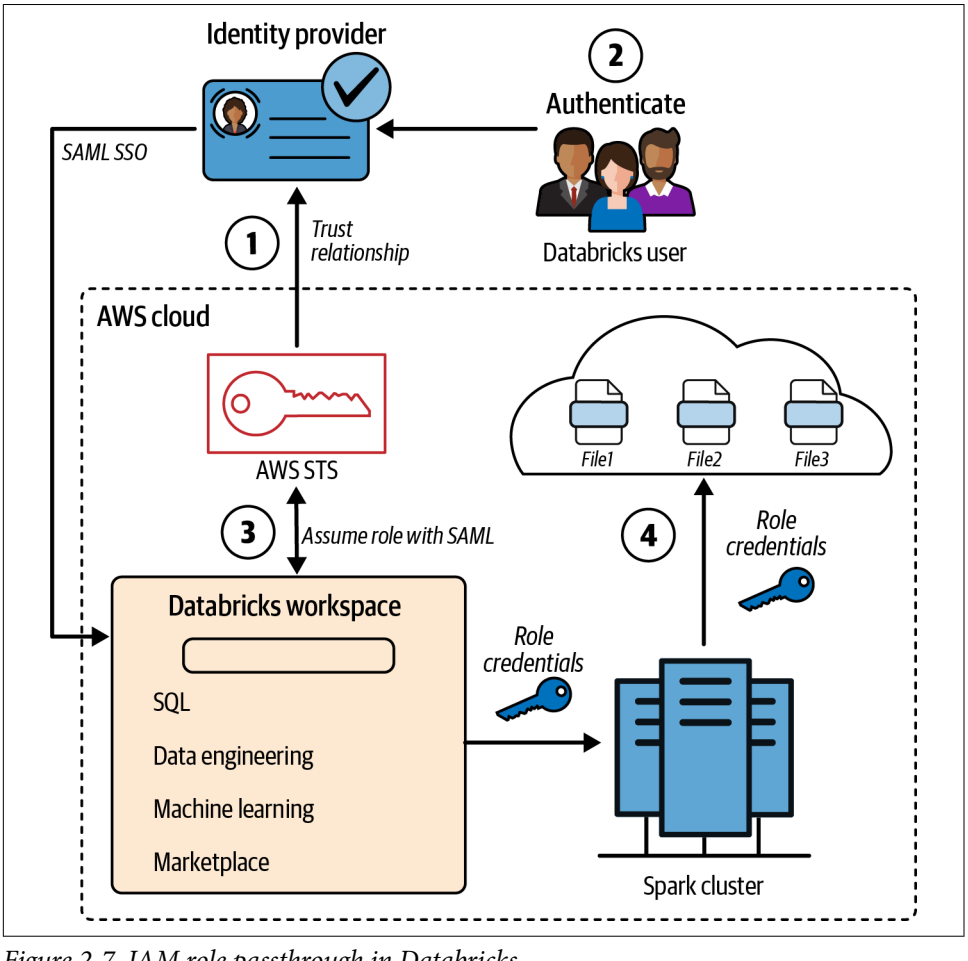


Figure 2-7. IAM role passthrough in Databricks

Passthrough can be enabled in an AWS Databricks workspace through four stages, outlined as follows:

1. A cloud administrator configures the trust relationship between your identity provider and AWS accounts for the identity provider to control which roles users can assume.
2. When you log in to Databricks via SAML single sign-on (SSO), the identity provider passes your role entitlements to Databricks.
3. Databricks invokes the AWS STS, passes the SAML response, and obtains temporary security tokens to assume the designated roles on behalf of the user.
4. The Databricks runtime leverages the temporary security tokens to authenticate and authorize user requests to S3, providing seamless and secure access to S3 resources from the Databricks cluster.

Unity Catalog Architecture

So far, we have discussed some of the data governance challenges HMS faces, including the limitations of enforcing access controls at the cluster level. When an access request is made without proper context, administrators struggle to determine the correct level of access for the user. Moreover, without custom-built solutions, tracking who has access to what data and for what purpose is complex. Unfortunately, custom-built fine-grained access control solutions can sometimes lead to toxic combinations, resulting in unauthorized access. Additionally, the limitations of HMS include its inability to share metadata across workspaces, its lack of support for ML and AI models, and file-based access controls. These limitations make HMS unsuitable for building the next generation of data governance in Databricks. Unity Catalog is designed to address the limitations of HMS. We will discuss in detail the architectural changes that made it possible.

While evaluating Unity Catalog as the governance solution at Nexa, the data architects had two crucial questions regarding Unity Catalog architecture and data management capabilities:

- The team sought clarification on the hosting location of Databricks services for Unity Catalog in the cloud provider and whether it functions as a control or data plane service. As outlined in Chapter 1, the Databricks Platform consists of the control plane (Databricks cloud account) and data plane (cloud account for Nexa Boutique). When Unity Catalog functions as a control plane service, it is crucial to identify the metadata or information about the data it stores in the control plane.

- The team wanted to know how Unity Catalog supports adhering to regulatory requirements around PII data storage, data transfer, and data sovereignty. The Nexa Boutique architects had to understand the critical architecture components underpinning Unity Catalog to make an informed decision.

This section briefly goes over the architecture of Unity Catalog, starting with the Databricks account, which helps centralize user and workspace management. Then, we will cover the regional metastore concept where all the metadata lives in Unity Catalog. We will address the question of what metadata the control plane stores, a topic that interests the architects at Nexa. As you progress through the book, you'll gain insight into how Unity Catalog handles sensitive data, such as PII, and how it helps you comply with regulatory requirements.

Centralized Governance with Unity Catalog

Table Access Controls solved significant challenges with the HMS in Databricks, but how do we scale this for the new age? The creators of HMS originally built it to address a specific use case: enabling users to access files stored in a data lake as if they were tables in a data warehouse. HMS excelled in its objective but fell short in data governance because it was not a primary design consideration. To address the limitations of HMS, developers created custom solutions and incorporated them into the system's architecture. For instance, engineers at eBay developed a Hive metastore event listener that transmits table change events to Apache Atlas to build their data governance for Hadoop clusters integrated with Apache Ranger.

Definition: Apache Atlas

Apache Atlas is a scalable governance platform that helps enterprises meet compliance requirements within Hadoop and across their data ecosystem. It provides open metadata management, data asset cataloging, classification, and collaboration tools for data governance and management.

Building custom solutions has limitations and can introduce security vulnerabilities at the enterprise scale. The rising challenges of keeping up with regulatory requirements, advancements in AI, and the need for a centralized data governance solution necessitated a more modern solution. Unity Catalog was designed from the ground up to support the contemporary Data Intelligence Platform architecture.

We've established that the Unity Catalog service resides in the control plane. But what exactly does it store in a metastore? To answer this, let's dive into the Unity Catalog architecture and explore the implications of its control plane placement. Figure 2-8 illustrates the overall Databricks architecture and the deployment of the control and data planes.

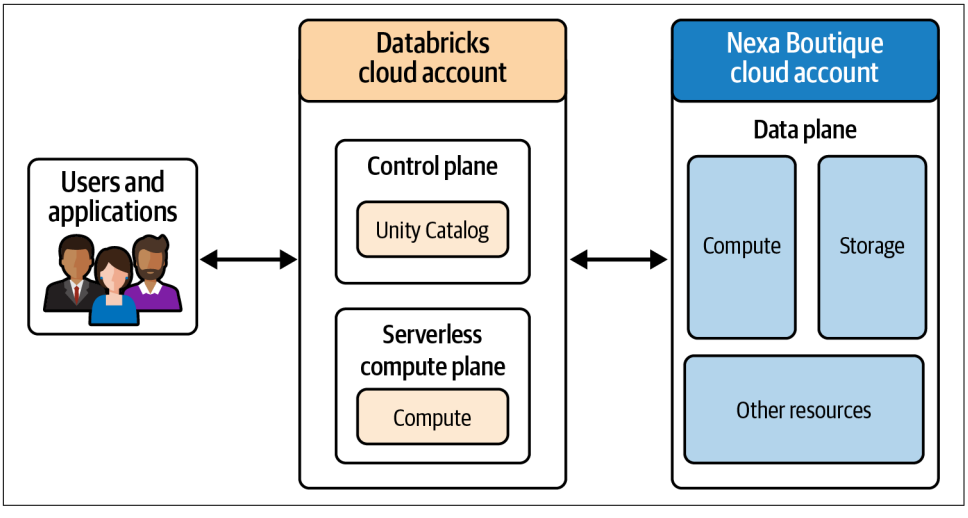


Figure 2-8. Overall Databricks architecture with Unity Catalog

Databricks account

In the HMS world, Databricks workspaces represent the boundary for access controls and metadata isolation. You could have multiple Databricks workspaces, each managed individually by your workspace admin. Defining access controls and user management per workspace demands significant effort from administrators, particularly when they onboard new teams or deploy new workspaces to the platform.

Databricks introduced an *account* concept in response to the need for centralized management, allowing administrators to manage multiple Databricks workspaces from a single, central location. The Databricks account enables centralized management of all workspaces created for an enterprise, streamlining the management process. The Databricks account has an account console that helps an *account admin* create and manage users, workspaces, and metastores centrally.

Definition: Account Admin

An account admin is a new role introduced with the Databricks account. When a user onboards as an account admin, a highly privileged role within the platform, they can log in to the account console and manage the entire Databricks deployment centrally.

Unity Catalog architecture improves upon HMS by centralizing user and metadata management, elevating it from a closed workspace to a more open and shared Unity Catalog account level. User management at a centralized account level simplifies administration for workspace administrators by enabling identity federation. Administrators can synchronize your users, groups, and service principals into the Databricks account console. For example, you can leverage your organization's identity provider's SCIM API to sync identities to Databricks from your chosen identity provider. Administrators can then assign these provisioned identities to any Databricks workspace using identity federation, eliminating the need to repeat this process across multiple workspaces. Before Unity Catalog, administrators had to configure SCIM integration individually for each workspace they deployed. Chapter 3, Identity Management, discusses how an administrator can manage the identities in Databricks and the best practices followed by Nexa.

Managing over 500 workspaces and their user management was a significant challenge at Nexa. However, identity federation streamlined the SCIM integration process, allowing administrators to configure it only once and manage user access centrally from the account console for all workspaces deployed within Nexa. Figure 2-9 shows the difference between HMS and Unity Catalog in managing users and metadata.

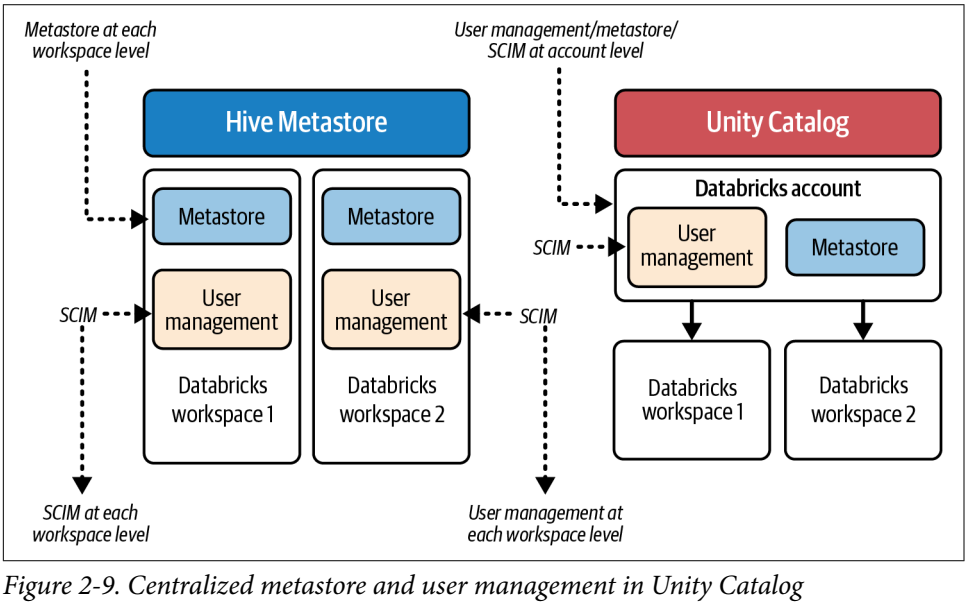


Figure 2-9. Centralized metastore and user management in Unity Catalog

The metastore regional construct

As mentioned, Unity Catalog is a centralized metadata management service that operates at the Databricks account level. Unity Catalog leverages the underlying cloud provider's regional services to store its metadata. At the heart of Unity Catalog is the metastore, which serves as the highest level of abstraction for data organization within the account. Under the hood, the metastore is part of a managed database instance provisioned and maintained by Databricks. The metastore holds metadata for the data assets managed by Unity Catalog, providing a centralized repository for data assets. The metadata includes table definitions, schemas, cloud storage locations for the files, access control definitions, audit logs in system tables, and much more.

Databricks mandates creating only *one metastore per cloud region* for ease of data sharing requirements. HMS limited access to a single workspace, and Unity Catalog had to solve the limitation to provide users with a proper data democratization experience. You can attach your Databricks workspaces to the metastore in the cloud region where your workspaces exist. Deploying all your services in the same cloud region streamlines networking and reduces network latencies and costs, as same-region data transfer is free or incurs lower expenses than cross-region transfers. Deploying services in the same cloud region also enables compliance with local data residency regulations, ensuring that data remains within the same geographic boundaries. A metastore can optionally have a cloud storage location, which stores data for managed tables, volumes, or ML models created in Unity Catalog. Although storage provisioning is available at the metastore level, it is not recommended since data isolation can be achieved through catalog-level storage. An account admin can assign a `metastore admin` role to a metastore, who can then handle all metastore management activities, including access controls and its child object creation.

The flexibility to attach a single metastore to multiple workspaces in the region opens up data-sharing capabilities across workspaces, which closes a significant gap in HMS: sharing metadata. Metadata in a Unity Catalog metastore and its access controls can be defined once, consumed, and enforced from any workspace they are attached to. Figure 2-10 illustrates a shared metastore across multiple Databricks workspaces.

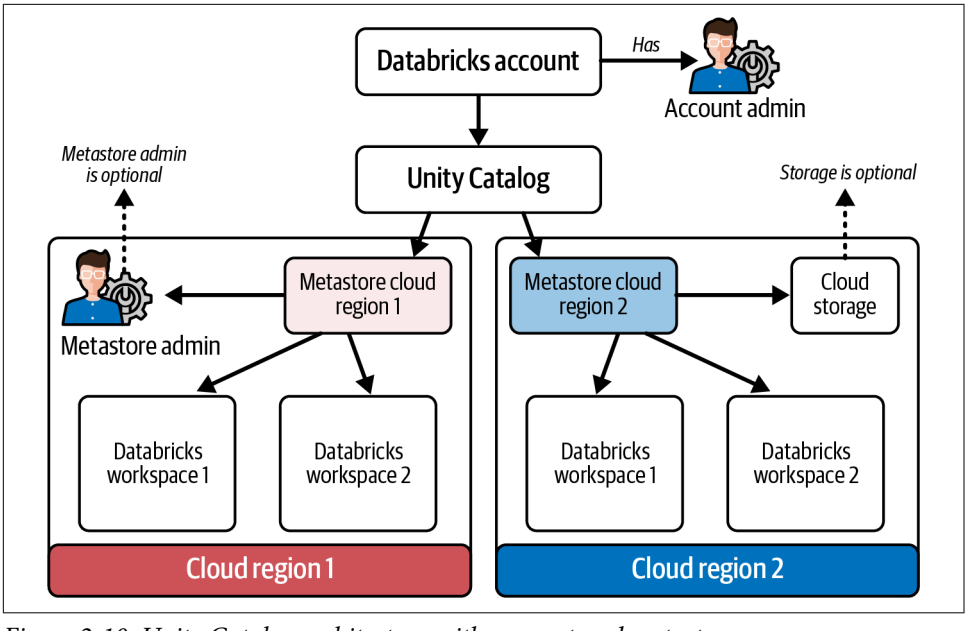


Figure 2-10. Unity Catalog architecture with account and metastore

Definition: Metastore Admin

Metastore admin is another highly privileged role in Databricks, managing all the access controls for a metastore. Due to the role's ability to grant itself access to all data assets within the metastore or create data shares, Databricks recommends that the role be kept optional and privileges delegated to a Databricks workspace admin.

So what details does the Unity Catalog metastore capture and store in its persistent storage layer in the Databricks control plane? The metadata captured and stored in the Unity Catalog metastore includes information on the data assets, such as the following:

- Names and schemas of tables and views and their corresponding file locations in cloud storage accounts.
- Names and file locations of volumes.
- Names and artifact location of ML models.
- Function definitions managed by Unity Catalog.
- Access controls for these data assets, including grants and privileges assigned to users of the Databricks workspace or service principals and IAM roles.
- Cloud storage locations and its credentials: Unity Catalog vends downscoped, short-lived credentials to consumers on demand using cloud storage locations and their credentials.
- Recipient and share details when sharing data, including the assets shared.
- Audit logs and usage that get captured in system tables.

It is essential to understand that the metastore is an extension to your data assets metadata rather than a storage location for the data itself. Your data remains stored within your cloud provider account, or what we refer to as the *data plane*, and Databricks does not transfer it to the Unity Catalog metastore living in the Databricks control plane.

A central metastore enables the ingestion of data sources to the lakehouse once, enabling consumption from any Databricks workspace from the same region as the metastore.

Figure 2-11 depicts how the architectural changes in Unity Catalog simplified the Nexa lakehouse deployment compared to HMS-based deployment.

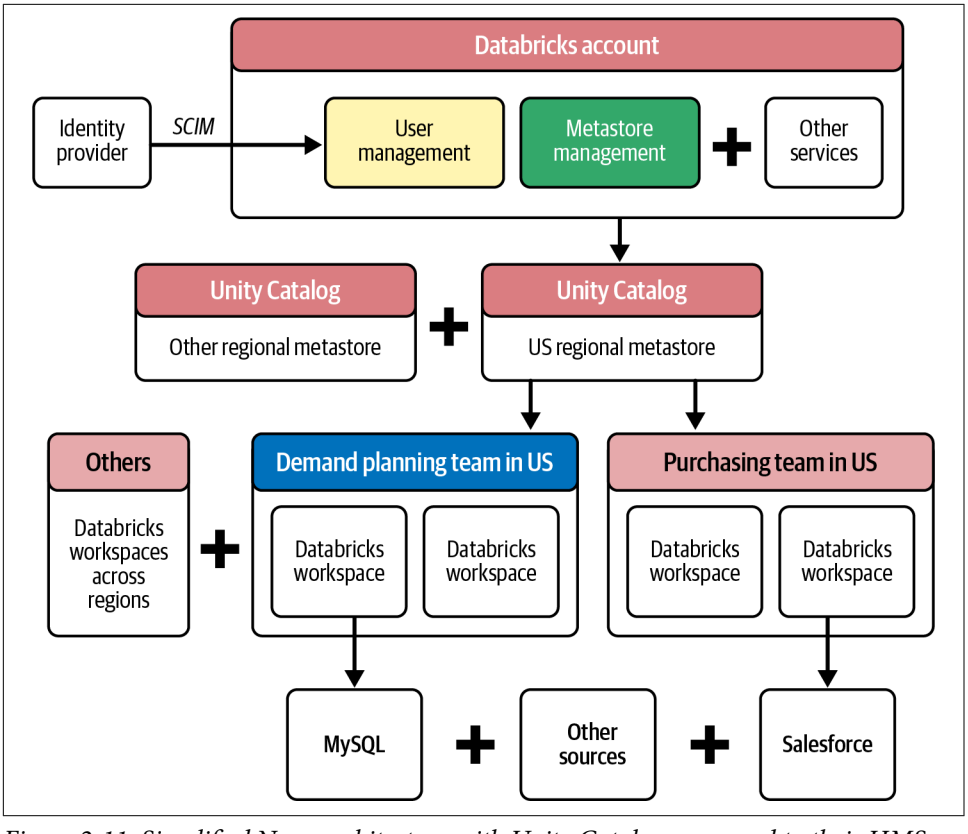


Figure 2-11. Simplified Nexa architecture with Unity Catalog compared to their HMS-based architecture in Figure 2-3

The Governance Model of Unity Catalog

So far, we have interpreted two fundamental features that define the architecture of Unity Catalog: a centralized account and the concept of a region-bound metastore. Now, we examine how Unity Catalog implements governance, beginning with its credential management process. We will guide you on how Unity Catalog centrally manages credentials and secures access by granting access to isolated individual users and processes rather than attaching to a cluster where everyone has equal access rights. Once you securely register a credential in Unity Catalog, we will look at how to use a storage credential to create external locations that map to cloud storage locations. We end this section by discussing the different compute modes in Unity Catalog and how processes and users get isolated in a Spark cluster, forming the critical architectural design enabled by Unity Catalog Lakeguard (discussed later in this chapter).

Decoupled Storage Credentials

When building heavily complex cloud-based solutions, where hundreds of services interact and interdepend, a single vulnerability can bring the entire system crashing down. Often, that vulnerability is not a sophisticated hack or a coding error but something much more mundane: credential management. In a Databricks workspace enabled with Unity Catalog, the metastore is crucial to a centralized credential management system. It securely stores and manages all credentials necessary for connecting to cloud object storage locations and external services, providing a single point of control and administration. The storage credentials feature enables the Unity Catalog metastore's centralized credential management capabilities, which provide a unified and secure way to store, manage, and control access to sensitive credentials. The type of storage credential used depends on the cloud platform hosting the Databricks workspace. Specifically, the storage credentials feature supports registering various cloud-specific identity mechanisms, as shown in Table 2-1.

Table 2-1. Comparison of cloud-specific mechanisms

AWS	GCP	Azure
IAM roles	Service accounts	Managed identities
Service principals		Service principals

In contrast to the HMS approach, where a credential attached to a cluster is accessible to all cluster users, storage credentials in Unity Catalog-enabled Databricks workspaces are governed assets that an administrator can grant access to specific users or groups on a need-basis. By decoupling credential management from cluster configuration, Unity Catalog provides a more secure and scalable approach to managing access to cloud storage and external services.

In Europe and the UK, Nexa hosted its Databricks workspaces for the LSC domain in its Microsoft Azure cloud account. ADLS became the cloud storage layer for its Databricks Data Intelligence Platform architecture. A managed identity in Azure is the credential to connect to the ADLS layer from a Databricks compute resource. Microsoft Azure provides a first-party resource called an `access connector` for Azure Databricks that can be used to connect managed identities to a Databricks account. A Unity Catalog metastore admin or a delegated Databricks workspace administrator registers this managed identity as a Unity Catalog storage credential rather than attaching it directly to a cluster, as with service principals in HMS. Once registered, administrators can grant users privileges to use the credentials to connect to an ADLS location. Databricks audit logs attribute actions to the initiating user, providing a clear audit trail that reflects the user's activities rather than the managed identity used to execute the process. At Nexa, the CDP team took ownership of the registered storage credentials via a Microsoft Entra ID group. Ownership with a

group is the recommended way of managing ownership of assets instead of assigning it to individual users.

Definition: Managed Identity

Managed identities in Microsoft Entra ID provide applications with an automatically managed identity for authenticating with resources. This eliminates the need for credential management and is recommended for use with Azure Databricks and Unity Catalog via an access connector.

External Location for Cloud Object Storage

Unity Catalog encapsulates cloud storage locations in its metastore as external locations. An administrator can create an external location using a previously registered storage credential and must grant the `CREATE EXTERNAL LOCATION` privilege on the registered storage credential for any users to create an external location in Unity Catalog. When creating an external location in the Azure cloud, as a cloud administrator, you must assign the Storage Blob Data Contributor role to the managed identity to grant access to the storage account. Unity Catalog generates a downscoped credential token per user based on their grants in Unity Catalog for specific datasets residing in the external location, thereby limiting access. Unity Catalog requires administrators to register an external location before registering tables against it. An external location is also a governed asset in Unity Catalog, so an administrator can prevent unauthorized access to these cloud-based object storage locations.

Unity Catalog prevents overlapping any storage location it manages, so any external location registered in Unity Catalog cannot overlap another external location within the metastore. Databricks recommends never creating an external table at the root of an external location. Instead, create external tables in subdirectories within an external location.

Before Unity Catalog, Databricks File System (DBFS), a distributed file system that interacted with cloud-based storage, was used extensively to create tables in HMS. Nexa heavily utilized DBFS because it allowed them to easily mount an ADLS container to a workspace using a service principal, granting access across the workspace. DBFS is helpful for use cases where everyone shares the same level of access to data in the platform. For use cases where the access level varies per user, dynamic views were used for row-level filtering and column masking. Unity Catalog deprecated DBFS and doesn't support creating tables against a DBFS object. While migrating from HMS to Unity Catalog, the CDP team at Nexa moved all of their DBFS storage locations to external locations in Unity Catalog, improving their storage layer governance manifold.

That solves the governance problems for table-based access, but how does Unity Catalog work with arbitrary files in DBFS? Uploading to DBFS is an easy way to process unstructured files, especially when running ML workloads like image processing. However, the lack of access controls on files landing in DBFS storage required a fix. `Volumes` replace arbitrary files in DBFS with a more secure, governed file store in Unity Catalog, thus closing the gap in governing files within a Databricks workspace. Figure 2-12 shows how Unity Catalog stores metadata, manages storage credentials, and defines external locations in a simple format.

Definition: Volumes

Volumes in Unity Catalog represent a logical storage space in cloud object storage, enabling access, storage, governance, and organization of files in any format, including structured, semi-structured, and unstructured data.

Volumes serve the purpose of storing arbitrary files in the cloud governed by Unity Catalog. Creating tables directly on top of a Unity Catalog Volume is not permitted. Instead, you can read data from a Volume but must write the resulting table to a separate external location.

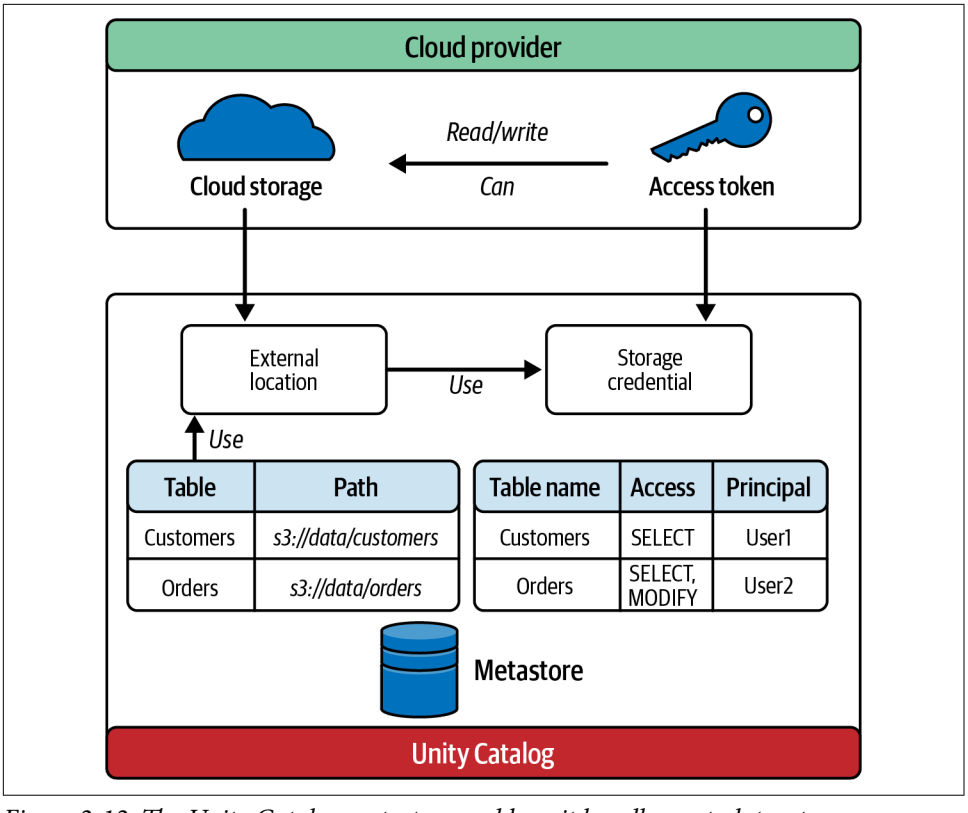


Figure 2-12. The Unity Catalog metastore and how it handles metadata, storage, and credentials

Users require the `READ FILES` permission to access data from an external location registered in Unity Catalog. With this permission, users can directly read data from the storage using path-based access. For example, you can use the following Spark code to read from an AWS S3 bucket (*s3://your-bucket-name/path/to/your/data/*) registered as an external location in Unity Catalog:

```
spark.read.format("parquet")
   .load("s3://your-bucket-name/path/to/your/data/")
```

However, an additional `SELECT` permission is required if you register a table against this external location. Granting only the `READ FILE` privilege is insufficient once a table is registered. By design, table-level privileges precede file-level permissions, overriding the `READ FILE` privilege. This means that path-based access will no longer work without `SELECT` access on the table. This behavior applies uniformly across AWS S3, GCS, and ADLS.

Compute Modes in Unity Catalog

Building on our previous discussion of centralized credential management, we will examine how Unity Catalog enforces fine-grained access control, ensuring that only authorized users can access the data. How does Unity Catalog prevent unauthorized access in Apache Spark compute, where shared credentials were once vulnerable? To answer this, we need to understand the different compute modes in Unity Catalog and how Lakeguard enables a secure, shared Apache Spark environment.

Definition: Lakeguard

Lakeguard is the technology behind the Unity Catalog compute engine, which enables isolated data processing. It enforces data governance at the compute level, isolating user code and Spark engine access. This allows secure sharing of compute resources for SQL, Python, and Scala workloads, with fine-grained access control and RLS.

A storage credential in Unity Catalog grants blanket access to the cloud storage location it is associated with. Granting access to the entire storage location's parent path can sabotage access controls because a table will probably use a child path only inside the parent storage location.

Consider a scenario with Nexa in Databricks on AWS, where an S3 storage bucket lands data from its Salesforce application. Data analysts at Nexa process the data to generate a BI dashboard in Power BI related to product shipment, which they share with their business executives. The CDP team registered an external location, *s3://nb_lsc_purchasing_prod/salesforce_gold*, and a corresponding `IAM ROLE` with `READ` and `WRITE` bucket policies as a storage credential in Unity Catalog. Table 2-2 lists the Unity Catalog tables created against the path where the data is landing.

Table 2-2. Cloud storage paths and the corresponding tables created in Unity Catalog

Storage path	Table
s3://nb_lsc_purchasing_prod/salesforce_gold/shipping	shipping
s3://nb_lsc_purchasing_prod/salesforce_gold/drivers	drivers
s3://nb_lsc_purchasing_prod/salesforce_gold/vendors	vendors

The administrator grants the analyst `SELECT` access to the `shipping` table, allowing them to query the data for the BI dashboard. The analyst fires a SQL query from the Power BI reporting application to request data from the shipping table. However, if Unity Catalog used the registered storage credential, it could inadvertently grant the user access to all datasets within the location, not just the shipping table.

To prevent unauthorized access, Unity Catalog first checks the requested user's grants and denies access if they don't have the necessary permissions. When a legitimate user requests an asset, Unity Catalog acts as a credential vendor and generates a downscoped access token specific to the requested asset, blocking access to other datasets. By generating a downscoped access token that can read files only from the *s3://nb_lsc_purchasing_prod/salesforce_gold/shipping* S3 storage path, Unity Catalog prevents the analyst from accidentally accessing any other tables they are not granted access to.

The complete lifecycle of data access, as managed by Unity Catalog, is illustrated in Figure 2-13.

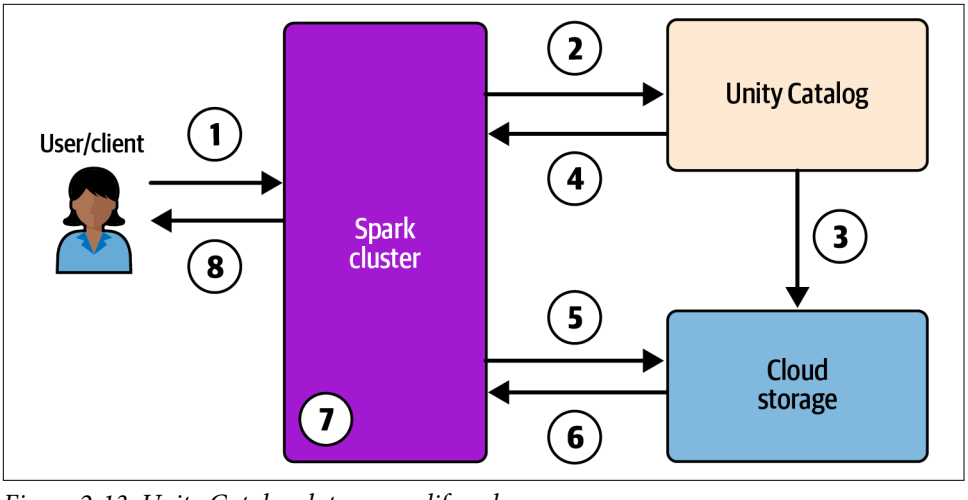


Figure 2-13. Unity Catalog data access lifecycle

Here is what happens under the hood when you run a query through a Unity Catalog compute cluster or a warehouse:

1. The end user executes a query against a Unity Catalog compute cluster or a warehouse.
2. Unity Catalog then checks the grants for the user against the requested table to ensure they have the necessary permissions.
3. If authorized, Unity Catalog assumes the role registered against the storage credential and generates a downscoped temporary credential, a time-limited access token.
4. Unity Catalog returns the downscoped temporary credential and the path for the files back to the cluster.
5. The cluster then requests data from cloud storage using the temporary token.
6. The cloud storage provides the requested data back to the cluster.

7. Once the data is received, the cluster enforces policies such as RLS and column masking to ensure that the end user accesses only authorized data.
8. Finally, the end user gains access to the data.

Unity Catalog provides users with two compute access modes: `standard` (previously known as `shared`) and `dedicated` (previously known as `single-user`). A standard access mode allows multiple users to share a Spark cluster and concurrently run jobs, providing total isolation between them. On the other hand, dedicated access mode mandates that administrators assign clusters to a specific user and can only execute jobs submitted by the designated user.

Isolation using Unity Catalog Lakeguard in standard or shared access mode

Unity Catalog acts as a credential vendor in the interface layer with users and applications, granting access to only the required dataset requested by the user, thereby eliminating the risk of unauthorized access to data.

When using an Apache Spark compute for data processing, the cluster is the boundary for data isolation. In a shared Apache Spark cluster, users and their processes utilize the same computing resources, the Java Virtual Machine (JVM), and temporary paths where Spark stores intermediate results. Spark clusters run processes as an elevated user. Processes that run within a shared cluster are not isolated from each other. Lakeguard resolves the isolation issue in Unity Catalog-enabled Apache Spark clusters by implementing robust user and process isolation, providing a secure foundation for multitenant deployments.

Lakeguard enables data governance at the compute level by isolating users, their code, and user-defined functions (UDF) from each other and the Spark engine. Unity Catalog Lakeguard achieves isolation by making multiple changes to the Spark cluster architecture. The changes include implementing user isolation using Spark Connect and enforcing process and code isolation through state-of-the-art sandboxing techniques based on containers. Figure 2-14 depicts how Spark Connect enables a client-server architecture in Spark, thereby isolating users and processes.

Definition: Spark Connect

Apache Spark 3.4's Spark Connect introduces a client-server architecture, enabling remote access to Spark clusters via the DataFrame API. This decoupling allows Spark to be easily integrated into various applications, IDEs, notebooks, and programming languages.

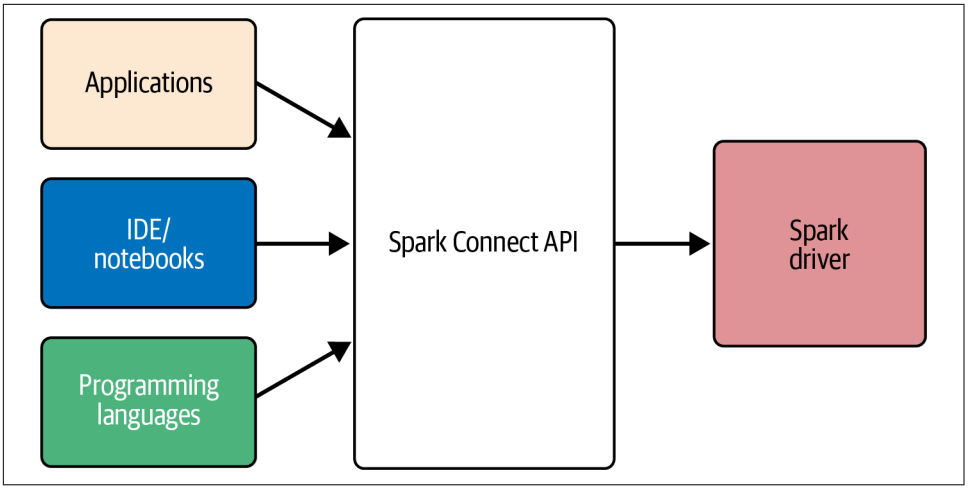


Figure 2-14. Isolating users and applications from the Spark driver using Spark Connect

Spark Connect is implemented based on the Apache Spark DataFrame API and will work only for applications implemented using DataFrame. If you have an application implementing the RDD API, migrating the application to a Unity Catalog-enabled shared Spark cluster will require code changes to implement the DataFrame API.

With its thin client library, Spark Connect embeds itself in application servers, IDEs, notebooks, and programming languages. By encoding client operations like SQL using protocol buffers and transmitting them to a Spark server via the gRPC framework, Spark Connect enables the isolation of clients in an Apache Spark cluster. When a client submits a process to a Spark cluster, the next step is to sandbox the process, including isolating user code from each other, by deploying the operations and corresponding user code to a container. By containerizing user code, Unity Catalog Lakeguard restricts access to the container's boundaries, preventing user processes from accessing other users' processes.

Additionally, containerization replaces the highly privileged user with a downscoped user, limiting access to only the necessary resources and reducing the attack surface. When users execute UDFs, the same methodology is applied, where the UDF is executed in its container, preventing cross-data access. Complete isolation of the user's code and process enables Shared Access mode to also support fine-grained access controls (FGAC). FGAC allows for restricting user access to data at both the row and column levels, enabling granular control over sensitive information. To enable FGAC, Unity Catalog supports row- and column-level filtering by applying functions to tables or columns and the ability to mask PII data.

Unity Catalog's shared access mode restricts users to nonelevated privileges and does not install operating system libraries and compilers, limiting its application to only those that do not require elevated privileges. As a result, shared clusters do not support distributed ML workloads and impose additional limitations due to the lack of elevated access to computing resources. Figure 2-15 illustrates how Spark containerizes the code and process in the Spark driver and the executors.

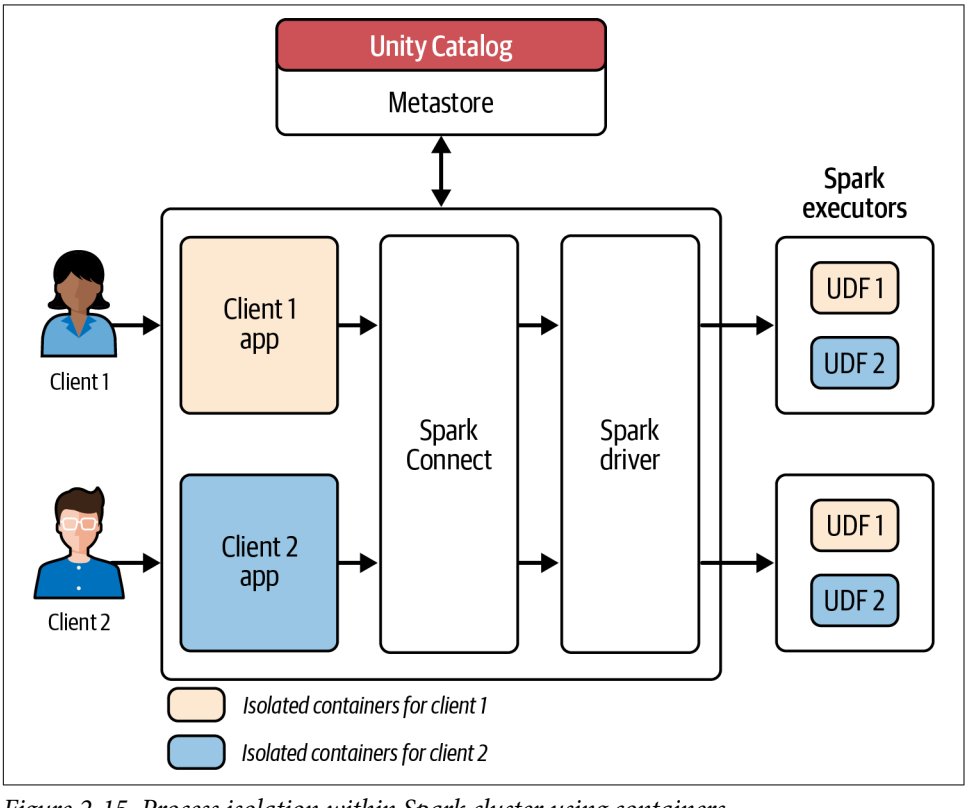


Figure 2-15. Process isolation within Spark cluster using containers

Single user or dedicated access mode for elevated privileges

Unity Catalog's shared access mode is unsuitable for workloads that require elevated privileges, such as distributed ML and AI due to the restricted access to the underlying compute infrastructure. For instance, when running low-level graphics processing unit (GPU) programming on an NVIDIA GPU using CUDA, your application requires direct access to the operating system libraries and the underlying machine APIs to execute efficiently. With its isolated containers and restricted access to the machine API and operating system, the standard access mode architecture imposes limitations that prevent GPU support because it cannot provide direct access to hardware resources required for low-level GPU programming. In contrast, the single-user

access mode provides elevated access to computing resources, empowering Unity Catalog to support a broad spectrum of workloads, including those that demand elevated privileges and GPU. In single-user mode, each user has a dedicated computing environment built on the original Spark architecture's foundation, centrally managed by Unity Catalog.

Elevated access to compute resources introduces new challenges, including the issue of overfetching when querying views or tables with FGACs enabled. At Nexa, a third-party vendor application captures and stores data related to external vendors across domains and regions. However, Unity Catalog's RLS functions are applied when accessing this data, restricting access to the domain teams. Specifically, RLS ensures that each domain team can access only data that belongs to their own domain and is from the same region where they operate. When a user queries a table with RLS applied, there is a possibility of overfetching data from the cloud storage. Overfetching of data occurs when Spark processes data from cloud storage locations, such as files in Delta, Iceberg, or Parquet formats. Because cloud storage requires Spark to read portions of files into the cluster before applying filters, it can lead to unnecessary data transfer and processing, even when a user has permission to access only a subset of the data within a table.

The nature of file storage makes it impossible to read only a selected number of rows or columns from a file block that contains multiple records. This is because files are written in contiguous blocks, making accessing specific columns or rows intricate without reading the entire block. When you execute a SQL query with filter criteria on a Delta file stored in S3 using Spark, the following steps occur:

1. Spark reads the file's metadata and index to identify which blocks contain the required data, allowing it to prune unnecessary data and optimize the query.
2. Spark reads the required blocks of the file in parallel using multiple tasks, leveraging the engine's distributed computing capabilities.
3. Within each block, Spark filters out the unwanted data based on the query's filter criteria, reducing the amount of data that needs to be processed.
4. The results from each task are combined to produce the final result.

This results in Spark transferring unnecessary data into the cluster, including information the user is unauthorized to access, leading to overfetching. As the process runs with elevated user access to the underlying compute, it may expose the overfetched data to the user. To mitigate this issue, Unity Catalog requires users to have access to the underlying tables when querying views and does not support tables with FGAC enabled.

Unity Catalog with Lakeguard addresses the issue of overfetched data access by introducing a serverless compute layer in the Databricks control plane. This layer filters

data retrieved from cloud storage, ensuring only authorized data gets transferred to the Spark cluster for processing. Additionally, Unity Catalog Lakeguard provides an extra layer of security by restricting access to the serverless compute that performs data filtering, thereby safeguarding sensitive data from unauthorized access by end users. Figure 2-16 depicts the single-user access mode and how the serverless compute enables FGAC on data.

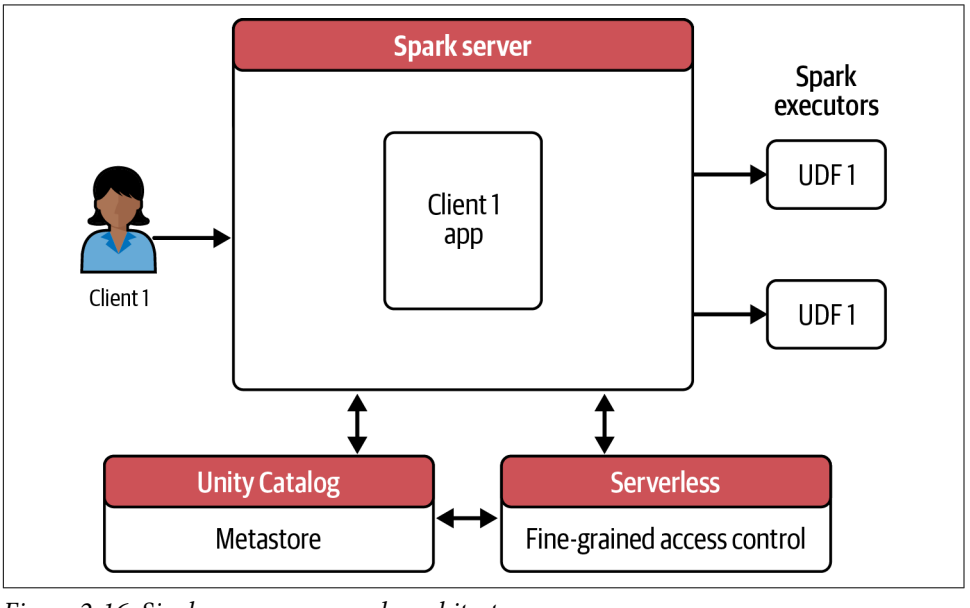


Figure 2-16. Single-user access mode architecture

Chapter 6 discusses in detail the use cases, features, and limitations of different compute options supported by Unity Catalog.

Data Management Features

Good governance relies heavily on effective data management, which depends on carefully cataloging and organizing data assets. In addition to technical considerations, the success of data management is also contingent upon the human element. The people, processes, and practices within an organization can either facilitate or hinder effective data management, making it essential to consider these factors when designing and implementing data management strategies. At Nexa, the data platform team centralized their operations and closely aligned their data management principles with the architectural blueprints and standards defined by the CDP team. Before Unity Catalog, each workspace had its metastore in its dedicated HMS, and the data or metadata associated with each source system lived in their respective Hive schemas. In most scenarios where data isolation was required due to regulatory

requirements or the existence of PII, a dedicated workspace was allocated. Let's now look at how Unity Catalog solves the data isolation needs and provides a robust data management framework.

Catalog as the Namespace

The regional metastore acts as the highest level of abstraction in Unity Catalog, providing a centralized repository for metadata in the control plane. You can store all of your managed data in a centralized cloud storage location associated with the metastore if you choose to do so. However, shared storage layers do not provide the necessary data isolation for domains or applications that must adhere to strict data separation requirements, whether driven by regulatory compliance or internal data-sharing policies. A catalog provides strong isolation in Unity Catalog by introducing a third tier to the namespace hierarchy, building upon the existing schema and table levels. Before implementing Unity Catalog, the Demand Planning team at Nexa stored their Salesforce data in HMS, which they implemented using the `schema.table_name` convention. Specifically, users accessed the team's sales channel data via the namespace `salesforce_gold.sales_channel`.

With the introduction of Unity Catalog, the CDP team established separate catalogs for each of their teams, allowing for greater flexibility and control over their data assets. For example, the Demand Planning team created their catalog named `nb_lsc_demand_planning_prod` to store all their production datasets. The new catalog resulted in a new namespace, `nb_lsc_demand_planning_prod.salesforce_gold.sales_channel`, providing a more granular, team, and environment-specific way of organizing and accessing their data. The CDP team also allocated different catalogs per environment to separate data based on their development, testing, and production environments. All their developers coded and ran unit test cases against the `nb_lsc_demand_planning_dev` catalog while running integration testing against the `nb_lsc_demand_planning_test` catalog.

In Unity Catalog, a *catalog* is a logical grouping mechanism for datasets with common characteristics or purposes. Organizing datasets into catalogs simplifies data management and discovery, making finding and accessing relevant data assets easier. Furthermore, catalogs provide a neat method of encapsulating data products and implementing a data mesh architecture, enabling you to decentralize your data platform and govern data more openly. Organizing metadata against catalogs and segregating it based on environment and business units was a welcoming change for Nexa that democratized its data access.

Data Isolation at Catalog and Schema Level

At Nexa, strict data-sharing restrictions existed between their development and production environments. Domain teams in the same region were concerned about sharing the same metastore and its storage location, as some of them had regulatory requirements that mandated storage isolation from users not entitled to access their data. The catalog provides an additional layer of organization for your metadata, but how does the data get isolated per team or application? Storing all managed assets in cloud storage at the regional Unity Catalog metastore level can lead to data isolation issues, making it difficult to separate data between teams and maintain confidentiality.

The metastore admin, while creating a catalog, can associate the catalog with an external location from a separate cloud account or subscription owned and managed by the domain teams. By bringing your storage to a Unity Catalog catalog, different domain teams can work together from a shared regional UC metastore with complete data isolation at the storage layer. The CDP team ensured that any new catalog creation has a new external location attached and that an external location never gets reused. The automated catalog creation happens via the Databricks Terraform provider, where these rules are baked into their DevOps process to enforce best practices across domains. Having catalog-level storage owned by the domain teams helped alleviate concerns and fast-tracked Unity Catalog adoption at Nexa at an incredible pace.

At Nexa, each application gets mapped to an independent schema in Unity Catalog, and all assets related to the application live in the same schema. Isolating its data per application at the schema level made perfect sense for Nexa. Unity Catalog extends data isolation capabilities from the catalog to the schema level, enabling applications within a catalog segregated in schemas to have their isolated storage layer. A schema can be attached to an external location, providing an isolation path per application—this helped Nexa immensely while building its data governance architecture using Unity Catalog.

Catalog to Workspace Binding

Unity Catalog introduces a significant architecture deviation from HMS by enabling the sharing of regional metastores across multiple Databricks workspaces in the same region. Once a Databricks workspace is attached to a metastore, all its metadata and user-accessible data become available to users within that workspace. The CDP team assigns a unique catalog to each DevOps environment at Nexa: development, testing, and production. It strictly limits access to production data, preventing it from being accessed from lower environments, such as development or testing environments. However, attaching a metastore to a workspace that enables all catalogs within the workspace creates a security concern: users with access to both development

and production catalogs and workspaces can now access production data from the development workspace, compromising data security. Unity Catalog has a catalog-to-Databricks-workspace binding capability to prevent over-provisioning catalogs across environments. Catalog-to-workspace binding enables an administrator to update catalog visibility so that it is visible only to users in workspaces where the binding to workspace is activated. Figure 2-17 depicts how storage isolation can be achieved in Unity Catalog using catalog-level external locations and catalog-to-workspace bindings in AWS.

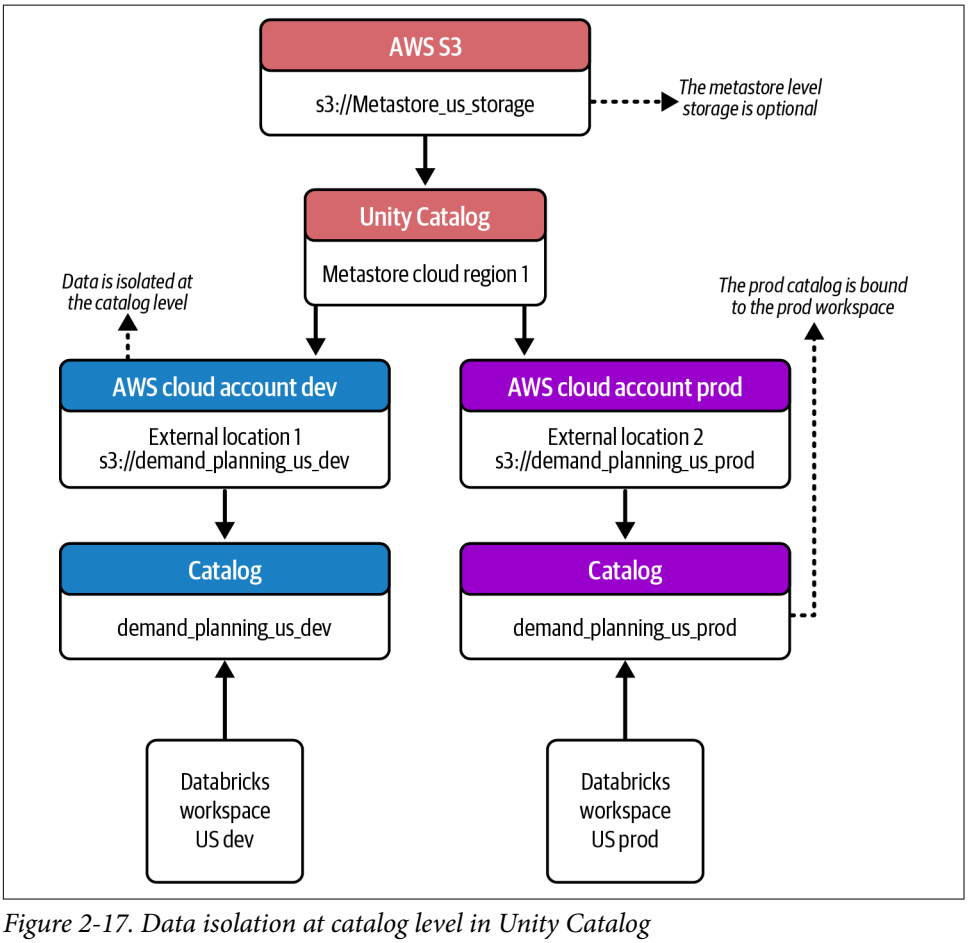


Figure 2-17. Data isolation at catalog level in Unity Catalog

Summary

> *Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time.*
>
> —Thomas Edison

HMS is currently the most interoperable metastore for the big data ecosystem. Still, it is not fit for modern data platforms. After years of tweaks and revisions to address its shortcomings, HMS has reached a point where it is ripe for replacement with a more modern solution. After reading this chapter, at first glance, you might think that Unity Catalog is simply a solution to the governance challenges posed by HMS. However, it unlocks a treasure of new possibilities within the Databricks platform. Moreover, its open source version extends these benefits to the entire community outside of Databricks. While this chapter has laid the groundwork for Unity Catalog by discussing the history and the architecture, it also answers the one question about the metastore, which the data architects at Nexa were looking for answers to.

Now that we have established a solid understanding of Unity Catalog's architecture and data isolation concepts, we are ready to dive into the next critical topic: access controls. Role-based access control is an area of particular interest to the data architects at Nexa, who have been seeking guidance on effectively handling PII data and implementing FGAC in Databricks using Unity Catalog. Chapter 3 explores the answers to these questions and more as we delve into the real-world example of Nexa Boutique's successful adoption of Unity Catalog. Through their story, you'll gain valuable insights into leveraging Unity Catalog's features to drive innovation and implement a range of use cases that meet your organization's unique needs.

CHAPTER 3
Identity Management

Be yourself; everyone else is already taken.
—Oscar Wilde

One of the perks of being in a customer-facing role is that you get to travel and visit different cities. If you're like Karthik, you enjoy traveling and staying at nice hotels. Imagine visiting a new city for a business meeting and staying at a hotel with modern amenities. Think of all the steps involved in booking a room at the hotel and up until you check out.

The first step is booking. You choose a hotel, pick a room type, enter the details of your stay, and provide your personal information and payment details. The next step is checking in when you arrive at the hotel, where you show your ID and get an access card. During your stay, you get to access the room assigned to you as well as the shared spaces. Once your stay is over, you check out of the hotel by returning your access card at the reception. Throughout your stay, the hotel keeps records of your activities for security and service purposes. For example, you might choose to enjoy the offerings from the minibar in your room, which often cost extra and are tracked and billed accordingly. All these mundane activities together constitute a perfect real-world example of *authentication*, *authorization*, and *auditing*—the three *A*s of IAM.

A hotel is a system. To access that system, you need to fulfill some criteria. The first step of booking includes defining the attributes of your identity. Once that is established, checking in by having your ID validated at the hotel reception is *authentication*. You being allowed to access only your room is *authorization*. Tracking and recording of usage of your access card and the hotel's paid services is *auditing*. In addition, as you know, the hotel staff have special access cards that allow them to enter any room to fulfill their duties. The activities that the hotel staff do with their privileged access is *administration,* which can be thought of as the fourth pillar of IAM.

IAM serves two primary functions:

Identity management
: *Identities* in the context of IAM are digital representations of entities that are allowed access to resources within a system. Managing identities involves verifying and authenticating the identities of entities, such as users, services, and systems, that are attempting to access resources within the system.

Access management
: Managing access means controlling and monitoring what resources authenticated entities can access within a system.

Any system intended for use by many users needs to implement IAM. Moreover, IAM is a crucial component of data and AI governance, as it is necessary to secure access to data and AI assets and enforce regulatory compliance and risk management. Needless to say, Databricks has IAM as a core part of its platform.

If you're new to Databricks and come from a background of using a SaaS solution, you might find it a bit difficult to navigate all the concepts just by diving into Databricks documentation. Therefore, in this chapter, we'll start with some fundamental concepts of Databricks and then gradually move on to IAM concepts. Furthermore, we'll focus on the *identity management* part of IAM in this chapter and dive deeper into access management in Chapter 5.

Databricks Constructs

If you have ever used the Databricks Platform, you would have experienced the UI of a Databricks workspace. A *workspace* is a Databricks construct consisting of various cloud resources that allow users to create, access, and manage Databricks workspace assets. It is common practice to have more than one workspace, and we have seen many organizations use separate workspaces to isolate the assets between development environments, teams, and business units.

As we briefly touched upon in Chapter 2, Databricks workspaces are associated with a Unity Catalog *metastore*, which is also a Databricks construct that acts as a store for metadata of all your data and AI assets. Unity Catalog metastores exist at the cloud-region level, and a workspace can only be associated with a metastore in the same cloud region.

All the workspaces within an organization usually belong to a single entity termed the Databricks *account*. An account is yet another Databricks construct that allows the admins to create and/or manage workspaces and Unity Catalog metastores and configure settings that apply across all the workspaces.

The relationship between a Databricks account, workspaces, and metastores is shown in Figure 3-1. A Databricks account can have multiple workspaces and metastores. All the workspaces in a given cloud region are associated with a metastore in the same region. In other words, you can have only one metastore per cloud region.

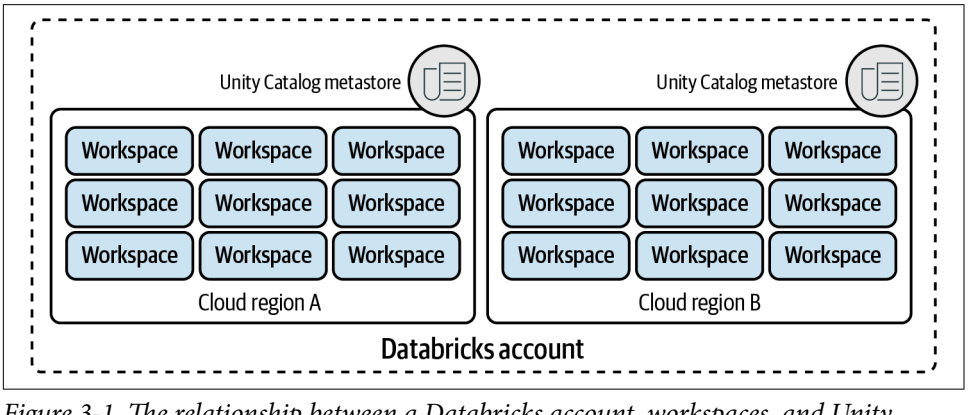


Figure 3-1. The relationship between a Databricks account, workspaces, and Unity Catalog metastores

Cloud-Specific Details

Since Databricks defines its constructs on top of cloud concepts, let's briefly examine the Databricks account and workspace hierarchy in the context of the major public cloud providers, namely AWS, Microsoft Azure, and GCP.

Databricks on AWS

For simplicity's sake, let's assume that your organization has just one AWS organizational unit and multiple AWS accounts. You can create Databricks workspaces in multiple AWS accounts that belong to a single Databricks account. In other words, a Databricks account can span multiple AWS accounts. Although you can create and use multiple Databricks accounts, it is recommended to have just one account to reduce complexity in managing identities, metadata, collaboration, and data distribution. On the other hand, you need to consider the limits that Databricks has on the maximum number of workspaces that you can create within a Databricks account. Figure 3-2 shows the relationship and cardinality between Databricks and AWS account constructs.

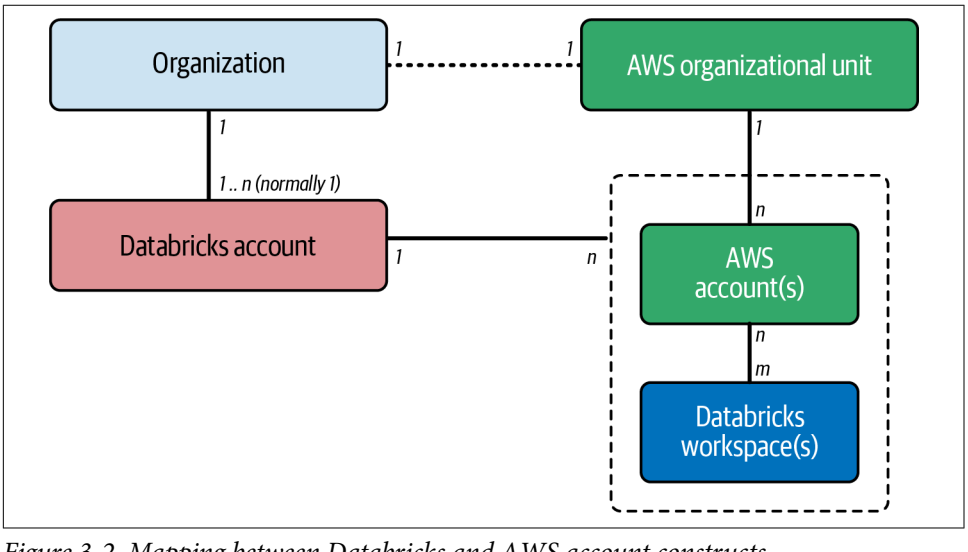


Figure 3-2. Mapping between Databricks and AWS account constructs

Azure Databricks

Databricks is a first-party service on Azure. This means you can use Azure Databricks like a native service offered by Microsoft Azure. Figure 3-3 shows the relationship and cardinality between Azure Databricks and Azure cloud constructs. In an organization, you usually have just one Microsoft Entra ID tenant. For a given Entra ID tenant, you can have only one Azure Databricks account. You can, however, create multiple Databricks workspaces across multiple Azure subscriptions. Unlike on AWS, on Azure, there are no documented hard limits on the number of Azure Databricks workspaces you can create per account. We have seen large organizations with more than 1,000 workspaces.

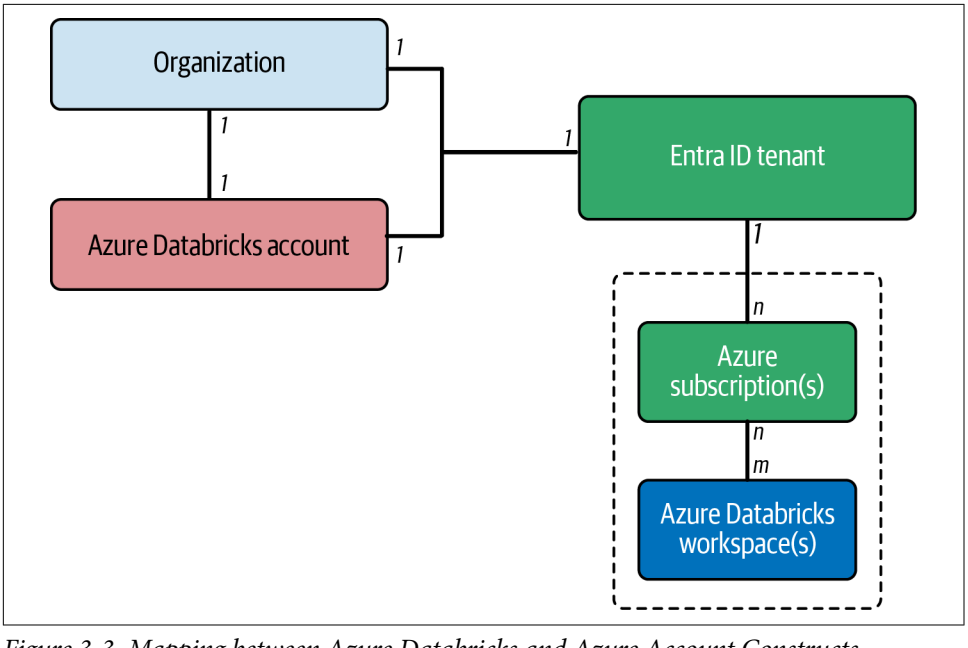


Figure 3-3. Mapping between Azure Databricks and Azure Account Constructs

Unlike on AWS and GCP, there are no specified limits on how many Databricks workspaces you can have on Azure. Moreover, creating a new workspace on Azure is relatively much simpler. This has led many Databricks customers to create a large number of workspaces, sometimes without any design considerations. Most of the time, the reason for creating a workspace is to isolate workspace-level resources.

Needless to say, it is better to have a design approach for creating workspaces, and the following are some of the considerations:

- Isolation at the business unit level: Clear separation of resources for the business unit.
- Isolation of operational tiers (e.g., development, stage, and production).
- Isolation at the platform level based on the requirements: For instance, some teams might need access to certain features of the Databricks Platform that could be enabled at the workspace level.
- Isolation based on the data category requirements: For example, you might have to dedicate workspaces for processing or anonymizing sensitive datasets.
- Isolation at the project level: Some projects might need dedicated resources to fulfill compliance requirements.

The idea is to have well-defined criteria for creating new workspaces based on your organization's real needs. In addition, you might need to set up monitoring and processes to enforce the criteria.

Databricks on GCP

Assuming that your organization has one GCP *organization*, you would have one Databricks account and multiple Databricks workspaces spread across GCP *folders* and *projects*. Figure 3-4 shows the relationship and cardinality between Databricks and GCP constructs.

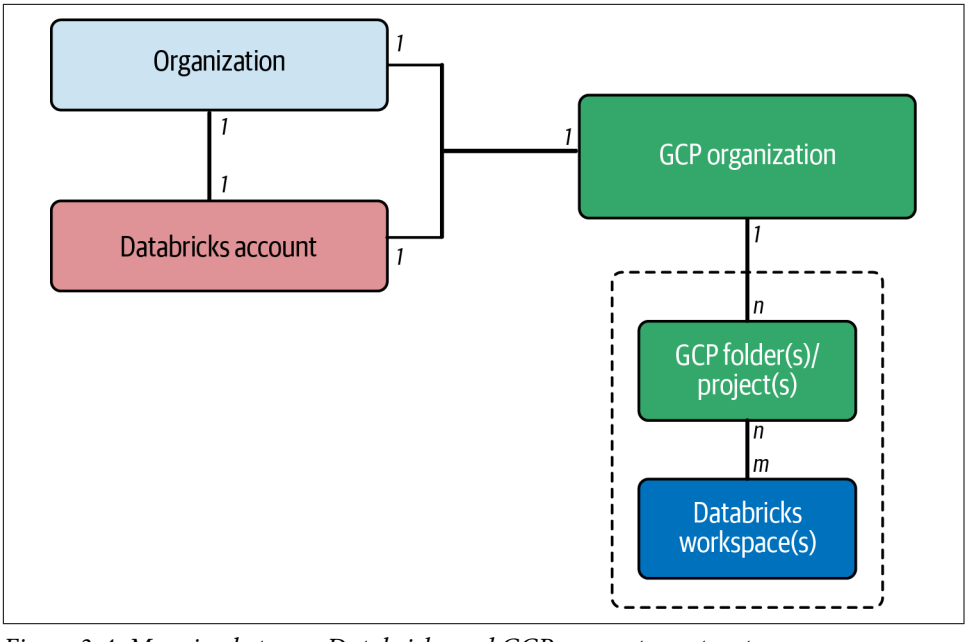


Figure 3-4. Mapping between Databricks and GCP account constructs

Access to Databricks and Beyond

The Databricks Platform is a cloud-based solution. This means that when you deploy Databricks, you create the necessary resources in the underlying cloud provider. These include the network, storage, security components, and more. Once the Databricks Platform is properly deployed and you start using the platform, it will create new resources in the underlying cloud provider to fulfil your requests. You can associate additional cloud resources, for example, object stores, with the platform and let it manage the assets stored on them. In other words, you will have a bunch of cloud resources in the scope of the Databricks Platform.

These Databricks-scoped cloud resources, especially the compute and storage, should be accessed via Databricks, and the access controls are set on the objects within the Databricks Platform. These objects, most of the time, are an abstraction on top of the cloud resources. For example, a Spark cluster within a Databricks workspace constitutes one or more virtual machines (VMs) in the underlying cloud provider. Although you can access all of these cloud resources directly if you have the cloud provider's IAM permissions granted separately, that is highly discouraged because it would mean that you are bypassing the strong security and governance capabilities offered by Databricks. Chapter 5 delves more into permissions modeling and access control mechanisms.

A typical, or let's say *ideal*, setup of Databricks is shown in Figure 3-5. Users, applications, and services access resources, artifacts, tools, and assets within Databricks. The platform users do not directly access the resources and assets in the underlying cloud provider. Access to the cloud resources is configured within Databricks. Therefore, the complexities of accessing the native cloud resources are abstracted away from the end user. Moreover, the access controls are applied to objects within Databricks. Databricks, in simple terms, is PaaS.

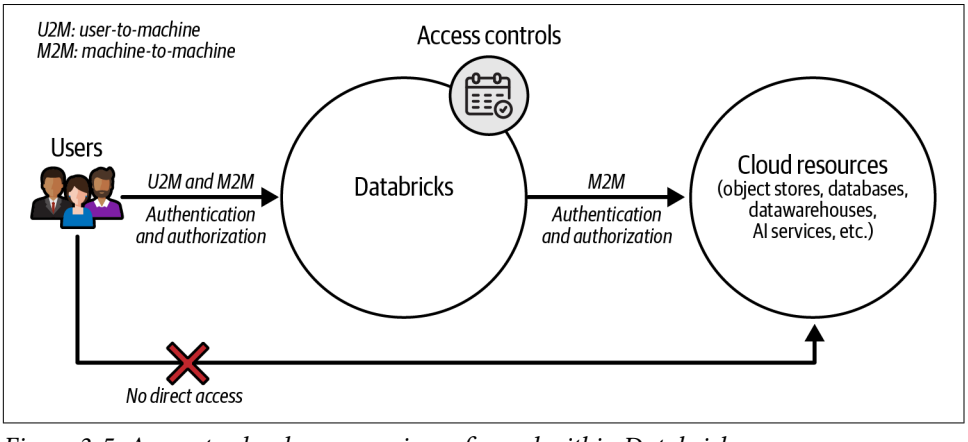


Figure 3-5. Access to cloud resources is configured within Databricks

Databricks Securables

Access control is all about who has access to what, right? Well, it's a bit more complex than that, but we'll stick with that for now. The *who* is what we refer to as a *principal*, which represents an *identity* that can access a resource or an asset. We cover identities in "Databricks Identities" on page 77. The *what* is something that should be protected from needless or unwanted access. In the context of a data platform, the *what* includes resources such as compute clusters, files, data and AI assets, and so on. We term these *securables*.

Securables within the Databricks Platform are at the workspace and metastore levels:

Workspace securables
: These are workspace-level objects such as clusters, jobs, notebooks, etc. These objects are scoped to individual workspaces and are therefore accessible only from that workspace and tied to its lifecycle.

Unity Catalog securables
: These are metastore-level objects such as catalogs, schemas, tables, views, and so on. These objects are scoped to individual metastores and can be accessed from any workspace associated with that metastore.

If you're not an account admin, you have access to neither the account console nor the account API.

Databricks Identities

In Databricks, identities exist at the account level, and you can allow them to access the account and workspaces. You can also grant privileges to the identities on the Databricks securables. Identities are the core component of IAM within Databricks, just like in any other PaaS or SaaS solution.

Databricks Identity Types

In Databricks, there are three types of identities:

User
: A digital representation of a human user uniquely identified using an email ID.

Service principal
: A representation of a technical, nonhuman user used by applications, services, and automation tools. A service principal is uniquely identified by its identifier.

Group
: A collection of individual users and/or service principals. A group can have another group as a member, resulting in a nested structure of identities. A group is uniquely identified by its name on the account level.

The authentication applies to users and service principals, whereas authorization applies to all three identity types.

Groups in Databricks

There are three types of groups in Databricks: account groups, system groups, and workspace-local groups (legacy). The permissions on Unity Catalog securable can be granted to only account groups. System groups are created and maintained by Databricks and cannot be deleted. At the account level, there is one system group called *account users*, which includes all the users and service principals within the account. At the workspace level, there are two system groups: *users*, which includes all the users and service principals assigned to that workspace and an *admins* group, which includes all the users with the workspace admin role.

Predefined Admin Roles and Responsibilities

From the platform perspective, there are two main categories of identities that interact with Databricks. One is the platform user who wants to implement use cases, and the other is the platform admin, who manages the platform, facilitates use case development, and enforces organization policies in terms of security and cost controls. You can allow Databricks Platform users (or simply users) to access one or more Databricks workspaces and grant privileges on securables. Three main administrative roles have privileged access to Databricks and are allowed to perform certain activities. Figure 3-6 shows the admin hierarchy.

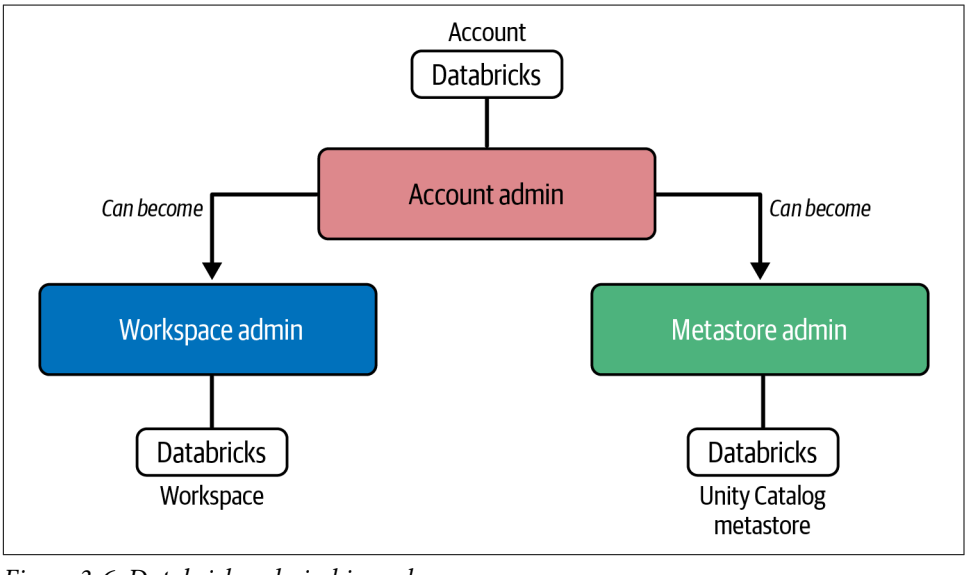


Figure 3-6. Databricks admin hierarchy

Table 3-1 lists the main admin roles and responsibilities/tasks along with the typical frequency that indicates how often they are performed.

Table 3-1. Databricks predefined admin roles and responsibilities

Scope	Role	Responsibility/task	Administration frequency (approximate)
Databricks account	**Account admin**	Account-level user provisioning setup	One time
		Manage network security configurations	Rare
		Feature enablement toggle	Rare
		Major issue/incident resolution (e.g., accidental deletion of users)	Rare
	Workspace admin	Create account groups	Frequent

Scope	Role	Responsibility/task	Administration frequency (approximate)
Unity Catalog metastore	**Workspace admin**	Create Unity Catalog securables (catalog, external locations, and storage credentials)	Frequent
		Delegate object ownership to specific users/groups	Frequent
	Metastore admin	Create and manage access to all Unity Catalog securables	OPTIONAL
		Delegate metastore-level privileges such as "CREATE CATALOG" to chosen users/groups	OPTIONAL (depends on the data architecture)
Databricks workspace	**Workspace admin**	Creating and managing compute resources and policies	Frequent
		Assign account groups to workspace	Frequent
		Manage workspace features and settings	Frequent

Account admin

This is the most powerful privileged role in the Databricks Platform. Being an account admin will allow you access to the Databricks account console and account APIs. Account admins can grant themselves access to any workspace and assign themselves administrative privileges on the workspaces and metastores. They can also assign account admin roles to other identities. You will need an account admin very rarely once you have fully set up the platform, provided that you have automated most of the account-level activities, such as user provisioning.

On AWS and GCP, whoever creates the Databricks account becomes the first account admin. They can then add other users and assign the account admin role, usually to a select few.

On Azure, it's different, as the user journey differs. Here, you do not create a Databricks account; you start by creating an Azure Databricks workspace. Once you have at least one workspace, you can access the account console. Since there are no account admins at this stage, Azure Databricks relies on a powerful Entra ID role to gain initial access to the account. *You need to have the Entra ID Global Administrator role to access the account console for the first time.* Once you log in to the account console, you become the first Databricks account admin. Now, you can assign the account admin role to other identities.

Because one or more workspaces already exist within the account, if any identities have been onboarded to those workspaces, you will see them in the account console. This might be confusing because you will be accessing the account console for the first time and will see identities that have already been onboarded. The reason is that identities in Databricks are at the account level—that is, even if you onboard identities on a workspace, the scope is still the account.

Metastore admin

This is an optional role that comes next to the account admin role in terms of privileges. A metastore admin has full privileges on the metastore, which means they have unbounded access to all the data and AI assets cataloged under that metastore. They cannot assign the metastore admin role to other identities, but they can individually grant all the privileges on the metastore to any identity. You do not need a metastore admin if you fully leverage the privileges of the workspace admin role and set up proper automation. Chapter 5 dives deep into the topic of who should be the metastore admin in a detailed discussion of access control mechanisms.

Workspace admin

A workspace admin has administrative privileges within the scope of the workspace. In addition, they have a few privileges on the account and metastore levels. This is the most relevant and useful admin role that you would need regularly.

In addition to the Databricks account, metastore, and workspace admins, there are additional predefined privileged roles used for administrative purposes, namely, marketplace admins, billing admins, group managers, and service principal managers. We will mention and explain those wherever they are relevant.

Interfaces to Access the Platform

There are multiple ways to use and manage the Databricks Platform, but fundamentally, there are two main interfaces: the UI and the REST API.

Databricks UI

Back in 2022, before joining Databricks, Karthik wanted to explore and understand the data platform that it offered. He signed up for a free trial to use Databricks on AWS. Databricks provided him with a link to log in, which pointed to the account console. Since Karthik signed up for Databricks, he automatically became the account admin and hence got access to the account console. He then went on to create a workspace by meticulously following the steps provided in the Databricks documentation. Once the workspace was up and running, he was able to log in to it, and the UI of the workspace was very different. The workspace UI looked more like the data platform he expected. The workspace UI is the one that is accessed and used by the account users or simply users.

If you are the account admin, you will have access to the account console for performing admin activities at the account level. If you are an account user, you probably have access to one or more workspaces. We say *probably* because you might be on the list of Databricks users but not have access to any workspaces at all because either the account admin or the workspace admin needs to give you access to a workspace before you can log in and use it. In addition, having access to a workspace does not automatically grant you access to any securables within the workspace until someone grants it.

Databricks REST API

Databricks provides APIs designed in REST architectural style to interact with the platform, and this has paved the way for two things. The first is the capability to automate the DevOps activities, and the second is the additional interfaces built on top of it. The following lists the tooling built on top of the Databricks REST API:

- Databricks command-line interface (Databricks CLI) for interactive deployment, testing, and debugging.
- Databricks Terraform provider for deploying and managing the data platform by automating workspace provisioning, deploying and managing workspace objects, Unity Catalog securables, and so on.
- Databricks SDK for Python and Java and Go for interactive development, testing, and debugging.
- Databricks Asset Bundles (DABs) to facilitate DevOps, CI/CD, and software development best practices for your data and AI projects.

Databricks provides APIs at the account level as well as the workspace level. Most of the operations that can be done using the UI can also be done through the APIs. Typically, you can do more with the APIs.

There is no separate UI or API for the Unity Catalog metastore in Databricks. The creation, deletion, and a couple of configurations of a metastore are done via the account console. Creating and managing the securables within the catalog are done through the workspace. Figure 3-7 depicts the interfaces for the account and the workspace. Interaction with the metastore happens either through the account or the workspace.

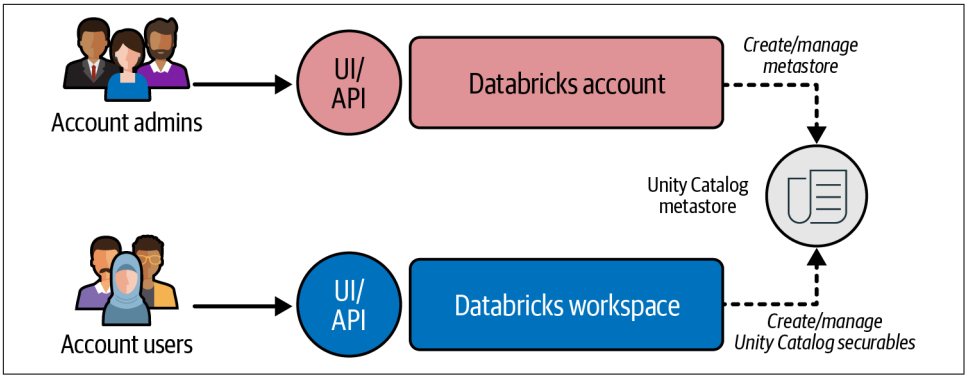


Figure 3-7. Databricks account and workspace context

Identity Provisioning

In most organizations, except maybe for very early-stage startups, identities are created, stored, and managed using a system known as an *identity provider* (IdP). The most commonly used IdPs are Microsoft Entra ID (formerly known as Microsoft Azure AD), Okta, OneLogin, and Ping Identity. In a given organization, an IdP is the source of truth for all identities.

This centralized approach for managing identities has several advantages, such as easier authentication handling, a simplified login mechanism for multiple systems, and streamlined user onboarding and offboarding. Moreover, you can enforce additional security measures, such as multifactor authentication (MFA). Therefore, whenever you introduce a new tool, service, or platform in your organizational tech stack, you plug in your IdP to facilitate part of the identity management process, such as user authentication.

The same applies to Databricks. When you deploy the Databricks Platform, you plug in your IdP. Although you can create, store, and manage identities independently within Databricks, you are better off using your IdP as the source of truth for all your identities. In other words, you create the users and groups in your IdP and provision them to Databricks. The identities within Databricks should be a mirrored version of what exists in your IdP.

Syncing Identities from Identity Provider to Databricks Account

Provisioning identities is not a one-time activity. You need to keep the identities in your IdP and Databricks in sync. This is necessary to make sure any changes to the identities in your IdP, such as changes in group memberships, deactivation of users, and so on, are reflected in Databricks. There are a couple of ways to set up the sync, one is automatic identity management, which we will discuss in "Automatic

Identity Management with Microsoft Entra ID" on page 84, and the others are based on the SCIM, an open standard protocol designed to simplify and automate identity management across multiple systems and applications. The following are the SCIM-based provisioning options to sync identities from your identity provider to Databricks:

SCIM provisioning connector
: The SCIM connector is an application within your IdP that you can configure to connect to the Databricks account via the SCIM protocol. It runs periodically to sync the provisioned identities to Databricks.

SCIM REST API
: Databricks provides the SCIM REST API, which can be used for creating and managing identities within Databricks. To sync identities from your IdP, you read the identity information using the APIs provided by your IdP and create or update the information in Databricks using the Databricks account SCIM API. You can automate the entire provisioning and syncing using custom scripts written in programming languages such as Python or using IaC solutions such as HashiCorp Terraform.

Identity provisioning is from the IdP to the Databricks account. Databricks also supports identity provisioning from the IdP to the Databricks workspace, but this is now legacy.

Why would you choose one over the other? That is indeed an important aspect to think about, potentially evaluate, and then make a decision about. From what we have seen, large organizations with well-established identity management and access request processes tend to go with the SCIM provisioning connector. This is because they are already experienced with it, and most of the time, such standard solutions are already integrated with an internal ticketing system that is used to manage access requests from users. Moreover, there is already a clear owner who would create and manage such an application. However, SCIM connectors have some limitations. For example, you cannot provision service principals and nested groups using the SCIM connector offered by Microsoft Entra ID.

Tech-savvy organizations tend to use custom scripting, which gives them flexibility in terms of what they can provision and how frequently they can sync. Although they need to develop and maintain these scripts, the setup is relatively simple and does not require frequent upgrades to the logic or the endpoints.

There are also situations where organizations take a hybrid approach: the SCIM connector for provisioning users and flat groups and custom scripts for provisioning

service principals and nested groups. The scripts overcome the limitations of the SCIM connector.

Irrespective of which option you choose, the provisioning and syncing mechanism can be fully automated. The SCIM connector uses a SCIM URL and token generated by a user with an account admin role. The scripts should be run using a service principal with an account admin role.

Syncing identities from an IdP to Databricks is unidirectional, as the IdP is always the source of truth for all identities.

Automatic Identity Management with Microsoft Entra ID

If you use Microsoft Entra ID as your identity provider, you have the option to use the *automatic identity management (AIM),* which is a more convenient option than using SCIM sync. Using AIM, you can directly onboard the identities present in Entra ID rather than creating a copy of those identities in Databricks. As of this writing, AIM is available only with Entra ID and only for Azure Databricks.

When it is enabled for your Azure Databricks account, the workspace admins can search and add users, service principals, and groups directly from the Entra ID to the workspace. There is no need to set up any sync mechanisms. Entra ID will be the source of truth for all the principals, and hence any changes to those principals are reflected in Databricks. For example, if a user is removed from Entra ID, that user will be deactivated in Databricks, and they can no longer log in. One more advantage of this option is that it does not have the limitations that SCIM connectors have, i.e., service principals and nested groups are also supported.

Furthermore, AIM makes it much easier to onboard business users to the Databricks ecosystem. For instance, if a user created an AI/BI dashboard in a workspace and wants to share it with a business user who has no idea about Databricks, then they can easily share the view-only version of the dashboard with the user based on their Entra ID identity. The receiver will be automatically added to the Databricks account, but not to the workspace, and they can view the dashboard. This has two advantages. First, you do not have to add the business user to the Databricks account using the SCIM sync mechanism, and therefore, it is much faster and more convenient to share the dashboard with the user. Second, since you are not adding the user to the workspace, you are inherently following the least privilege principle, and you have fewer users on the workspace. Sharing workspace-level objects, such as dashboards, with nonworkspace users helps to scale. You can share dashboards with thousands of business users without worrying about reaching the workspace-level identity limits.

Databricks Workspace Assignment

Provisioning users from your IdP to the Databricks account will not grant access to anything within the account. For a user to access a workspace, the user needs to be assigned to that workspace. An account admin can assign identities present in the Databricks account to any workspace in that account. A workspace admin can assign account-level identities to the workspace of which they are an admin. Figure 3-8 shows the flow of identities from IdP to the Databricks account and then to workspaces. Assigning account-level identities to workspaces is also automated and is typically done using the REST APIs or Terraform scripts.

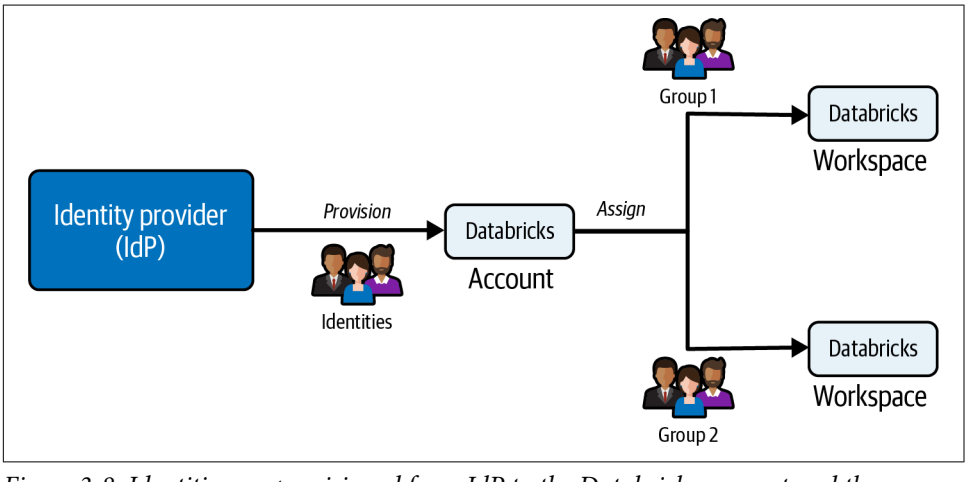


Figure 3-8. Identities are provisioned from IdP to the Databricks account and then assigned to Databricks workspaces as required

At Nexa Boutique, the IdP is Microsoft Entra ID, where all the identities are created and managed. For any new person joining the organization, a new identity is created in Entra ID, and if the person leaves, the identity is purged. There is only one Entra ID tenant where all the identities exist and it acts as the source of truth, especially regarding users and groups, for all the systems within the organization.

When Nexa started its data platform journey with Databricks, Unity Catalog and Databricks account level identity management did not exist. This meant Nexa had to provision identities from Entra ID to each Databricks workspace separately. This is what we call *workspace-level identity provisioning*. It works the same way as account-level identity provisioning, the main difference being that you configure the workspace as the target of provisioning instead of an account. It is also based on SCIM. Workspace-level SCIM provisioning is now legacy, and an interesting thing to note here is that this feature was in Public Preview and will never be generally available—a rare example with Databricks.

Nexa's data platform, based on Databricks, has significantly grown in usage over the years, and they now have hundreds of workspaces and tens of thousands of users and growing. Now, with Unity Catalog and centralized identity provisioning, identity management within Databricks in general has become much simpler. Of course, they had to go through the pain of migrating the workspace-local groups and corresponding access controls to account level. But they had help from Databricks field engineering teams, and they leveraged a tool called UCX, which is an open source project developed as part of Databricks Labs, to help Databricks users upgrade their legacy Databricks setup to a Unity Catalog based setup. The UCX tool automates the process of upgrading workspace-local groups to account groups. We will cover the topic of migrating to Unity Catalog in Chapter 11.

Nexa chose the hybrid approach for provisioning identities from Entra ID to the Databricks account. They use the Microsoft Entra ID SCIM connector for provisioning users and groups, and a custom script written in Python to provision Entra ID managed service principals. This means the major part of identity provisioning is done by the SCIM connector. The reason for this choice was simple: Nexa's cloud team was already familiar with it and was willing to own it. The SCIM connector comes with no extra cost—there is no need to maintain it. Moreover, it comes with logging of events that is helpful in debugging and audits. Using a script to provision Entra ID managed service principals was not really a choice but a necessity, as the SCIM connector does not support the provisioning of Entra ID managed service principals. The frequency of sync for the Entra ID SCIM Connector is every 40 minutes, and the script is configured to run every hour.

As mentioned in Chapters 1 and 2, Nexa has a multiple cloud setup, mainly with AWS in the US and Azure in Europe. Naturally, they have more than one Databricks account: one Databricks account on AWS and one Azure Databricks account corresponding to the Azure tenant. Nexa uses two SCIM connectors for provisioning users and groups: one for Databricks on AWS and one for Azure Databricks.

Eventually, when AIM became available as a Databricks Public Preview feature, Nexa started adopting it for its Azure Databricks account. It prevents the overhead of maintaining the SCIM provisioning connector and an additional script for provisioning service principals. Nexa plans to extend the AIM for Databricks on AWS whenever it becomes available.

User Access Provisioning and De-provisioning

Nexa uses ServiceNow for IT Service Management (ITSM) and wanted to manage Databricks account and workspace access provisioning through ServiceNow. ServiceNow integrates well with Microsoft Entra ID. So, if and when a user wants to get access to a Databricks workspace, for instance, they would request it in the ServiceNow portal and the approvers would be notified. Once approved, the user identity would be added to the Entra ID group that has access to the corresponding workspace, and the user would be notified. Upon the next identity provision sync, the user would be added to Databricks as part of the group. Figure 3-9 shows a simplified version of the user access provisioning flow.

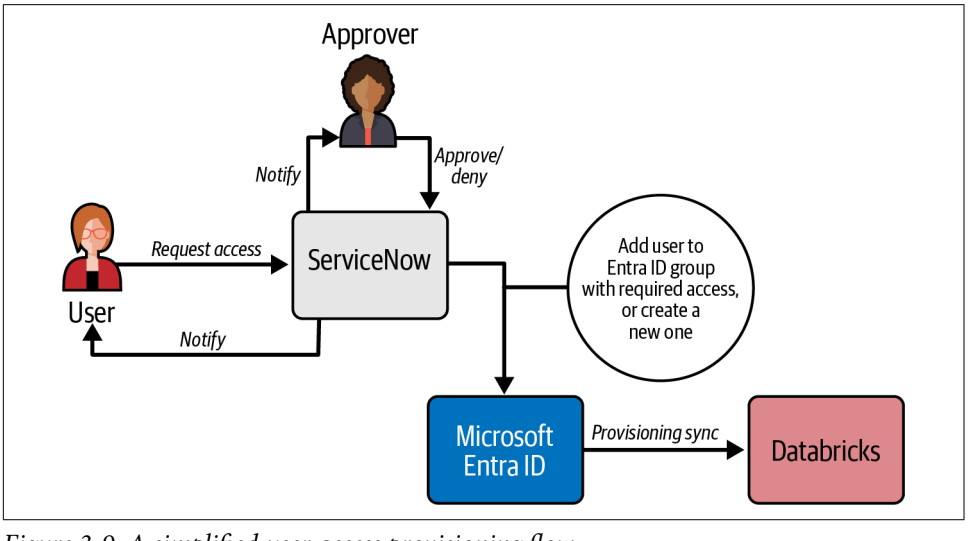


Figure 3-9. A simplified user access provisioning flow

Whenever a user leaves the organization, the user identity is disabled or deleted in Entra ID. This will result in the user being deactivated in the Databricks account upon the next SCIM sync. Whenever an identity is deactivated in a Databricks account, it is also deactivated in all the workspaces. Figure 3-10 shows the user de-provisioning flow. A deactivated identity should be deleted to completely purge it from Databricks. An identity that is deleted from the account is also deleted from all the workspaces.

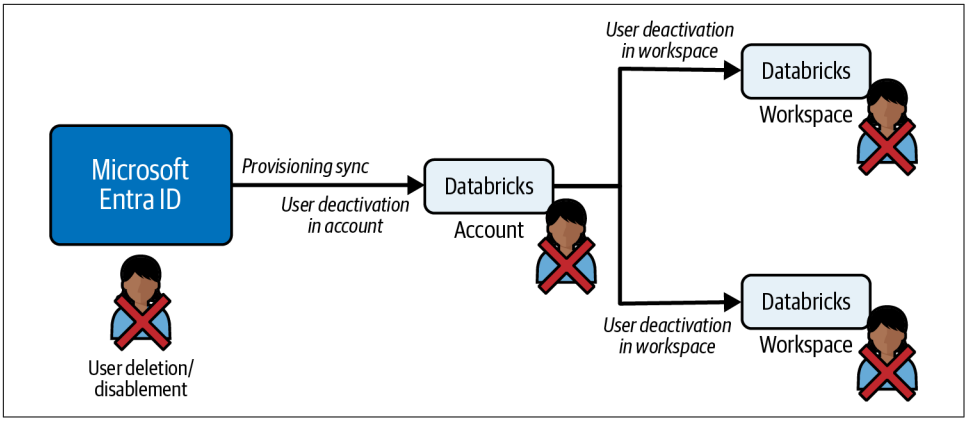


Figure 3-10. User de-provisioning flow

As you may have noticed, you can manage the user provisioning and de-provisioning flow with ITSM and IdP without directly interacting with Databricks, except for cleaning up orphaned identities.

Single Sign-On

Irrespective of our technical knowledge level or years of experience in tech, we are all guilty of one thing, at least most of us, and that is using the same password across multiple applications or services. We know it's not secure, but still, we do it. According to some studies, this is the case for the majority of the population.

Think of an organization that has multiple applications and services. If all the users in that organization had to maintain a different set of credentials for every application and service, what do you think would happen? Just for the sake of argument, let's say everyone maintains a different set of credentials. How much time would they spend entering those every time they wish to log in to something? How many password reset requests might be raised? You get the idea.

This problem is addressed by SSO, an authentication method that allows users to log in to multiple applications and services with a single set of credentials. This enhances UX while improving security and efficiency in an organization. Most of the commonly used IdPs support SSO, and you can set up SSO for Databricks.

There are various standard open protocols used to implement SSO. Databricks supports two of the most commonly used protocols:

OpenID Connect (OIDC)
: OIDC is an open authentication protocol developed by the OpenID Foundation and published in 2014. It extends the OAuth 2.0 authorization framework to provide a standardized method for user authentication and identity verification. OAuth 2.0 is designed only for authorization, and it enables applications to securely access resources on behalf of users without sharing their credentials.

Security Assertion Markup Language (SAML)
: SAML is an open standard protocol for authentication and authorization developed by OASIS and first released in 2001. It is primarily used to verify user identity and determine what resources or services the authenticated user has access to.

We won't go deeper into this topic as it is beyond the scope of this book.

Cloud-Specific SSO Options

The following are the SSO configuration options for each cloud provider:

Databricks on AWS
: Once you configure SSO on the account console, you will have an option called Unified Login. By enabling Unified Login, you just have to manage one SSO configuration for the account and all workspaces that have Unity Catalog enabled. This means you do not have to configure SSO for each workspace separately. Alternatively, you still have the option to enable SSO for each workspace separately, but this is not recommended.

Azure Databricks
: The SSO is backed by Microsoft Entra ID and is available by default for both the account console and the workspaces.

Databricks on GCP
: The SSO based on Google Cloud Identity (or GSuite) is available by default for both the account console and the workspaces. You can configure your Google Cloud Identity account (or GSuite account) to federate with an external IdP such as Microsoft Entra ID, Okta, and so on to verify user credentials. As of this writing, Databricks is working to simplify this by allowing any IdP to be configured directly with Databricks on GCP for SSO. This will reduce administrative overhead.

Nexa Boutique leverages SSO, which is available by default with Azure Databricks. For Databricks on AWS, they have enabled Unified Login for SSO on the account, and all their workspaces on AWS use a single central SSO configuration. Nexa uses the OIDC protocol as recommended by Databricks. The Unified Login for SSO benefited the company in the following ways:

- Simplified administration by lowering total cost of ownership (TCO) with centralized management and all the new workspaces were born secure.
- Easier onboarding of new users to features such as AI/BI dashboards, Genie spaces, and Databricks apps.
- Increased productivity by having a single account-wide SSO config, which meant less troubleshooting and hence happier users.

Programmatic Authentication Methods

One way to use the Databricks Platform is by logging into a workspace in your browser to perform the desired activity. The Databricks UI is feature-rich. You can leverage the notebook interface to develop code and execute it. You can set up jobs to run periodically and debug the ones that fail. You can view the data and its metadata directly within the UI. It's also great for ML and AI model training, evaluation, and deployment. You can also visualize your datasets and create dashboards. In essence, the Databricks UI caters to various personas such as data engineers, data scientists, data analysts, and ML or AI engineers.

Despite having access to a great UI, you might want to access Databricks and the data cataloged in it through external applications and services. For example, you might prefer VScode over a Notebook for coding, you might want to visualize the data in Power BI or Tableau, or Airflow might be your favorite tool for orchestrating ETL processes. All these interactions directly or indirectly use the APIs offered by Databricks, and you need to set up proper authentication for them to function.

Until now, we've primarily focused on user authentication. Let's take a quick look at how you can authenticate to Databricks programmatically:

Interactive
: For interactive programmatic access to Databricks, in other words, for attended authentication scenarios, you can use the OAuth user-to-machine (U2M) authentication mechanism. If a user is connecting to Databricks and performing some activities, then the U2M authentication method is suitable. For instance, if you are writing code in your IDE that you would like to run on Databricks, then you connect to a Databricks workspace using U2M OAuth authentication from your IDE. In addition, applications and services can get API access to a Databricks account and workspace resources on behalf of a user. In simple terms, OAuth U2M authentication is based on short-lived OAuth tokens for users.

Noninteractive
: There are a few ways to manage authentication for noninteractive access to Databricks, in other terms, unattended authentication scenarios:

OAuth token federation
: This is the most secure and recommended way for authenticating to Databricks, as it enables you to securely access Databricks APIs using tokens from your IdP. We will cover this in detail later in this chapter.

OAuth machine-to-machine (M2M) authentication mechanism
: Using this option, applications and services can get API access to Databricks account and workspace resources using an OAuth access token associated with a service principal. In simple terms, OAuth M2M authentication is based on short-lived OAuth tokens for service principals. This option is suitable for automated setups and when applications or services need to independently connect to Databricks. For instance, if you wish to manage the Databricks resource deployments through Terraform, you can use a service principal and corresponding OAuth token to authenticate Databricks Terraform provider against the Databricks account or workspace. This option should be used in case, for some reason, you cannot use OAuth token federation, and make sure you regularly rotate the service principal client ID and secret.

For authentication using Personal Access Token (PAT)
: Databricks PAT tokens can be used to get API access to Databricks workspace resources on behalf of a user or a service principal. PAT tokens are less secure because they are usually long-lived and do not automatically refresh. Therefore, PAT tokens should be used as a last option when the target application or service does not support any other authentication methods, such as OAuth.

PATs Are a Security Risk

From the security perspective, it is highly recommended not to use PATs, mainly because PATs are long-lived secrets, with a default max lifetime of two years, and this is the root of several issues:

- They end up being created and forgotten.
- These are bearer tokens, which means, whoever has them can use them.
- These tokens can fully impersonate the user, that is, someone with the token has access to everything that the user does.
- These tokens are still valid when the user's role changes, which can lead to privilege escalation. For example, if you, as a Databricks workspace user, create a PAT and use it to run an ETL job, and you eventually become the workspace admin, the PAT token that you created now has workspace admin privileges without you intending it to.

Even if for some inevitable reason, you need to use PATs, it is best to put some restrictions:

- Set a lower expiry for newly created PATs, typically less than 90 days.
- Revoke all old PATs. Databricks automatically revokes all PATs that haven't been used for 90 days or more.
- Account admins should monitor and revoke unused PATs from the account console through the token report.

For more details on this and other related topics, see this video.

Cloud-Specific Authentication: Azure Databricks

In our opinion, Azure has the upper hand when it comes to authentication and authorization mechanisms, especially in terms of simplicity. Apart from offering a powerful and well-integrated IAM solution, it also offers authentication methods that streamline access across its resources, which simplifies resource-to-resource access management. Let's look at an example. Let's say you are building a simple application that uses a virtual machine (VM) and a storage account. If you want to access the storage account from the VM, the best option is to use a managed identity (MI), which is a security feature used for authentication and authorization in Azure. MIs are suitable for use within the Azure environment. Moreover, as the name implies, Azure manages the credentials, and you do not need to provision or rotate any secrets, which means less maintenance and increased security.

However, not all Azure resources support MIs, in which case you can use an Entra ID service principal or Azure service principal (SP), which is also a security feature used for authentication and authorization. Unlike MIss, Azure SPs require manual management and rotation of secrets. Therefore, Azure SPs are considered less secure than MIs and should be used only when a MI is not an option.

In essence, if you want to access Azure Databricks from any other Azure resource, you can either use MIs, provided that the resource supports it, or you can go with an Azure SP. For developing and running code or experimenting with Azure Databricks interactively, you can also use the Azure CLI, provided by Azure, to connect to Databricks accounts or workspaces.

As an authentication mechanism, you generate and use Microsoft Entra ID tokens for Azure MIs, Azure SPs, and users. *Entra ID tokens* are a type of security token that serves as proof of authentication. These tokens are short-lived and typically expire within an hour, making them best suited for an automated setup.

Cloud-Specific Authentication: Databricks on GCP

On GCP, there are *service accounts,* which are very similar to Azure SPs. You can use GCP credentials authentication to access Databricks accounts or workspaces. The authentication is based on OAuth tokens corresponding to service accounts that act as Databricks users. Another option is Google Cloud ID Authentication, which uses the Google CLI to authenticate. Like the previous option, it also uses OAuth tokens corresponding to service accounts that act as Databricks users.

OAuth Token Federation

Using long-lived personal access tokens or OAuth client secrets to access Databricks APIs can be insecure and result in maintenance overhead. Imagine having to manage hundreds and thousands of secrets across the organization, or the *security debt,* which is the same as technical debt but in the context of security, that would result from allowing individual teams to create and use PAT tokens or OAuth client secrets. Figure 3-11 shows an application accessing the Databricks APIs using a PAT token or OAuth secret associated with a service principal. Note that you assign privileges to the service principal, and the application using the service principal's PAT token or OAuth secret can utilize those privileges. The tokens or secrets shared with the application typically have a long lifetime, and in case it's leaked might result in security incidents.

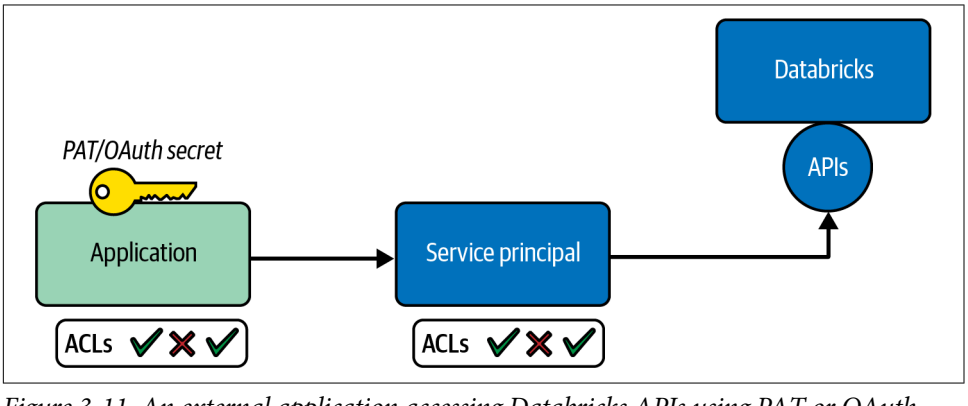


Figure 3-11. An external application accessing Databricks APIs using PAT or OAuth secret of an SP

An alternative approach is to use the *OAuth token federation* that allows you to use tokens from your IdP and exchange them to get short-lived Databricks OAuth tokens to access its APIs. Since the OAuth tokens are short-lived and there are no Databricks-managed secrets involved, this approach is inherently more secure. Consider a scenario where you want to automate running the workloads externally using a DevOps solution such as GitHub Actions or Azure DevOps. Rather than

storing the PAT tokens or OAuth secrets, you could use *workload identity federation*. A Databricks account admin can configure workload identity federation using a *service principal federation policy.*

A service principal federation policy is associated with an SP in Databricks and specifies the IdP, termed the *issuer,* from which the SP can authenticate, and the workload identity, specified using a *subject* and *audiences,* that is permitted to authenticate as the SP. An example service principal federation policy for GitHub Actions looks as follows:

```
{
"iss": "https://token.actions.githubusercontent.com",
"aud": "https://github.com/my-github-org",
"sub": "repo:my-github-org/my-repo:environment:prod"
}
```

Figure 3-12 illustrates the workload identity federation mechanism, which functions as follows:

1. Set up the service principal federation policy that corresponds to a Databricks SP.
2. The client (external application/tool) issues a request to the IdP to get the federated token.
3. The client issues a request to the workload runtime through the Databricks token endpoint using the federated token, requesting an OAuth access token.
4. The workload runtime validates the request and issues an OAuth access token.
5. The client accesses the Databricks APIs using the OAuth access token.

You can also set up an account-wide token federation using *an account federation policy.* This policy will allow all users and SPs in your Databricks account to access Databricks APIs using tokens from your IdP, allowing you to centralize the management of token issuance policies in your IdP.

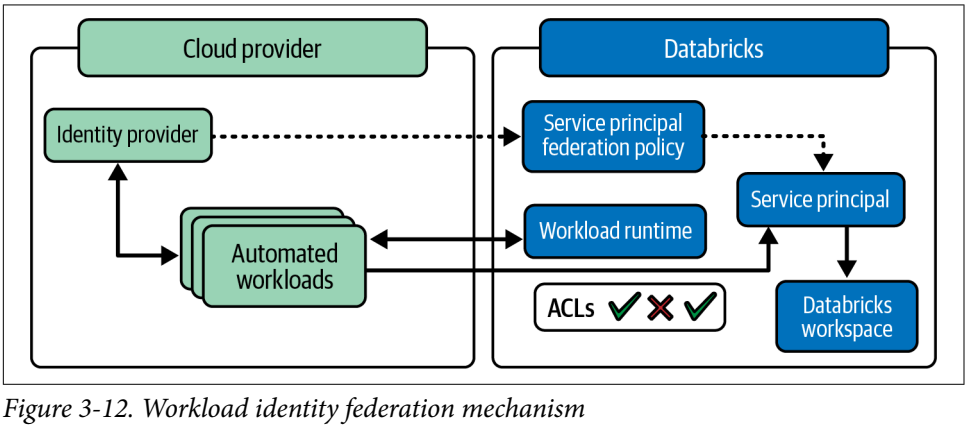


Figure 3-12. Workload identity federation mechanism

Identity Best Practices

Nexa Boutique takes a highly disciplined approach to managing its tech stack. It understands the importance of keeping technical debt at bay, so it always tries to follow best practices. Nexa has a great relationship with Databricks, and it regularly syncs with the Databricks field engineering teams to stay up to date on the latest developments, best practices, and product vision.

Nexa observes the following best practices, which it developed in collaboration with Databricks. These work best for Nexa, and most of these would apply to other organizations as well:

- Nexa has enabled MFA and conditional access policies in Microsoft Entra ID to secure their Databricks Platform setup.
- Nexa creates and maintains all its identities in Entra ID, including Databricks-specific groups, which are then provisioned to the Databricks account. Any changes to the group membership are made in Entra ID. In general, no changes to the group membership are allowed within Databricks. The groups that originate in IdP are termed *external* groups and are, by default, immutable in the Databricks account console and workspace admin settings page.
- Nexa avoids creating nested groups to prevent complicated permissions modeling.
- The access controls, as in access to workspaces and privileges on Unity Catalog securables, are applied to groups rather than individuals. This prevents the need for changes to the access controls when the status of individuals changes—for example, if an individual leaves the organization or moves on to a different project. This reduces the maintenance and keeps the privileges clean.
- Nexa disabled PAT tokens on all workspaces by default. If anyone needs to use PAT tokens, they must get an exception from the security team by providing a convincing reason. All such exceptions are documented and regularly reviewed.
- Nexa avoids using any legacy features. They have decommissioned all their workspace-level SCIM provisioning and only provision identities at the account level.
- Nexa has enabled Unified Login for Databricks on AWS, so it needs to maintain SSO configuration only at the account level.
- Nexa does not have a permanent account admin. Instead, it elevates pre-approved users to account admin roles on a temporary basis to perform any activity, such as enabling a feature or handling escalations. For this purpose, it employs the Microsoft *Entra Privileged Identity Management (PIM)* feature.
- Nexa monitors the Databricks audit log events related to identity and access in the system tables to detect anomalies and raise alerts.

Summary

Controlling and managing who can access what is a crucial part of any system within an organization. It starts with identities that you define in your IdP. Identities in Databricks exist at the account level. You can leverage your IdP for SSO and for managing identities, which you can provision to Databricks either through AIM or using the SCIM protocol and regularly sync. Once the identities are provisioned to the Databricks account, you can assign them to workspaces.

Various admin roles within Databricks help you manage identities and control access to accounts and workspaces. You can primarily use the Databricks UI or the APIs to access the accounts and workspaces. Databricks supports multiple authentication methods apart from SSO, such as OAuth token federation, OAuth tokens for users and service principals, and PAT tokens, which are all relevant for programmatic access to the platform.

This chapter focused primarily on the identity management part of IAM. In Chapter 5, we will dwell on access management and specifically focus on access control mechanisms on workspace and Unity Catalog securables. We will also cover broader governance topics, such as FGACs and attribute-based access controls, in Chapter 5.

CHAPTER 4
Unity Catalog and Compute

"Technology; The Multi-User Computers" is an article in the *New York Times*'s print edition on August 23, 1984. A quote from the article from James S. Campbell, chairman of Fortune Systems: "Until now, the world has not really believed in Unix and multi-user systems. Now it will."

Fast forward to 2006, when AWS announced the first cloud platform by introducing S3, closely followed by Elastic Cloud Compute, which became the backbone of many modern cloud-based applications. 2008 saw the launch of Google Cloud and the release of its App Engine. The same year, Microsoft announced its cloud computing operating system, Windows Azure. Advancements in cloud computing have since pushed data platforms forward to the cloud, and today, all major data processing platforms are cloud native, and Databricks is no exception.

When you entrust your most valuable and sensitive data to cloud platforms, extending that trust to the cloud-based data processing platforms that help you unlock its full potential is essential. Trust builds when you understand the governance standards implemented in the data platform. This chapter aims to further strengthen your trust in the Databricks platform when using Unity Catalog for your governance needs.

At Nexa Boutique, data governance is a top priority, and it played a central role in the decision-making process when selecting Unity Catalog as the governance solution for its Data Intelligence Platform. The data architects at Nexa recognized that a metastore is just one piece of the data governance puzzle in a cloud-based data platform. The compute engine's ability to enforce FGAC on user data is equally vital for a secure data platform. Multiple compute engines processing data complicate FGAC implementation, making it harder to maintain consistent governance standards. Databricks Data Intelligence Platform with Unity Catalog stands out by delivering consistent governance capabilities across all its compute engines, regardless of how end users consume the data. Understanding Unity Catalog's compute infrastructure, including

its capabilities and limitations, made the decision easier for the data architects at Nexa. This chapter explores how Databricks' compute options are carefully designed and governed by Unity Catalog to establish and maintain the trust of its users.

When designing a data governance solution, there are two parts to the puzzle: a metastore and the compute engine. Here, we discuss different architecture patterns for implementing a data governance solution and its complexities. Databricks offers its compute options in multiple flavors, including classic and serverless architectures. We explore the classic compute options here with support for Unity Catalog and explain how you should use them for your compute requirements. Serverless computing is a powerful option that supports Unity Catalog access controls. We'll explore the architecture and identify the specific use cases where serverless is the most suitable choice.

Implementing Governance: A Two-Part Problem

As the data architects at Nexa rightly pointed out, merely defining access controls within a metastore is insufficient to ensure effective data governance. Instead, all compute engines interacting with the data must rigorously adhere to and implement the access controls specified in the metastore. This requirement proves to be a significant technical challenge. Databricks offers different compute types, including classic generic compute, serverless generic compute, SQL Warehouse Pro, and SQL Warehouse serverless. On top of these compute types, users access data via multiple programming languages and interfaces, including SQL, Python, Scala, R, and API-based remote executions, which makes implementing a common governance framework across various tools and compute engines complex.

When designing a data governance solution, it is helpful to break it down into two fundamental components:

The metastore
: Serves as a centralized repository for all access controls and metadata about the data

The compute engine
: Enforces access controls to ensure that FGACs, such as row-level filtering and column-level masking, are applied consistently

The second component, the compute engine, poses the most significant challenge. If enforcing access controls were a straightforward task, the problem of centralized governance would be largely solved. But it's not. The complexity of enforcing access controls on a compute engine is what makes centralized governance so tricky to achieve. The most straightforward implementation of data governance would be assigning users a dedicated compute resource to run a single language, similar to a traditional RDBMS, allowing them to interact with the compute, as illustrated in Figure 4-1.

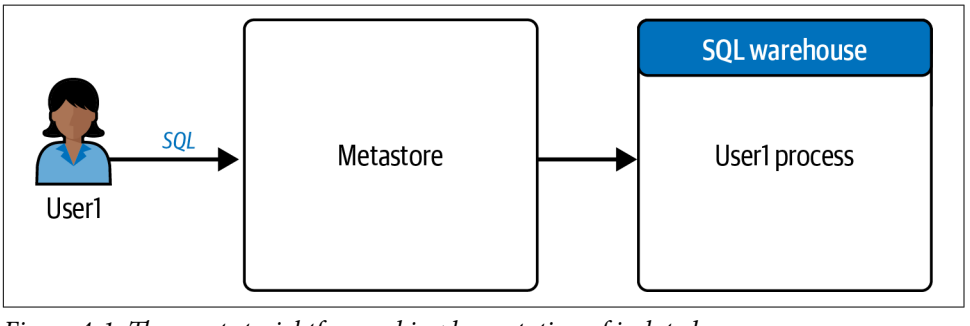


Figure 4-1. The most straightforward implementation of isolated governance

Sharing the metastore across multiple compute engine types and accessing data through different programming languages complicates the challenge, as each compute engine must enforce access controls consistently and reliably, regardless of how end users consume data. This requires a compute layer that is trusted and compatible with the established governance standards. Anything less would compromise the integrity of the governance solution. The complexity of implementing fully isolated data processing grows exponentially with three key factors:

- The number of users
- The variety of programming languages supported
- The different compute engines involved

While other factors, such as the storage layer architecture and processing capacity, also impact performance, these three factors have the most significant influence on the overall complexity of implementing isolated data processing. The complexity plotted in a chart will resemble the graph in Figure 4-2.

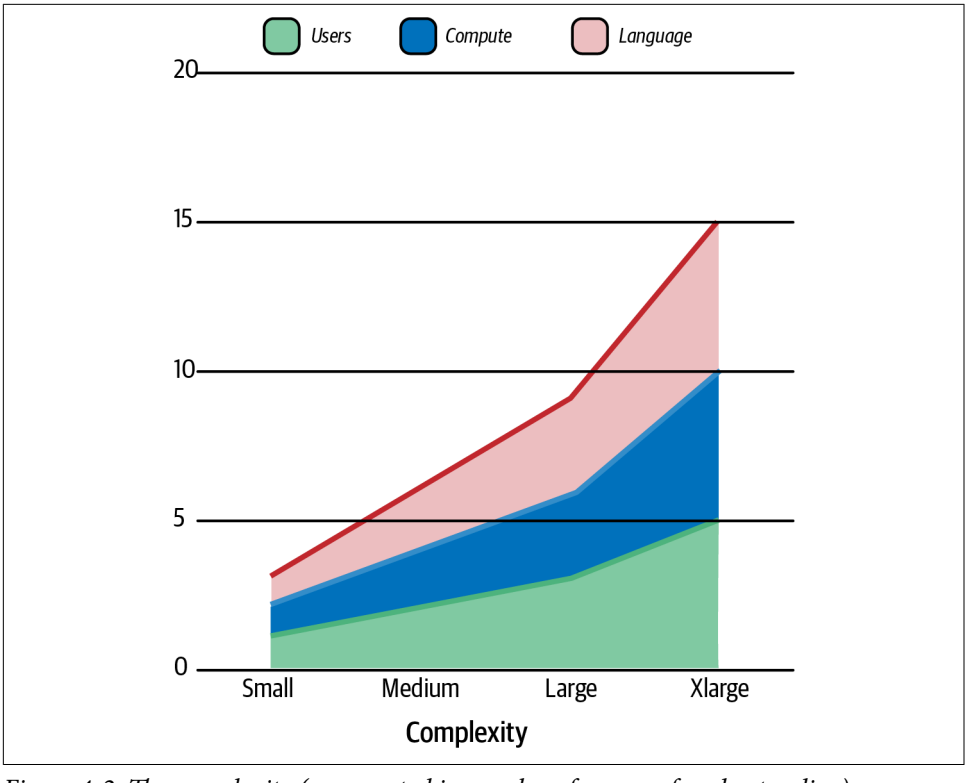


Figure 4-2. The complexity (represented in numbers for ease of understanding) grows as users, compute, and supported languages increase

Deploying dedicated compute resources per user is not scalable at the enterprise level, as costs can escalate rapidly. To mitigate this, compute engines must support multiple users, enabling resource sharing to reduce costs. The next step in achieving this is implementing process isolation, which separates each user's processes from interacting with others, including temporary files generated during processing, as shown in Figure 4-3. Although this architecture is complex, interpreting it is relatively straightforward. Again, it is similar to the RDBMS systems as it supports only one querying language for user access and is widely supported by many cloud data-processing tools.

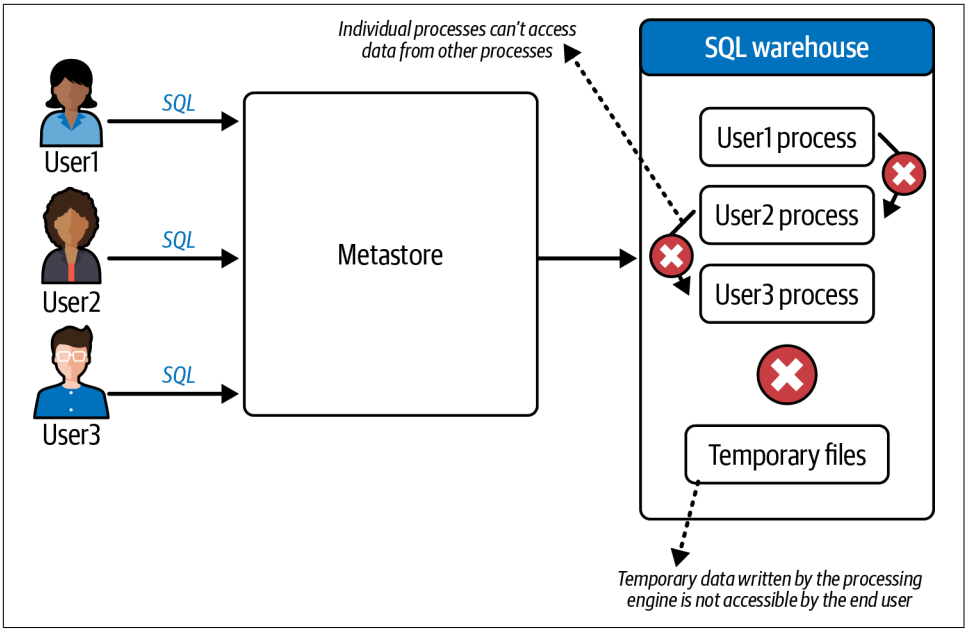


Figure 4-3. Process isolation enabling compute resource sharing

A data platform must accommodate multiple programming languages to be effective, as the specific requirements can differ significantly between organizations. The complexity increases when enforcing governance from a central metastore on compute engines that run different programming languages. A recommended approach for optimizing support for multiple languages and UIs is to deploy a dedicated compute engine for each language to accommodate various user personas, as illustrated in Figure 4-4.

Dedicated compute per user reverts to our initial challenge: to securely share compute resources among diverse user personas who execute multiple programming languages. The most complex implementation can be considered an architecture where a generic compute engine supports multiple programming language interfaces while enabling multitenancy within the engine, which enables compute resource sharing. For data governance, the implementation warrants isolating users, their code, and the process within the compute engine and blocking access to shared temporary locations where the process writes temporary files while processing datasets.

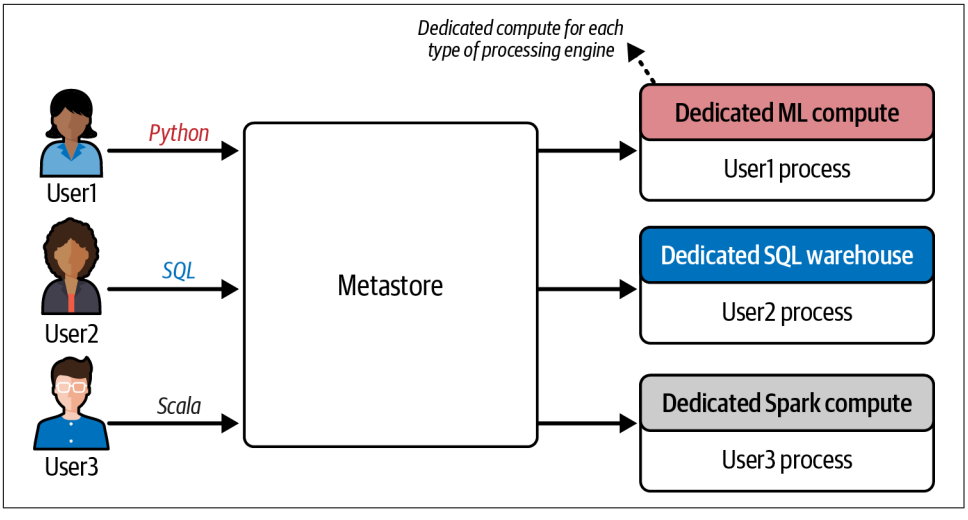


Figure 4-4. Implementing governance supporting multiple languages and compute engines

Figure 4-5 illustrates a similar implementation. In this scenario, multiple users with different roles and needs access and process data from a shared metastore, each using their preferred programming language. For instance, some users leverage Python, while others use Scala, and some rely on more traditional SQL-based access from a data warehouse. The shared metastore acts as a central repository, managing the metadata for all tables and serving the diverse requests from these users, regardless of their programming language of choice. Given that multiple user processes share the same compute resource, isolation becomes a top priority. Each process must be sandboxed to prevent unauthorized access to other users' datasets, ensuring the security and integrity of sensitive data. During processing, most data engines write data to temporary locations. It's essential to protect the integrity and security of this data, as unauthorized access to these temporary paths could compromise sensitive information and expose it to unauthorized users. This compute sharing challenge is what Unity Catalog, along with Lakeguard-powered shared compute architecture, solves for its users, making it an enterprise-grade data governance tool for multiuser Apache Spark clusters.

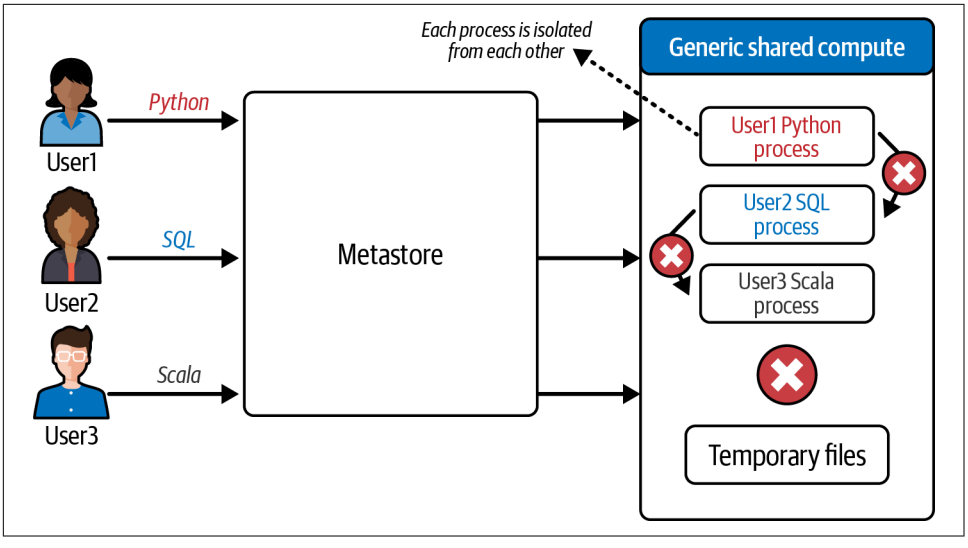


Figure 4-5. Implementing governance supporting multiple languages on shared compute

Nexa Boutique's data platform caters to diverse user personas, including data engineers, data scientists, data analysts, AI engineers, and business users. Within the platform, different teams have distinct preferences for programming languages. For instance, the Demand Planning team's data engineers favor Python for building data ingestion pipelines, whereas the Warehouse Operations team's data engineers prefer Scala. Meanwhile, data analysts rely on SQL to create Power BI dashboards, and data scientists typically use a combination of Python, R, and MATLAB for predictive analytics and ML modeling.

Before onboarding to Databricks and Unity Catalog, application teams used different tools for data governance in their data platform, resulting in broken governance. The fact that users had direct access to files in the data lake, which undermined the table-level access controls, further compounded the problem. Data sharing across domain teams was a cumbersome and inefficient process, as data was often duplicated across multiple applications, each with its own distinct governance framework and toolset. Figure 4-6 depicts the inadequate governance model at Nexa Boutique prior to the implementation of Databricks and Unity Catalog. Specifically, data governance was siloed from other critical assets, such as ML models built using that data, leading to inconsistencies and inefficiencies. In the past, it was a common practice to grant unrestricted access to the underlying data files for tables that were used to train ML models, which posed security and data integrity risks.

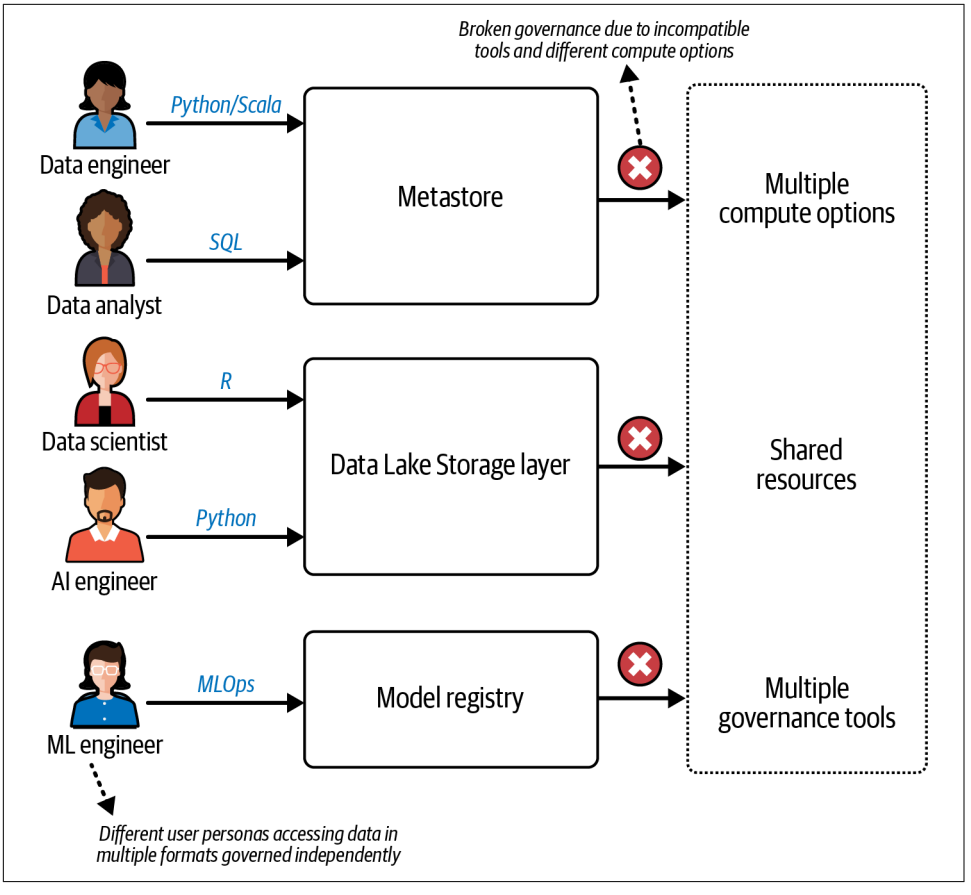


Figure 4-6. Multiple tools and access methods result in a broken governance model at Nexa

Unity Catalog addresses the governance challenge by tackling the two critical components:

- A regional metastore that handles metadata across all objects shared across different compute engines
- Unity Catalog Lakeguard that isolates users and processes to enforce FGACs across the different compute engines and programming languages

Chapter 2 briefly discussed the architecture for these, where we saw how Spark Connect enables user isolation and Lakeguard enables process and code isolation in a Spark cluster. Let's look more in-depth at the compute options in Unity Catalog and understand how you can use the right compute for your workloads.

Classic Compute in Databricks

Databricks uses the classic compute architecture for compute resources provisioned within your cloud account to process data that resides there. This means that when you run Databricks workloads on data stored in your own cloud account, such as Amazon S3, Azure Blob Storage, or GCS, Databricks will automatically deploy and manage classic compute resources, such as clusters, within your cloud account to execute the workload. The classic compute architecture offers flexibility in fine-tuning the infrastructure used for the Spark clusters, from defining the instance type for driver and worker nodes to fine-tuning Spark parameters to optimize workloads for cost and performance. While flexible, the classic compute architecture supports data governance using Unity Catalog and enforces FGAC on data. Unity Catalog works in two modes in classic compute: standard or shared access mode and dedicated or single-user access mode. Figure 4-7 illustrates the classic compute architecture in Databricks, where the compute instances live in the Nexa Boutique cloud account.

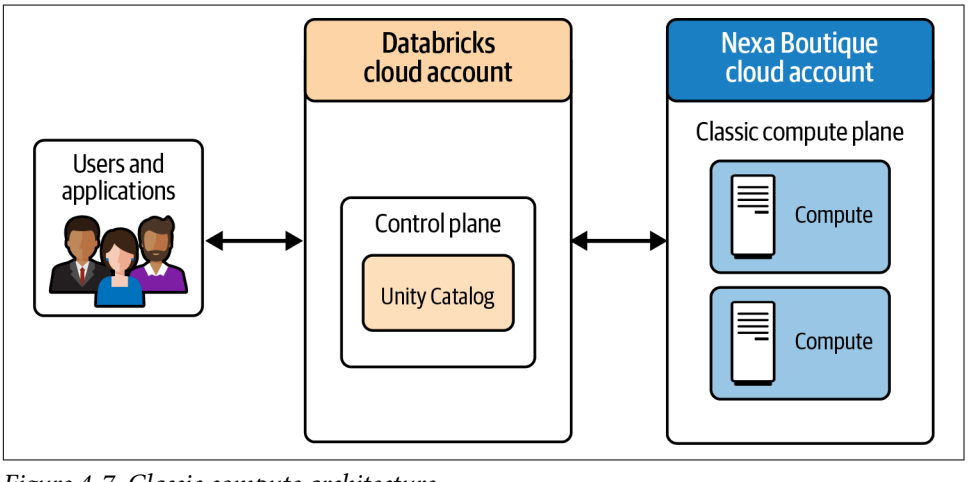


Figure 4-7. Classic compute architecture

Standard or Shared Access

Sharing a compute resource between users with total isolation between users is fundamental when building a governed data platform. Implementing such a system is complex, and the Unity Catalog standard compute is no different. It is intricate, and we appreciate the engineering effort to enable this for end users. It is a bold move to completely rearchitect a data platform's core data-processing engine that also stops supporting some features from the previous architecture. As a result, many legacy features need migration to supported formats in the Unity Catalog standard compute. In Chapter 2, we saw how the user isolation enabled by Spark Connect, built on top of DataFrame APIs, ended support for the RDD APIs in standard compute. Any

process that does not run in isolation or that could break Unity Catalog's governance standards does not work in a standard compute access mode. As of this writing, there are limitations for standard compute, some of which will eventually be fixed over time, while others, such as support for RDD, are unlikely to be implemented.

The standard mode supports end users with multiple programming languages, including these:

- Python
- SQL
- Scala

While creating an all-purpose compute in the Databricks workspace, the end user can choose the access mode under the advanced settings, as shown in Figure 4-8.

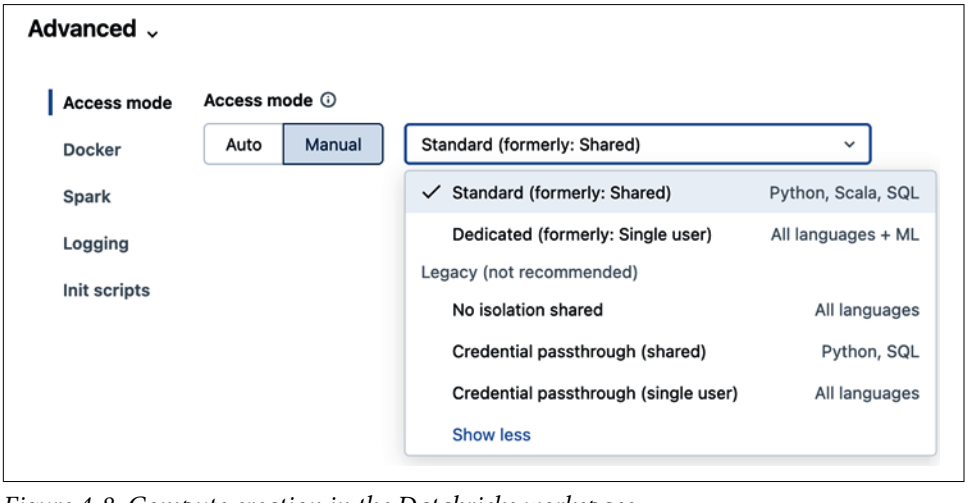


Figure 4-8. Compute creation in the Databricks workspace

You should use standard access mode as your default compute mode when working with Unity Catalog. This mode supports most of your workloads, except distributed ML, which means ML DBR is unavailable in standard access mode. The architecture for a standard compute in Databricks is similar to the multilanguage, multiuser complex architecture discussed previously. However, the implementation involves additional complexities when building governance for a distributed processing engine like Apache Spark. Due to its distributed nature with a driver and several worker node-based implementation, the traditional Spark architecture requires isolation between applications in the driver and the process executed by the executors in the worker nodes, as illustrated in Figure 4-9.

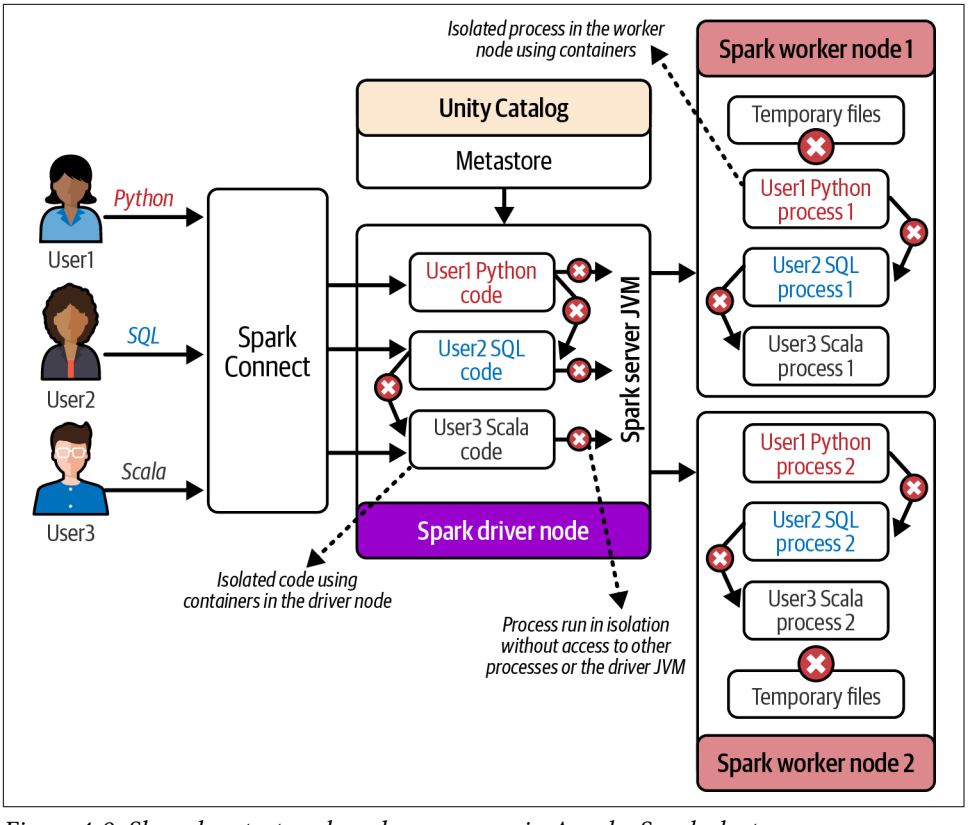


Figure 4-9. Shared metastore-based governance in Apache Spark cluster

Traditionally, Apache Spark was built without strong user isolation and was best fit for a single user or a single application with elevated access to the underlying infrastructure. When shared with different users, unrestricted access can put data security at risk. This shared, unrestricted architecture is what led Databricks to create a more isolated Apache Spark compute that can be shared across different users or applications without compromising data security. The Spark engine and the applications use the same driver node JVM in the traditional Spark architecture. The standard access mode restricts access to the driver JVM for the processes running in the driver node. Individual process and code running in the driver node are isolated from each other and each uses its own JVM. Similarly, in the worker nodes, processes including UDFs run in isolation without access to the executor JVM. Temporary paths where Spark writes intermediate results are also blocked for access from other processes, making it more secure.

Unity Catalog Lakeguard for standard cluster

Apache Spark is an open source data processing engine that powers a wide range of data platforms, from large enterprises to startups. Companies of all sizes leverage open source Spark to build and scale their data infrastructure. The major cloud and data-processing platforms provide multiple flavors of managed Spark. But why is the standard access mode Spark cluster superior to any other offerings, and what makes it exceptional?

Unity Catalog's standard access mode Spark compute engine provides complete isolation between its users, their code, and processes across the driver and executor nodes, enabled by Lakeguard. In Chapter 2, we saw how a combination of Spark Connect and containerization techniques enabled isolation and improved data governance in the standard cluster. We will look at real-life examples of how the standard access mode works.

Nexa Boutique prioritizes customer feedback and product reviews, responding promptly to ensure customer satisfaction and minimize churn risk. The company harnesses the power of sentiment analysis ML models to uncover valuable insights from customer product feedback. It leverages these insights to drive targeted internal process improvements, specifically addressing areas where customers have expressed negative sentiment.

The marketing team proactively engages with customers to address and resolve their concerns. The *customer* table is secured with FGACs, allowing the marketing team to view customer email and address details while keeping this information hidden from all other users. Lakeguard empowers the marketing team to query the table in isolation while simultaneously restricting access to unauthorized users and ensuring robust data security within a standard cluster.

At Nexa, marketing and demand planning teams have access to the same customer data in the Unity Catalog *customer* table. However, they have different levels of access granted on this table. The marketing team uses SQL to query the *customer* table, while the demand planning team uses PySpark to read the same data in a Databricks notebook.

But here's the thing: the demand planning team should not see the customer's email and address, so Nexa administrators use a Unity Catalog masking function to mask this information. This Unity Catalog function is like a filter that hides sensitive information from the demand planning team or any user, not from the marketing team. The Unity Catalog function to mask the email column for nonmarketing users can be implemented as shown here:

```
CREATE FUNCTION email_mask(email STRING) RETURN CASE
 WHEN is_member('marketing-team') THEN email
 ELSE 'REDACTED'
END;

ALTER TABLE
 customer
ALTER COLUMN
 email
SET
 MASK email_mask;
```

The masking function is enforced on the data in real-time, based on the user accessing the *customer* table. So, when the demand planning team user runs their PySpark notebook, the compute cluster retrieves the relevant files from cloud storage, applies the filtering and masking, and delivers the masked data. Even though the demand planning team user is not supposed to see the unmasked email and address columns, the file fetching process still needs to retrieve these columns. But don't worry, Lakeguard's containerization techniques ensure the data is processed in isolation within a container that runs as a nonroot user. The demand planning team user will see only the masked email and address columns.

On the other hand, the marketing team's SQL query fetches the relevant files in the same cluster but in a separate container isolated from other processes. Since the marketing team can access the unmasked data, the Unity Catalog function doesn't apply any additional masking. After applying the relevant filtering, Lakeguard delivers the resulting dataset to the marketing team user in plain text.

Lakeguard's isolation mechanisms ensure that the marketing team's process and the demand planning team's process remain separate, with no access to each other's data or temporary Spark output paths. This guarantees that the demand planning team's process cannot access the unmasked email and address information from the marketing team's data, thereby enforcing strict data governance and access controls. The whole process is illustrated in Figure 4-10.

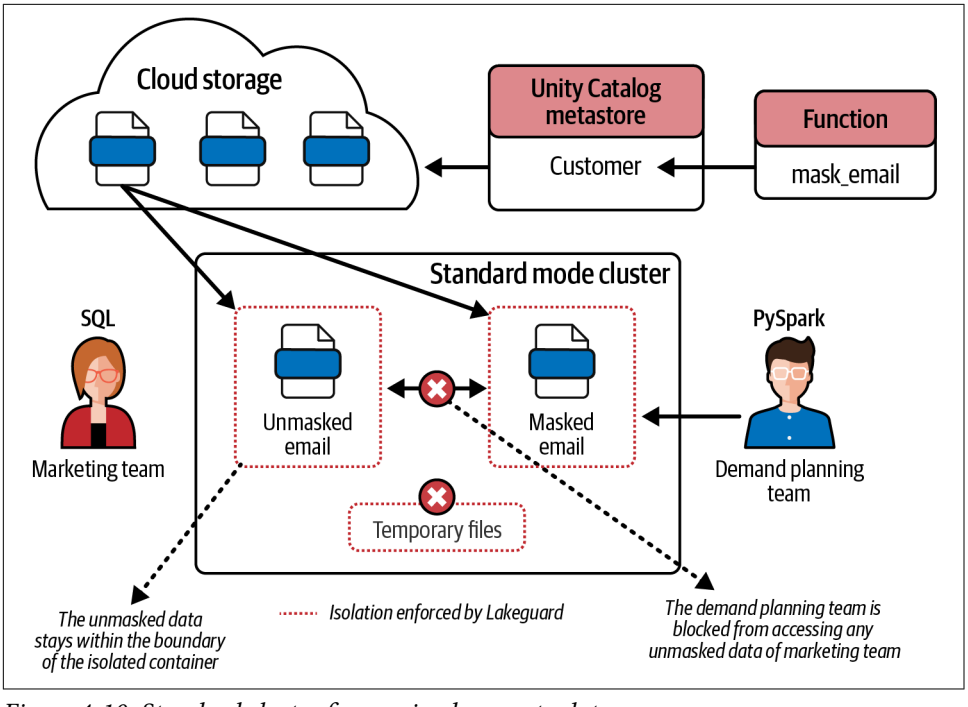


Figure 4-10. Standard cluster fine-grained access to data

Unity Catalog guardrails and service credentials

When it comes to Unity Catalog compute, there is a question that the data architects at Nexa had about bypassing Unity Catalog access controls using credentials external to Unity Catalog. In a Databricks on AWS workspace, you can utilize an instance profile with the required access via an IAM policy to read data stored in an S3 bucket. You can also use this instance profile to integrate with other AWS services, such as Simple Notification Service (SNS), and leverage the AWS software development kit (SDK) for Python (Boto3) for extended AWS capabilities. If users could circumvent the guardrails of Unity Catalog by using credentials external to Unity Catalog, the consequences would be catastrophic. For example, a user authorized to use a Spark cluster may be able to read data from the storage layer using an external credential that is not managed by Unity Catalog, potentially allowing them to access unauthorized data. To prevent this, Unity Catalog limits access to external locations registered with Unity Catalog to only those storage credentials managed within Unity Catalog. Specifically, in a Databricks on AWS workspace, you can register an IAM role as a Unity Catalog storage credential. Once created, you can use this storage credential to register an S3 bucket as an external location in Unity Catalog. However, if you try to access the external location without the necessary `READ FILES` permission, Unity Catalog will block access. This is true even if you use an external instance profile with

broader read and write access to the same S3 bucket. In other words, Unity Catalog's access controls take precedence over any external permissions you may have.

If you look at the network and file system access limitations and requirements for Unity Catalog shared access mode, it states, "You cannot connect to the instance metadata service (IMDS), other EC2 instances, or any other services running in the Databricks VPC." When an instance profile is attached to an EC2 instance, a temporary credential can be fetched by calling the EC2 metadata service via the API (*http://169.254.169.254/latest/meta-data/iam/security-credentials/<iam_role_name>/*).

Unity Catalog doesn't provide FGAC for AWS instance profiles attached directly to a cluster. Hence, instance profile-based authentication is not allowed in standard access mode, as Figure 4-11 depicts.

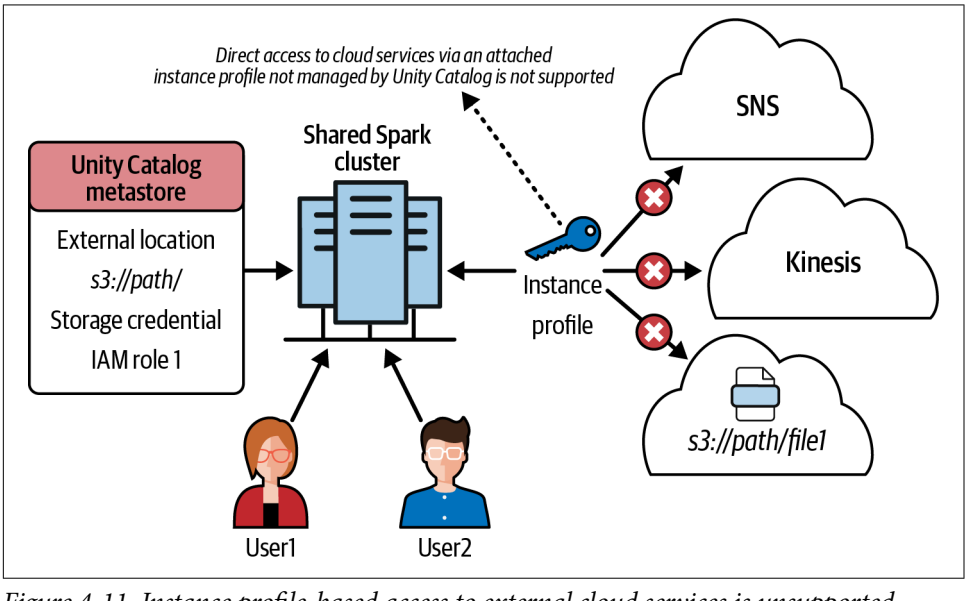


Figure 4-11. Instance profile-based access to external cloud services is unsupported

The Databricks on AWS workspace at Nexa used Amazon SNS to send sales campaign data to customers. However, Unity Catalog's standard access mode clusters do not support this approach. Furthermore, Unity Catalog does not support Declarative Pipelines that use instance profiles for authentication with Amazon Kinesis, which brings transactional data from Nexa's point of sale (POS) systems. Cloud administrators initially provisioned long-lived access tokens to facilitate authentication in Unity Catalog's standard access mode clusters, storing them in Databricks secrets. Although this temporary workaround was adequate, a more elegant solution was required.

Definition: Lakeflow Declarative Pipelines

Lakeflow Declarative Pipelines (formerly known as Delta Live Tables) is a declarative framework for building data-processing workloads in Databricks. Declarative Pipelines can automatically handle orchestration, dependencies, data quality, and monitoring, helping you concentrate on building business transformations.

Unity Catalog introduced service credentials to provide a Unity Catalog-governed credential for accessing external cloud services, analogous to storage credentials. Service credentials create a more streamlined and secure connection by elevating an instance profile or MI at the cluster level to a central Unity Catalog metastore. Users sharing a standard access mode cluster can be individually granted permission to utilize service credentials, thereby alleviating data governance concerns.

Service credentials enabled the reactivation of SNS and Kinesis connectivity from Unity Catalog standard mode clusters, eliminating the need for long-lived credentials. Figure 4-12 illustrates the service credential implementation that enables secure connectivity to external cloud services.

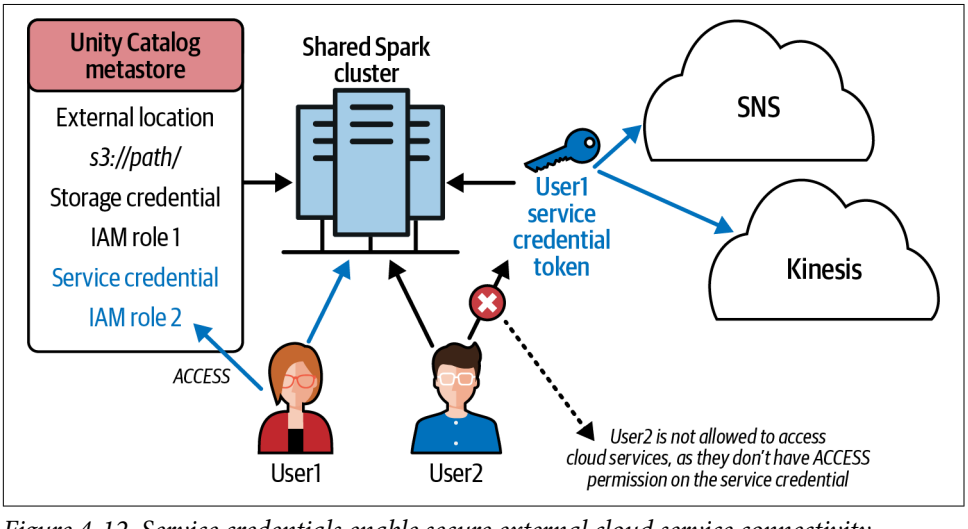


Figure 4-12. Service credentials enable secure external cloud service connectivity

Figure 4-12 illustrates that only User1 has permission to use the service credential registered in Unity Catalog. In contrast, User2 is unauthorized to use the service credential and, therefore, cannot access the AWS SNS or Kinesis service. After registering an IAM role as a service credential in Unity Catalog, you can create a `boto3` session using the service credentials to connect to the SNS service, as demonstrated in the following code, provided your administrator has granted you the `ACCESS` privilege on the service credential:

```
import boto3
credential = dbutils.credentials
   .getServiceCredentialsProvider('your-service-credential')
boto3_session = boto3.Session(
   botocore_session= credential,
  region_name='your-aws-region'
   )
sm = boto3_session.client('sns')
```

Similarly, in Azure Databricks, you could leverage an MI registered as a service credential to authenticate while using the Azure SDK, as shown in the following code:

```
from azure.keyvault.secrets import SecretClient # example Azure SDK client
credential = dbutils.credentials
   .getServiceCredentialsProvider('your-service-credential')
vault_url = "https://your-keyvault-name.vault.azure.net/"
client = SecretClient(vault_url=vault_url, credential=credential)
```

Limitations of standard access mode

Owing to the revised architecture and, in general, abiding by the governance standard set by Unity Catalog, the standard access mode comes with certain limitations compared to an HMS-based Databricks compute. As of this writing, these limitations exist, and since Databricks is continuously improving Unity Catalog, some of these limitations will eventually be resolved.

Dedicated or Single User Access

Dedicated access mode uses the traditional Apache Spark architecture, where the user assigned to the compute node acquires elevated access privileges. In Databricks, the dedicated mode supports ML runtimes and runs most ML and AI workloads. In Chapter 2, we saw the architecture of a dedicated mode cluster and how the serverless layer enables FGAC for tables and views. No Spark Connect and container-based isolation are available in the dedicated mode and are accessible by only the user assigned to the compute resource. The dedicated access mode is similar to the most straightforward implementation of isolated governance we discussed, except that the Databricks implementation based on Apache Spark supports multiple programming languages in the same compute engine. At Nexa, the administrators explicitly designated dedicated mode clusters for ML and AI workloads and data scientists working in the R programming language. The dedicated mode supports more languages and features than the standard access mode, including the following:

- Python
- Scala
- R

- SQL
- Machine learning Databricks runtime (MLR)
- Databricks container service (DCS)

The dedicated access mode architecture provides elevated access to the process running within the compute, necessitating more stringent data governance requirements. One notable issue that arises from this is the overfetching of files when querying views or tables with FGAC, as discussed in Chapter 2. To illustrate this, let's revisit the *customer* table example, where marketing teams have been granted access to view customer PII information while others have restricted access. We can now examine how this works in a dedicated access mode compute.

When a marketing team user submits a query on the *customer* table, the system retrieves the corresponding files from cloud storage using the temporary credentials issued by Unity Catalog. Figure 4-13 represents the cloud storage location for the *customer* table in Unity Catalog, where the physical files are written in a Delta format. Per the query executed by the marketing team user, the files `File1.parquet and File2.parquet` will be fetched from the cloud storage for further processing by the compute engine:

```
Select
 *
from
 customer
where
 customer_id = 1011
```

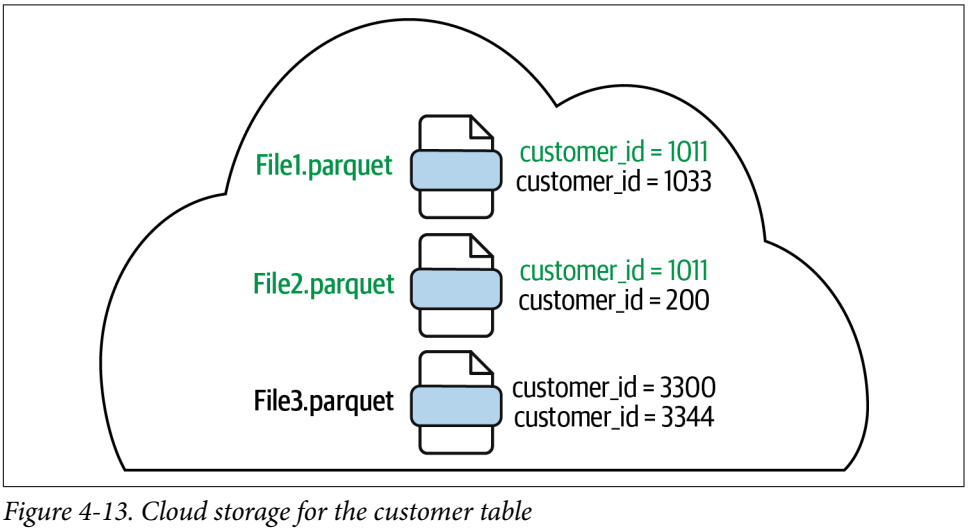


Figure 4-13. Cloud storage for the customer table

The dedicated compute then processes the files, filtering out unwanted data. Once the data is filtered, the system returns the query results to the user. This filtering works well for a user who has full read rights on the table. No masking rules are applied, and because the user has full read rights on the table, any accidental access to nonfiltered data via temporary paths where Spark writes data during processing does not affect data governance.

When a demand planning team user submits the same query on the customer table, the system retrieves the required files, `File1.parquet` and `File2.parquet`, from cloud storage. It transfers them to a dedicated compute environment. The system then filters out unnecessary data to generate the requested information. However, since the demand planning team user lacks full read access to the table, the system applies data masking rules to protect sensitive PII. Specifically, the system filters the data in plain text, masks the PII data, and then serves the filtered and masked data to the demand planning team user, ensuring that sensitive information remains protected.

In dedicated access mode, the process operates with elevated privileges, posing the potential risk of exposing sensitive, unmasked data to users, as depicted in Figure 4-14. Since files are retrieved from cloud storage without masking, the combination of elevated privileges and unmasked data increases the risk of unauthorized data exposure. Dedicated access mode mitigates this risk by restricting user queries to tables without FGAC.

Additionally, dynamic views, streaming tables, and views are also limited to being accessed from a dedicated mode compute engine.

Views are created on a table, providing a limited or aggregated data representation. Since views are materialized at runtime by the compute engine, the underlying files of the table are required to generate the data. Since files are written as contiguous blocks, the system may transfer columns that are inaccessible to the user to the dedicated compute engine, potentially exposing sensitive data. Access to a view is permitted if the user has read access to the underlying tables the view refers to. Limiting access to tables and views with FGAC enabled in dedicated compute mode reduces the risk of exposing unauthorized data. Still, it also restricts data access for users, limiting their ability to query and analyze the data.

Definition: Streaming Table

A streaming table is a Delta table that supports incremental data processing, which is ideal for handling continuously growing datasets. It's perfect for pipelines that require fresh data and low latency and is also helpful for large-scale transformations where results can be updated incrementally as new data arrives.

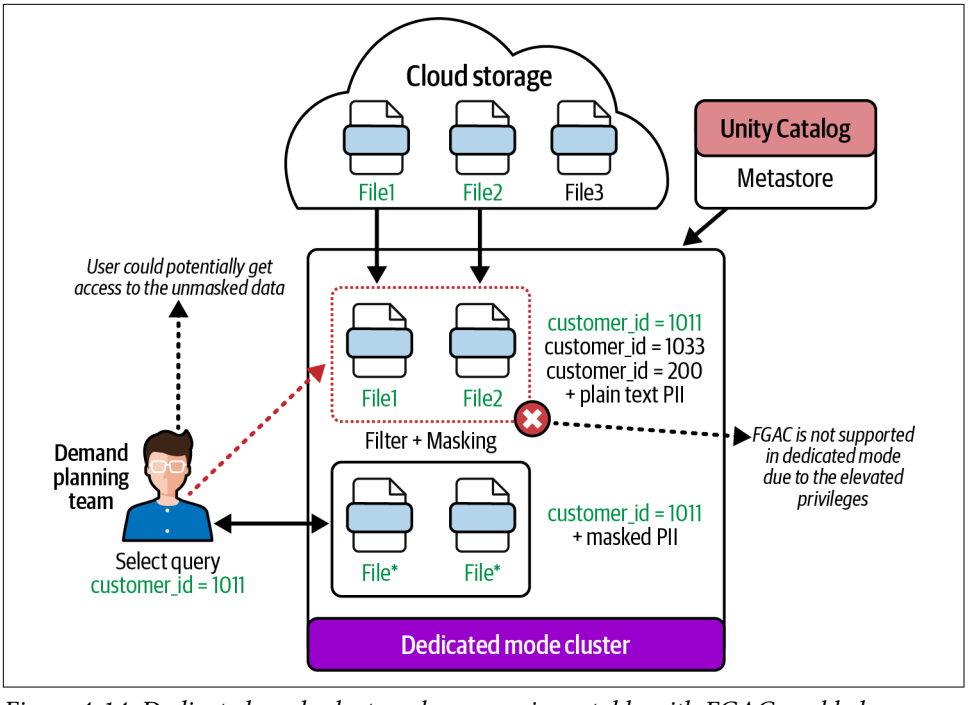


Figure 4-14. Dedicated mode cluster when querying a table with FGAC enabled

The limitation of querying tables without any FGAC is present only if you run Databricks dedicated mode compute with a runtime below version 15.4 LTS. By implementing a serverless filtering fleet, Lakeguard enables FGAC on dedicated mode compute for Databricks runtime version 15.4 LTS and above.

Unity Catalog Lakeguard for dedicated cluster

At Nexa, most data tables had either row filters or column masking applied. However, the dedicated compute access mode lacked FGAC, rendering tables with FGAC inaccessible from dedicated compute mode clusters. Due to limitations of the standard cluster in supporting Databricks MLR, all ML workloads at Nexa ran on dedicated access mode compute clusters. However, analytics on tables with FGAC were impossible until Lakeguard enabled FGAC in a dedicated compute mode cluster. Lakeguard utilizes a filtering fleet that runs in the managed data plane owned by Databricks using serverless compute, introduced in Databricks runtime version 15.4 LTS, to enable FGAC in dedicated compute mode clusters. The serverless filtering fleet filters data read from cloud storage, removing any unauthorized information. It then loads the filtered data to the dedicated compute and presents it to the consumer.

The Lakeguard filtering fleet service fixes the problem of over-fetching files while querying tables from a dedicated access mode cluster. The serverless filtering fleet is created in the managed data plane and is inaccessible to the user. Any files read from the cloud storage are first loaded into the serverless filtering fleet. All data filtering, including row-level filtering and column masking, occurs within the serverless layer. Additionally, view materialization is also applied within this layer, ensuring that only the relevant data is processed and returned while maintaining the security and access controls defined by the filters and masking rules. The final dataset that satisfies the user's access grants and filter conditions is then loaded back into the dedicated compute layer and served to the end user. Lakeguard enables FGAC on multiple objects in a dedicated access mode compute, including the following:

- Views
- Materialized views
- Streaming tables
- Dynamic views
- Tables with row filtering functions applied
- Tables with column masking functions applied

Definition: Materialized View

A materialized view is a pre-computed view that's refreshed on a schedule. Databricks recommends it for tasks that involve complex data processing, such as data transformations, aggregations, and optimizing slow queries or frequently used calculations.

Definition: Dynamic View

A view that can implement FGAC by dynamically filtering data based on certain attributes of the user or their identity. Unlike a static view, which often serves fixed data, dynamic views can filter or modify the data during runtime based on the user's role.

Let's analyze the same select query by a Demand Planning team user on the customer table. The required files, `File1.parquet` and `File2.parquet`, are loaded into an intermediate serverless filtering fleet compute environment managed by Databricks. This environment is not directly accessible to users, ensuring that filtering and access controls are enforced on the data.

The filtered and masked data that satisfies the user's query is then loaded into a dedicated mode cluster, which returns the results to the end user. By isolating the data filtering and masking process from the elevated access dedicated mode cluster

and from the user itself, Lakeguard enables FGAC on the dedicated mode cluster, unlocking multiple use cases at Nexa. Figure 4-15 illustrates the operation of the filtering fleet in a Unity Catalog-dedicated access mode compute environment.

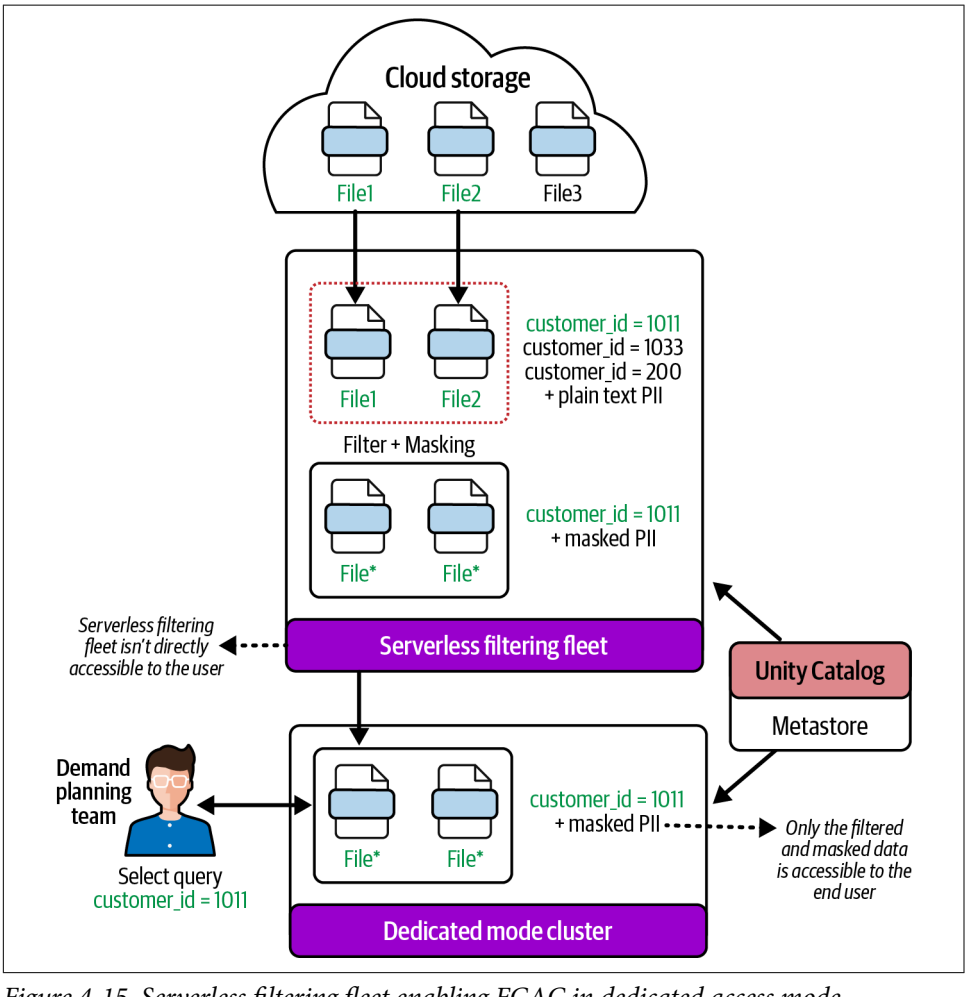


Figure 4-15. Serverless filtering fleet enabling FGAC in dedicated access mode

Limitations of dedicated access mode

The dedicated access mode is recommended only for running workloads unsupported in standard access mode. The dedicated mode has limitations due to its elevated access to the underlying compute. You are advised to check Databricks latest documentation on the supported functionalities and understand the limitations before you build your data and AI applications against a dedicated access mode compute in Unity Catalog.

Assigned to Group Cluster

Nexa's critical workloads, which include distributed ML and GPU-dependent applications, require dedicated access mode compute. Moreover, some of its legacy frameworks that utilize RDD and DBFS also need it. As a result, the Nexa platform administrators assign most of its data scientists and ML engineers their own dedicated access mode clusters that run the Databricks MLR. This approach can lead to scalability and cost-effectiveness issues when creating and maintaining individual clusters for each user.

To address this challenge, Databricks introduced assigned-to-group clusters, which enable teams to share dedicated compute resources via a group. Databricks admins can synchronize these groups with a Databricks account from an identity provider or create and manage them directly in the workspace. For more comprehensive coverage of identity management, refer to Chapter 3, which provides an in-depth examination of this topic. The dedicated access mode compute can be assigned to a group, allowing all users to share the compute resource while benefiting from all the features of a dedicated access mode cluster.

Within Nexa's Demand Planning team, a community of data scientists leverages data analysis and predictive modeling to forecast future demand. Utilizing time series analytics, specifically ARIMA (AutoRegressive Integrated Moving Average), they examine historical demand data to identify patterns and trends that forecast predictions about future demand. By analyzing these forecasts, the team can better understand the potential impact on the supply chain and make data-driven decisions to optimize inventory management and logistics.

Each data scientist in the team was initially assigned a dedicated compute mode cluster to develop their ML models, which resulted in significant costs and infrastructure maintenance overhead for the Nexa platform administrators. To reduce the total cost of ownership (TCO), the administrators created a new Microsoft Entra group for data scientists and assigned it to a dedicated mode compute cluster. With assigned-to-group clusters, data scientists in the designated group can share the dedicated compute resource, improving cluster utilization and reducing TCO.

From an architectural perspective, the assigned-to-group cluster uses the same design as a dedicated compute mode cluster. The permission model is crucial to understand in this context, as it behaves differently in assigned-to-group clusters. Users who are part of the group might have different levels of access to data in Unity Catalog, and these individual permissions are enforced when they consume data from a standard or dedicated access mode compute assigned to them as an individual user.

In contrast, assigned-to-group clusters ignore individual user grants and permissions. Instead, the group's grants and permissions are used, giving all users the same level of access to data, regardless of their individual permissions. When sharing a dedicated

access mode compute among users via a group, administrators and data owners must grant the group access to any required objects managed by Unity Catalog to run the workloads. Figure 4-16 lists the individual data scientist's access to tables in Unity Catalog and how access controls work when these data scientists access a dedicated compute via a group membership `NB_DP_Data_Scientist`.

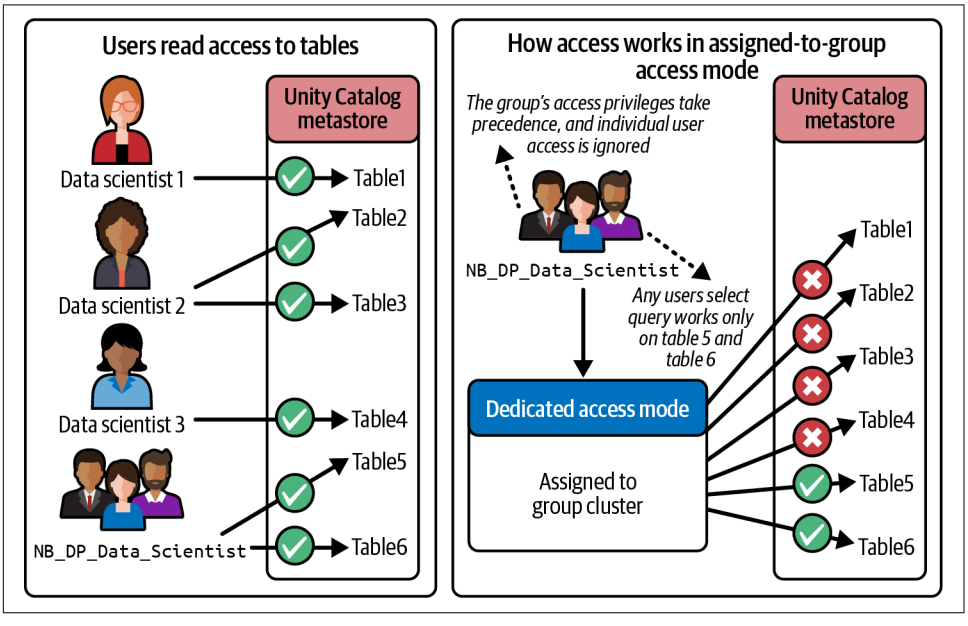


Figure 4-16. Table access for data scientists at Nexa Demand Planning team who are part of the `NB_DP_Data_Scientist` group

SparkML is a widely used, CPU-based distributed ML library. One of the primary limitations of Spark-ML has been its dependence on the RDD API. As a result, its deployment is restricted to Unity Catalog dedicated clusters or requires an assigned-to-group cluster. This limited its flexibility and usability in a standard access mode compute that can be shared with multiple users. However, with the Apache Spark 4 release, the open source community is introducing support for SparkML on Spark Connect. This development enables SparkML to be available on standard clusters and, in the future, on serverless environments as well. It requires you to be in the Databricks runtime with DBR 17 or higher.

Going Serverless with Databricks

Classic compute allows you to host Databricks compute infrastructure directly in your own cloud provider's account, giving you complete ownership and control over the resources utilized by Databricks to process your workloads. You can bring your own virtual networks (VPC/VNET) from your cloud account, where Databricks compute resources are deployed. Classic compute offers flexibility for you to tailor your Spark clusters to meet specific needs, allowing you to select your preferred VM types and Databricks runtime versions for optimal workload performance. If you're accustomed to optimizing your Spark workloads by tweaking Spark configurations, classic compute supports this level of customization. Since the infrastructure for your compute lives in your own cloud provider account, there is a natural isolation of resources between users. On top of that, you can further configure your network settings to use the cloud provider's private link support and enable firewalls for your cloud storage, further enhancing security.

Definition: VPC/VNET

A virtual private cloud (VPC) or virtual network (VNET) is a virtualized network infrastructure that provides a secure and isolated environment for cloud resources. VPC/VNET allows users to define their own IP address range, subnets, and network configurations, offering a high level of control and security. By providing an isolated environment, users can protect their cloud resources from unauthorized access.

Despite all these capabilities, classic compute has certain limitations, the biggest of which is the cluster startup time. Any classic compute creation request could take between 2 and 10+ minutes, depending on the region and availability of the specific VM requested. The majority of the time taken is attributed to the delay in provisioning VMs by the cloud provider, and the rest is for setting up the software. For short-running jobs, startup time can be an overhead when you have several of them. The longer startup time will also add up in the TCO, as you will be charged for the compute resources by your cloud provider during the startup time. Apart from the startup time, you must consider capacity planning and reservation for resources within your cloud provider. Setting up your own network and VPC/VNET, along with subnets and IP address allocation, is another overhead for the platform administration team. As your business grows and you onboard new workloads, scaling your infrastructure to meet increasing demands is hindered by the cloud provider's imposed quotas and limits on their services. You also need to consider Databricks runtime version upgrades and always keep an eye on upgrading to supported versions when the current one is deprecated. But what if you can eliminate all this management overhead and have Databricks manage your infrastructure for you

automatically? That is where serverless computing infrastructure comes in and solves several issues, accelerating your progress in innovating with data.

Building a serverless architecture is hugely complex. If you have experience setting up Apache Hadoop or Spark clusters on premises, you know the intricacies involved. Now, imagine achieving the same level of complexity in a fully automated serverless environment—that's exactly what is accomplished with serverless Spark clusters that are provisioned by serverless compute in mere seconds. Databricks serverless launches millions of VMs daily, all governed by the access controls provided by Unity Catalog, demonstrating the scale at which data governance operates using Unity Catalog.

Two primary factors influence the startup time for classic compute environments:

- The time required to provision infrastructure resources like VMs from the cloud provider
- The time spent on installing and configuring the necessary packages and dependencies within the infrastructure

The serverless compute pool addresses the first factor by eliminating the need for cold starts of VMs whenever a workload requests compute resources. Instead, Databricks maintains a pool of pre-warmed VMs that are always ready to serve workloads, significantly reducing the startup time. The serverless infrastructure is deployed within the Databricks serverless compute plane, which enables faster startup times by leveraging pre-loaded packages and dependencies. These pre-loaded resources are made available to users on demand, instantly allowing for efficient and rapid execution of workloads. Major serverless product offerings widely adopt this architectural pattern in the cloud computing landscape, providing a robust foundation for secure and isolated workloads. Integrating all available network security options ensures that each user's workload is executed in a dedicated and isolated environment, minimizing the risk of unauthorized access or data breaches.

The second factor involves installing and configuring the libraries and dependencies required to run the Spark cluster. These include the Databricks runtime image and other utilities that help monitor resources. The serverless infrastructure uses a custom operating system, container images, and other optimizations to ensure faster compute availability to end users. The Databricks engineering blog *Booting Databricks VMs 7x Faster for Serverless* talks in-depth about the optimizations built into serverless to achieve a faster startup time for the compute. Combining these implementations has resulted in a serverless infrastructure availability of under 30 seconds for most workloads in Databricks.

In Databricks, serverless compute infrastructure spans multiple product areas, including the following:

- Serverless generic compute
- Serverless data warehouse
- Serverless model serving
- Serverless Databricks Apps
- Serverless Lakeflow Declarative Pipelines

The range of serverless compute options in Databricks is continually expanding, and we anticipate that Databricks will eventually transition from its current PaaS model to a SaaS offering. Let's look deeper at the current serverless offerings, their architecture, and how they integrate with Unity Catalog to give you the best data and AI governance.

Serverless Generic Compute

Imagine waiting six months to get a new Hadoop cluster up and running in your on-premises data center. That's what the team at Nexa used to face. The process was long, from getting approvals to procuring hardware and configuring the system, and every step was a hurdle that slowed down innovation. But what if you could flip that script? With serverless compute, you can. You can have the infrastructure you need to build your application in seconds. No more waiting, no more hassle. Serverless offers you instant access to the compute and infrastructure you need to bring your ideas to life.

Serverless generic compute in Databricks allows you to run interactive workloads, scheduled jobs, and workflows to orchestrate the data and AI tasks within Databricks. Serverless generic compute is architecturally different in its implementation from an infrastructure perspective but shares the same backend as a standard access mode compute in Unity Catalog. Using the standard access mode allows for enforcing Unity Catalog's access controls in a completely isolated way, leveraging the power of Spark Connect. Serverless for Databricks jobs and workflows enables you to run interactive workloads from a Databricks Notebook or IDE and scheduled job runs that utilize the Databricks workflow orchestration tool. Figure 4-17 depicts how serverless compute works in Databricks generic compute, utilizing the serverless compute pool and Spark Connect-based Unity Catalog standard access mode.

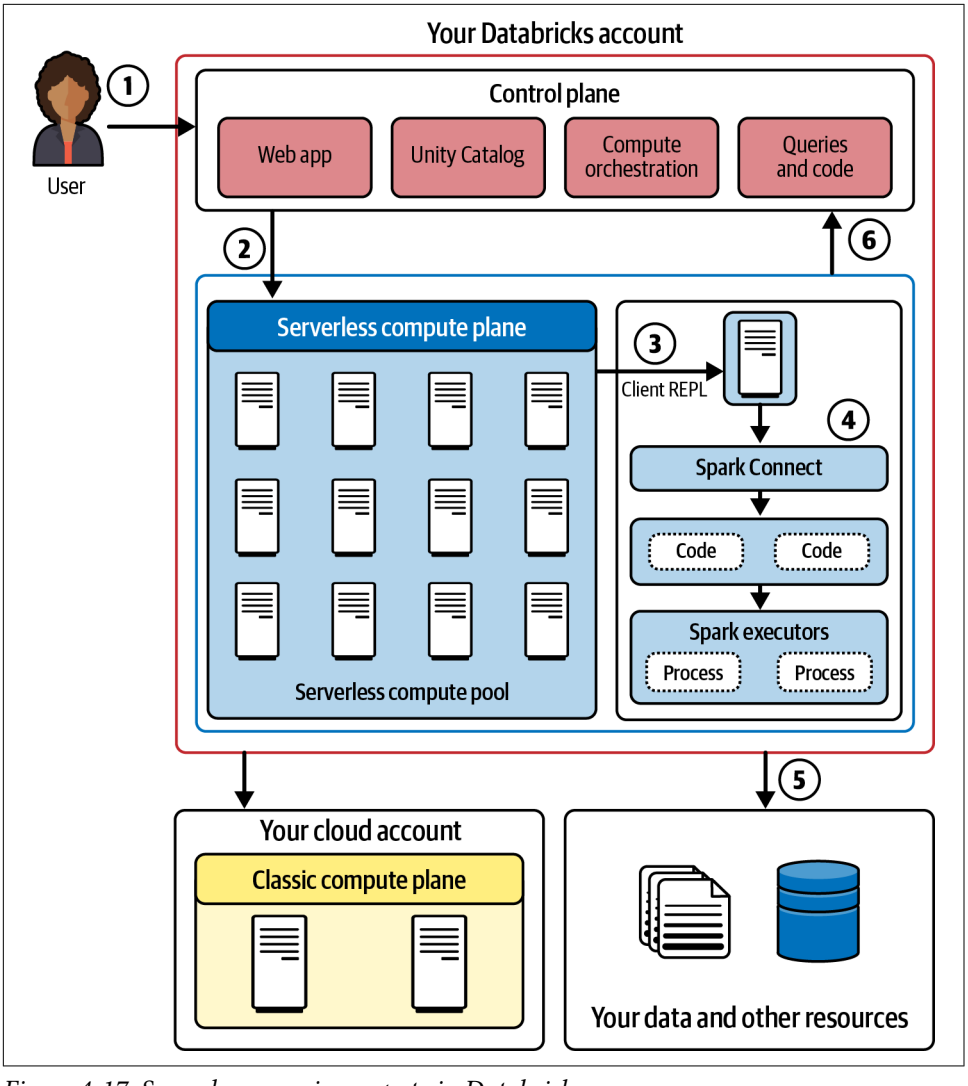


Figure 4-17. Serverless generic compute in Databricks

When a user requests a serverless compute, the operation proceeds through the following steps:

1. The user submits a request to connect to a serverless compute from the Databricks workspace. As of this book's writing, all requests between the user, the control plane, and the compute plane are encrypted using Transport Layer Security (TLS) 1.2.

2. The control plane then contacts a pool of VMs maintained by Databricks to execute the job. This pool is unallocated and contains freshly created VMs without any credentials associated with them.
3. In the first instance, a client Read-Eval-Print Loop (REPL) VM is allocated to evaluate the user request.
4. The client REPL then connects to a Spark cluster via Spark Connect and uses Unity Catalog to generate temporary credentials to access data. The workload runs in its own private network without any public IP address allocated. Moreover, workloads running in serverless compute cannot communicate with each other. As of this writing, all attached storage to the compute is encrypted using AES-256.
5. To read any data, the compute connects to the customer data plane, where it reads the cloud storage using the short-lived credential generated by Unity Catalog.
6. The processed data is then returned to the user. Once the workload is completed, all associated storage and compute resources are securely wiped and not reused for other workloads.

Serverless compute is fully integrated with Unity Catalog and enforces access controls consistently to ensure secure access to data. At Nexa, all short-running workloads embraced serverless, helping it reduce the overall TCO by avoiding the costly startup times associated with classic compute. Serverless, being version-less, also meant the data teams within Nexa could start using new features that Databricks releases without going through a Databricks runtime version upgrade process.

Serverless Data Warehouse

A data warehouse lets you execute SQL queries against data stored in your cloud storage. The serverless generic compute is a more complex iteration of the serverless product portfolio within the Databricks ecosystem. Compared to its predecessors, like the serverless data warehouse, the serverless generic compute needed a multilanguage, multiuser, and isolated setup similar to the standard access mode compute in Unity Catalog, making it more complex to implement.

On the other hand, the serverless data warehouse supports a SQL-based interface that utilizes the serverless pool but without the Spark Connect-based architecture. A Databricks SQL warehouse process does not require elevated access to the compute resources, unlike the distributed ML workloads that necessitate root access to operating system-level files. It follows the in-depth network isolation that enables all of the serverless workloads in Databricks, including the following:

- Using dedicated VMs
- Network isolation with no public IPs
- Encryption at rest and in transit using AES-256 and TLS 1.2 (as of this writing)
- Application of the principle of least privilege

Serverless warehouses are available in a range of sizes, from 2X-Small to 4X-Large, each with a varying number of VMs for worker nodes. The smallest size, 2X-Small, has just 1 VM, while the largest size, 4X-Large, has up to 256 VMs. The key benefit of serverless data warehouses is their ability to rapidly scale up to improve workload performance and quickly scale down to reduce costs when utilization is low, making them an ideal choice.

Like serverless generic compute, serverless warehouses are also created from a pool of pre-warmed VMs. To ensure security and isolation, each warehouse is network-isolated and containerized, preventing cross-network access between customers.

Additionally, data access is carefully controlled, allowing data to be retrieved from customer cloud storage via the private network of the cloud backend without ever being exposed to the public internet. Unity Catalog strictly governs all data access and follows the principle of least privilege, which utilizes downscoped credentials issued by Unity Catalog to grant specific users access to specific objects, as authorized by Unity Catalog.

At Nexa, serverless data warehouses serve all of its BI reporting due to their instant availability and performance improvements over the classic warehouse.

Figure 4-18 illustrates the serverless data warehouse created from the serverless compute pool for three end users. User 1 has two medium warehouses, while User 2 has one large warehouse. User 3 has two small and one medium warehouse, all deployed in their own isolated containers with cross-network access prevented.

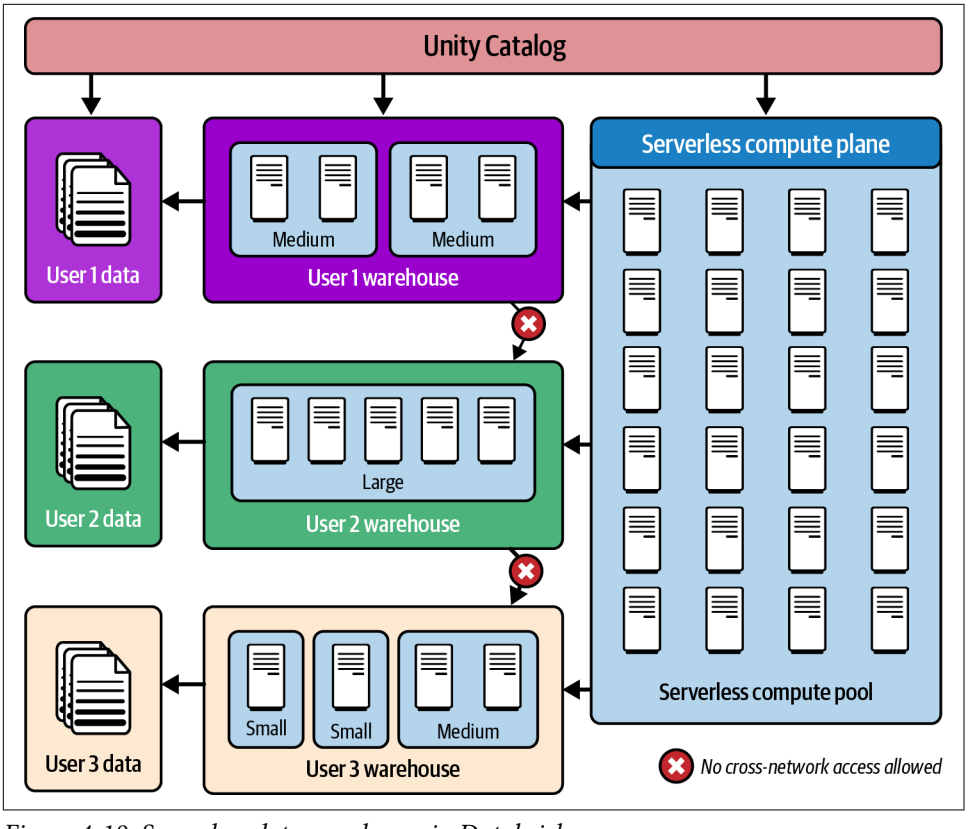


Figure 4-18. Serverless data warehouse in Databricks

Serverless Model Serving

Mosaic AI model serving is a Databricks offering that allows you to deploy, govern, and query your ML and AI models in real-time and batch mode. It uses the serverless infrastructure to host the models as a REST API endpoint that you can integrate and invoke from your applications. Unity Catalog governs the access controls for model artifacts and feature tables deployed in Unity Catalog. Like other serverless offerings, models deployed in a serverless environment are isolated, and Unity Catalog controls data access. The model deployment process retrieves model artifacts from a Unity Catalog-governed storage location within your cloud storage. Additionally, any external libraries the model requires are pulled from public repositories to generate the model image, which is always encrypted. While users deploy a model, compute resources are allocated from the serverless pool that installs the image needed for the model.

Serverless model serving comprises two parts:

- Generating and deploying the model image from Unity Catalog and external repositories
- Serving the model endpoint for incoming requests

Figure 4-19 illustrates the first part, where the model artifact is pulled from Unity Catalog and installed on serverless compute.

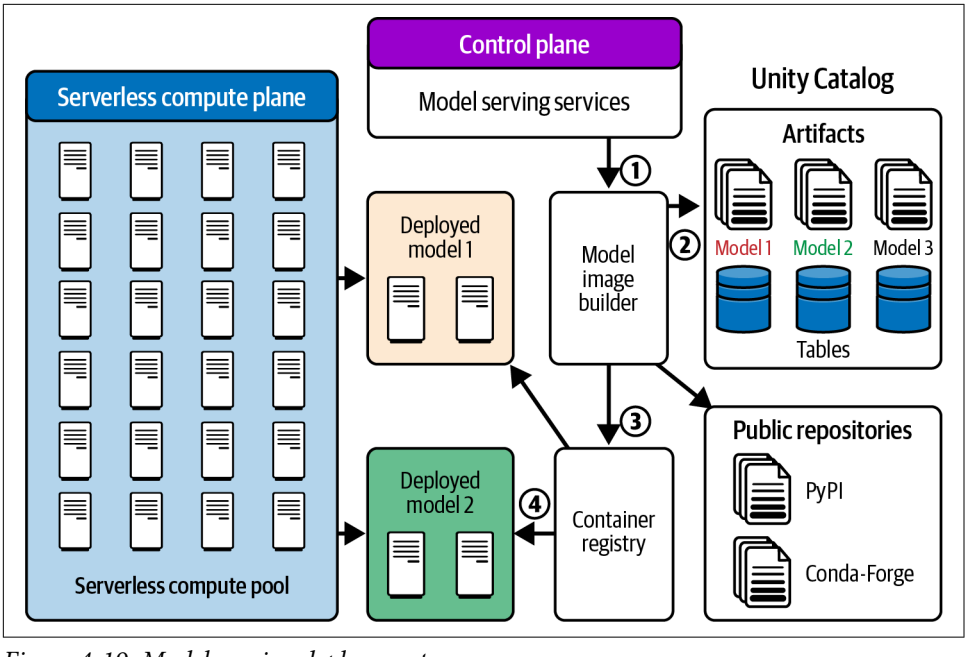


Figure 4-19. Model serving deployment

Here is how it works:

1. The user submits a request to the control plane to deploy a registered model in Unity Catalog. After the access controls are validated, a model-serving service initiates the process of downloading the required model artifacts from the Unity Catalog model registry.
2. A model image builder service downloads the model artifacts from the user's storage layer and any dependent libraries from public repositories like PyPI or Conda-Forge.
3. A container image created from the artifacts is then encrypted and registered against a container registry.
4. VMs allocated from the serverless pool deploy and configure the container image to serve the user requests.

The second part is where the deployed model is served to you via REST API endpoints that you can use for batch or real-time inference within your applications. Figure 4-20 illustrates the serverless model serving process, where the model refers to feature tables from the feature store in Unity Catalog to serve your requests.

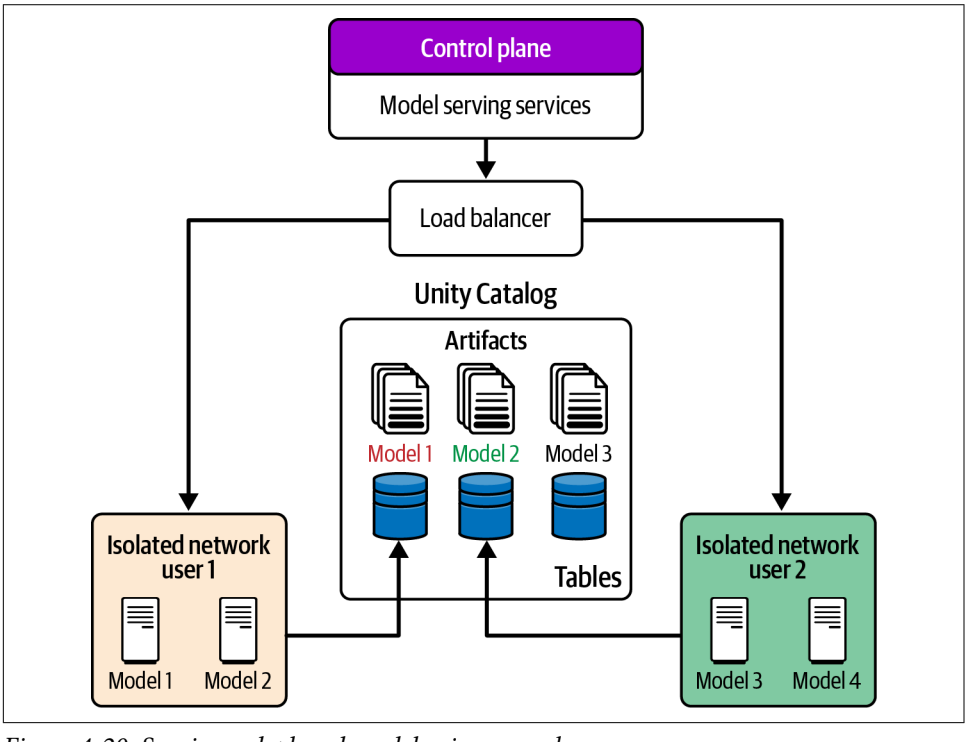


Figure 4-20. Serving a deployed model using serverless

Serverless Databricks Apps

Serverless Databricks apps enable you to build custom applications with your preferred frameworks, including Dash, Shiny, Gradio, Streamlit, React.js, and Flask, and deploy them to Databricks-managed serverless infrastructure. With governance using Unity Catalog at the core, you can securely access data assets to support various use cases. For example, data scientists at Nexa used the Databricks app to implement a writeback feature, enabling them to adjust forecasts from their ML models. You can develop your app using your favorite IDE, such as PyCharm or Visual Studio Code, and integrate it with your CI/CD pipeline for easy maintenance. Databricks' automated serverless deployment allows you to focus on building your app while Databricks handles deployment and infrastructure maintenance.

Databricks apps integrate natively with other services like Unity Catalog for governance, the Databricks data warehouse for executing your queries, generic compute

for executing Spark workloads, querying a serverless model for inference, and much more. Figure 4-21 depicts how an app is deployed using the serverless infrastructure in Databricks. Each app is allocated a service principal on deployment, and an administrator controls all access to the other resources via the service principal. Any data access is audited and governed by the Unity Catalog permission model.

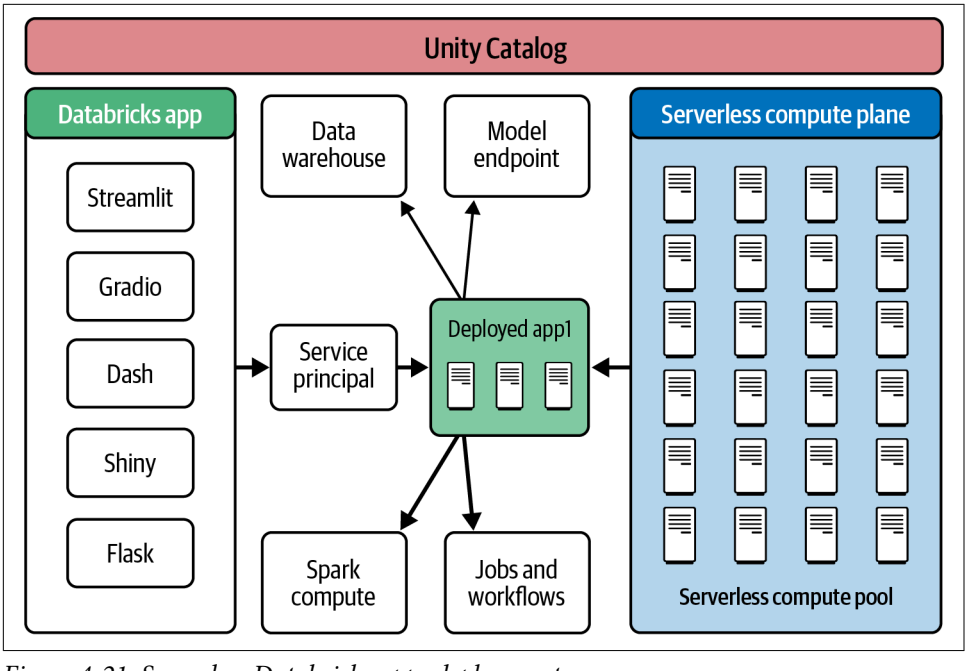


Figure 4-21. Serverless Databricks apps deployment

Serverless Lakeflow Declarative Pipelines

Lakeflow Declarative Pipelines is a declarative ETL framework for building data processing pipelines using Python and SQL. It creates streaming tables and materialized views to preserve the data processed in the Delta file format. Lakeflow Declarative Pipelines comes with many capabilities out of the box, including automated inferencing of dependencies, the ability to define data quality rules using expectations, orchestration, and error handling. Lakeflow Declarative Pipelines is also available in the serverless computing infrastructure, providing instant compute to meet your data processing timelines. From an architecture perspective, Lakeflow Declarative Pipelines uses the standard or shared access mode compute in Unity Catalog and uses the same architecture as serverless generic compute in Databricks.

Summary

> *I don't need a hard disk in my computer if I can get to the server faster…carrying around these nonconnected computers is byzantine by comparison.*
>
> —Steve Jobs

The compute topic in Unity Catalog is vast and complex, and one could easily write a whole book on it. This chapter provides a concise overview of the basics. We outlined the architecture for the current Databricks offerings, providing a foundational understanding of compute in Unity Catalog. Serverless computing is the future of data processing. You can focus on what matters most by eliminating infrastructure management overhead—driving growth. Unity Catalog plays a key role in this vision, providing unified governance across all objects and tools on the Databricks platform. With this foundation in place, you can build solutions that propel your business forward rather than getting bogged down in infrastructure management.

Chapter 5 examines how to model access controls and permissions in Unity Catalog and provides an in-depth access control model you could adopt based on the implementation examples at Nexa.

CHAPTER 5

Access Controls and Permissions Model

The more privilege you have, the more opportunity you have. The more opportunity you have, the more responsibility you have.

—Noam Chomsky

In Chapter 1, we introduced the lakehouse paradigm and explained how Databricks Platform stores the data in the cloud object storage and provides an optimization layer for efficient querying and ACID properties. In Chapter 3, we explored identity management concepts and touched upon access management. In this chapter, we describe how Unity Catalog secures the data and AI assets, various access control mechanisms, permissions models, and how you can design a scalable data architecture. We also discuss different governance models and how our fictional organization Nexa Boutique leveraged Unity Catalog to design a federated governance model with centralized policies.

Access Management

Chapter 3 briefly mentioned workspace and Unity Catalog securables. The access controls for workspace securables are applied at the workspace level, and the access controls for Unity Catalog securables are applied at the Unity Catalog metastore level. This is depicted in Figure 5-1.

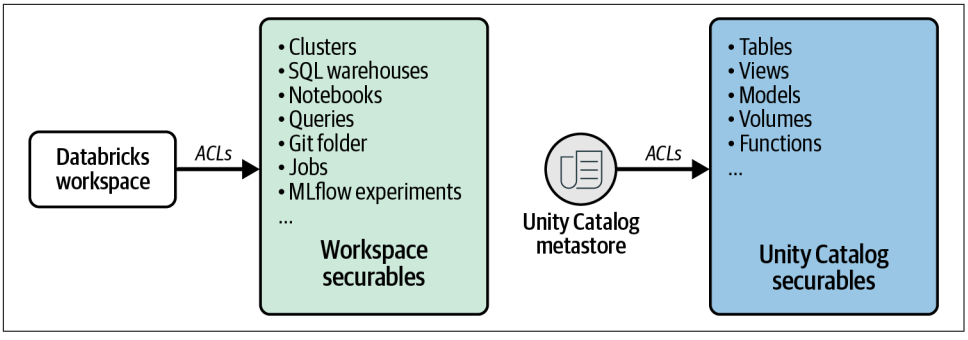


Figure 5-1. Access controls for Databricks workspace and Unity Catalog securables

From the access management perspective, you can think of a Databricks workspace as a combination of project and compute layers. The workspace securables such as notebooks, queries, jobs, dashboards, and MLflow experiments that allow you to develop project-specific assets constitute the project layer. Clusters, SQL warehouses, pools, cluster policies, model serving endpoints, vector search endpoints, and so on that provide compute for your workloads, constitute the compute layer. Unity Catalog deals with the data layer. It primarily controls access to the data and AI assets stored in the cloud object storage. You can also connect other databases, data warehouses, and data catalogs and control access to the assets within these data stores when accessed from Databricks. Figure 5-2 illustrates the positioning of the Databricks workspaces, Unity Catalog, and the data layer.

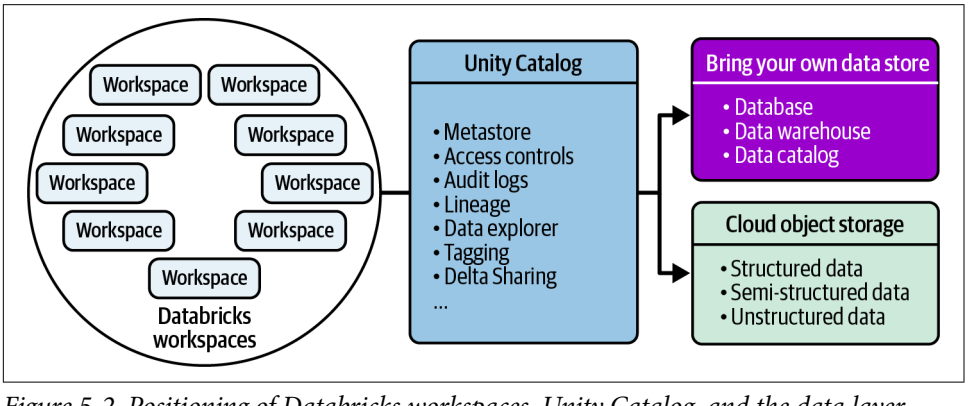


Figure 5-2. Positioning of Databricks workspaces, Unity Catalog, and the data layer

If we pick a simple example of an analyst trying to query a table in Databricks, they first log in to a Databricks workspace to which they have been given access by the admins. Once logged in, they open a notebook or the query editor to write the query. Then they attach a compute instance, either a cluster or a SQL warehouse, to run the query. All these objects that the analyst is using to achieve their goal are at

the workspace level and hence they should have access either to create or use them. For the analyst to access the desired table, the permissions are granted at the Unity Catalog metastore level. Figure 5-3 illustrates this example.

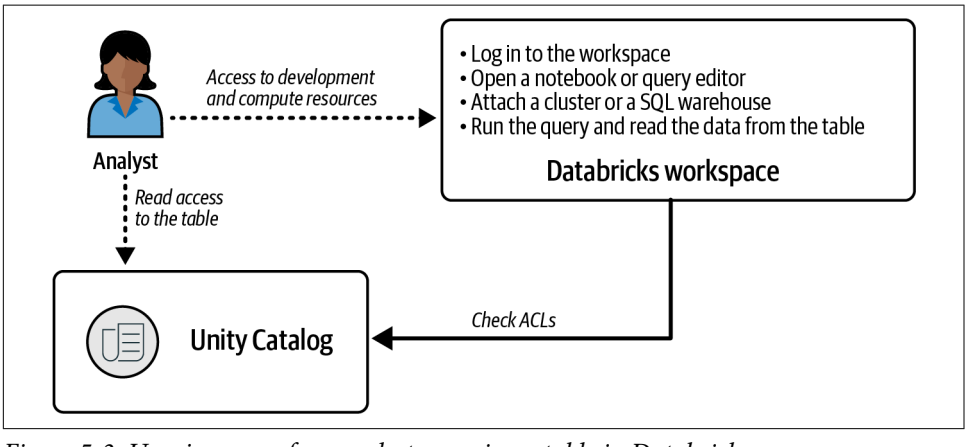


Figure 5-3. User journey of an analyst querying a table in Databricks

An important aspect to notice here is that the analyst does not need to know where the data is actually stored or on which cloud provider Databricks is running. They do not even need to know the term *Unity Catalog*. All such details can be abstracted away. The only thing the analyst needs to know about is how to access the workspace and use it to query the table. Furthermore, if the analyst is using a BI tool such as Power BI and just wants to access the data from Databricks, they do not even need to know about Databricks, provided the connection from the BI tool to Databricks is properly configured and appropriate permissions have been granted by the admins. Chapter 9 looks into the different ways to access data and AI assets from Databricks.

Privileges Versus Permissions

Privilege refers to the permissions and capabilities granted to a principal (user/group/service principal), that allow them to perform specific actions on securables. For example, user X being granted `SELECT` on a table and user Y being granted `CREATE TABLE` on a schema. Privileges are associated with a principal.

Permission refers to what actions, such as read or write, can be performed on which securables, such as tables and schemas. Permissions are associated with the securable.

In this book, we use the terms privilege and permission interchangeably.

Access Controls

Different types of access control mechanisms are used for managing access to resources and assets. You might be familiar with *role-based access control* (RBAC), which is a role-centric access control mechanism, where permissions are associated with roles and these roles are assigned to principals. *Role-based* indicates that the permissions are granted based on your role within the organization. RBAC is offered by major public cloud providers, cloud data platforms such as Snowflake and Starburst, and on-premises solutions such as Oracle and Teradata.

Databricks primarily offers *access control lists* (ACLs), which are lists of permissions attached to resources or assets that specify which principals have access to them. ACLs are user-based or identity-based. Moreover, the permissions on Databricks securables are defined within the Databricks Platform. It is not possible to define access controls for the Databricks securables outside of the Databricks Platform, such as on an identity provider.

Although you can grant permissions to individual users, the best practice is to grant permissions to groups. By defining persona-specific groups and granting relevant permissions to these groups, you can simulate the RBAC approach with Databricks. For example, you can create a group called `data_engineer` that reflects the job role of the individuals who are part of this group. You grant all the necessary privileges such as the ability to use a cluster, read and write data to a set of tables, and so on, to this group. Whenever a new individual joins this data engineering team, they should be added to this `data_engineer` group, and if someone leaves the team, they should be removed from the group.

As you may have realized, the kind of permissions that you apply on development and compute resources that are at the workspace level differ from the permissions that you apply on data and AI assets that are at the Unity Catalog metastore level. Therefore, it is important to explore both aspects to get a complete understanding of access controls on the Databricks Platform. First, we will explain the access controls on the workspace securables and then we will venture out into Unity Catalog access controls.

Workspace Access Controls

Access controls on the workspace level primarily deal with applying ACLs to workspace securables. There are three categories of users in a workspace:

Admin
: A workspace admin has the highest privilege `CAN MANAGE` on all the workspace securables. The admin can grant permissions on any workspace securable to any principal in the workspace.

Owner
: A workspace securable has an owner, who is the creator by default. The creator of the securable gets `CAN MANAGE` permission by default and can grant permission to any principal in the workspace.

User
: A workspace user by default has no access to any securables. The user needs to be granted access to a securable to use it, or they can create a new one, provided they have the required privilege to do so.

Entitlements are properties that define how a principal can interact with the Databricks workspace. They can restrict the creation of compute resources by a workspace user.

The highest privilege that you can have on all the workspace securables is `CAN MANAGE`. Apart from that, each workspace securable has a different set of privileges. For example, a `cluster` has `CAN ATTACH TO` and `CAN RESTART` privileges and a `notebook` has `CAN READ`, `CAN RUN`, and `CAN EDIT` privileges.

The best way to manage access to workspace securables is to define persona-specific groups and grant permissions to these groups. If we take the example of an `analyst` persona, who wishes to access and analyze data from within the Databricks workspace as well as using a BI tool, the permissions on workspace securables would be as follows:

1. Access to Databricks workspace: The analyst can log in to the workspace.
2. `CAN MANAGE` privilege on a personal directory on the workspace: The analyst can create a notebook, a query, a dashboard, or a Genie space.
3. `CAN ATTACH TO` privilege on a cluster or `CAN USE` privilege on a SQL warehouse: The analyst can use the compute instance to run the queries.

In addition, the analyst can be granted permissions on workspace securables created by other users. The exact permissions depend on the securables and the level of access the analyst needs to be given. The important thing to note here is that the analyst should have access to all the securables that are required to use a feature. For example, if the analyst has been granted access to a dashboard, the analyst also needs the permissions to use the underlying compute instance, which is a SQL warehouse, and access to the underlying tables and views.

In essence, you first create a group with `data_analyst` in the group name with any prefix or suffix to indicate the business unit or project. You then add individual analysts to the group. Create all the necessary resources such as clusters, SQL

warehouses, and so on. Finally, you grant all the necessary permissions to the group. Figure 5-4 shows the securables and ACLs relevant to this example.

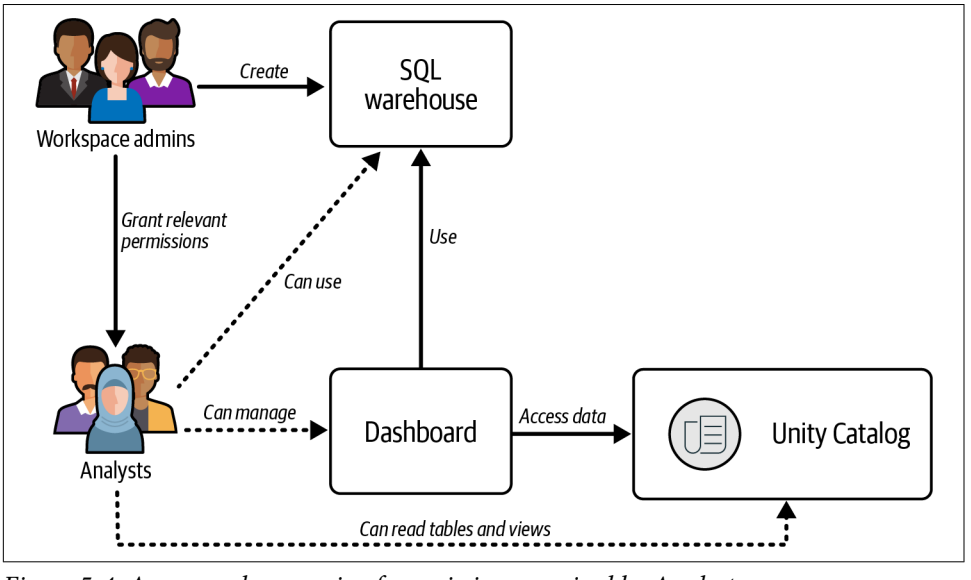


Figure 5-4. An example scenario of permissions required by Analyst persona

Similarly, you can define other personas such as Data Engineer, Machine Learning or AI Engineer, and grant relevant permissions.

Unity Catalog Access Controls

Unity Catalog securables are structured under a three-level namespace hierarchy. The root of all the objects in Unity Catalog is the metastore, under which you can create catalogs. A *catalog* is at the first level of the object hierarchy, under which you can create schemas. A *schema,* which is also termed a *database,* is the second level of the object hierarchy, containing all the data and AI assets such as tables, views, models, and so on. Figure 5-5 shows the Unity Catalog object hierarchy.

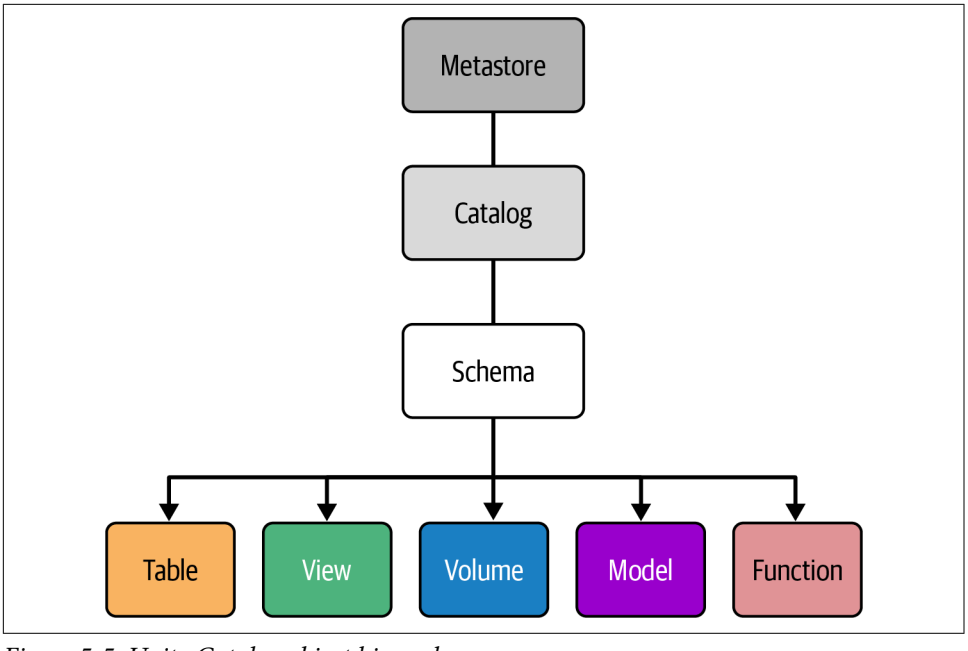


Figure 5-5. Unity Catalog object hierarchy

As you might have noticed, the *volumes* that store unstructured data and the *models* and *functions* that constitute AI assets, are placed alongside *tables* and *views*. The access controls, permissions model, and sharing mechanisms are consistent across all these assets. Figure 5-6 shows an example of data and AI assets in Nexa Boutiques's retail sales, cataloged in a metastore.

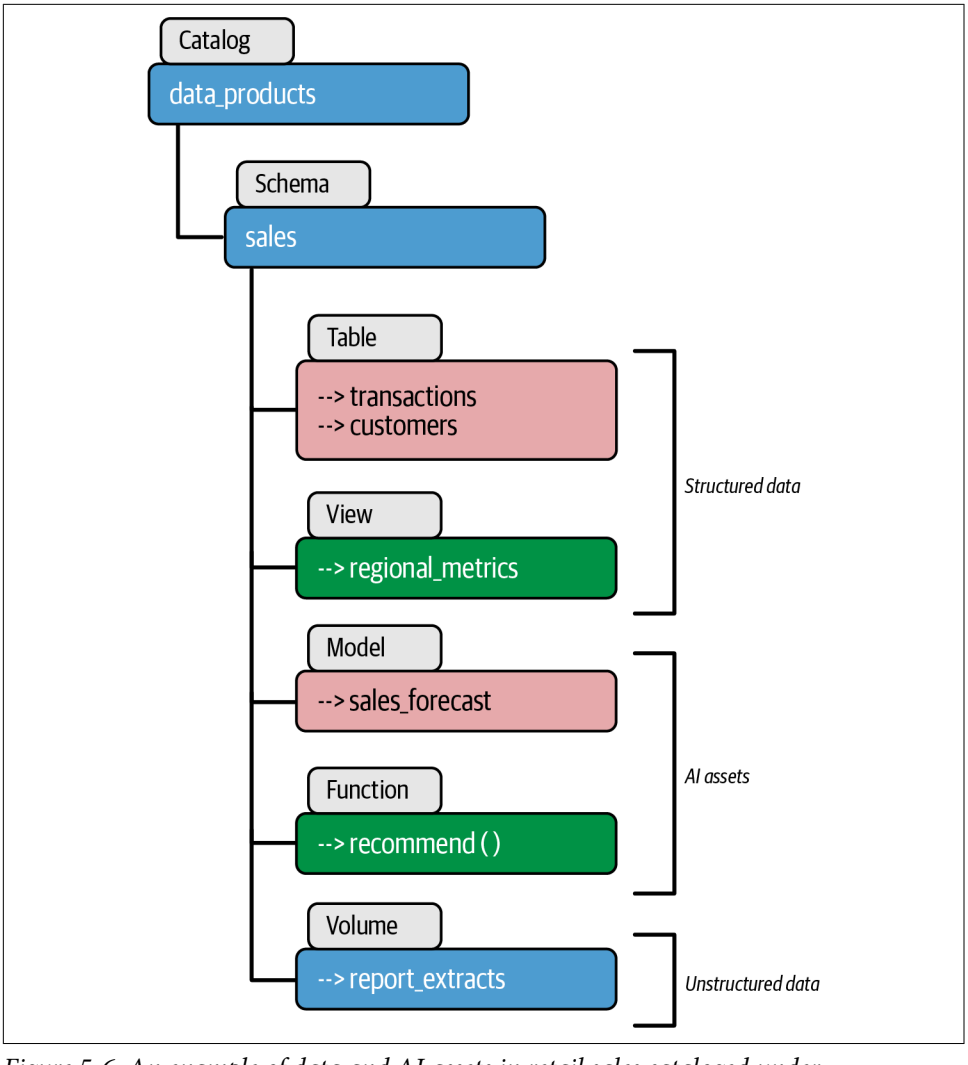


Figure 5-6. An example of data and AI assets in retail sales cataloged under catalog/schema

In the preceding example, if you want to access the tables `transactions` and `customers`, you would reference it in relation to their parent catalog `data_products` and schema `sales`. A SQL query to access the tables would look like this:

```
SELECT * FROM data_products.sales.transactions;
SELECT * FROM data_products.sales.customers;
```

Unity Catalog securables and the corresponding permissions are defined at the metastore level. A *metastore* is an abstract entity that acts as a metadata layer. It primarily stores the following metadata:

Structural metadata
: The object hierarchy, schema definitions, data types, data source locations, and so on

Discovery metadata
: Tags, comments, and descriptions that facilitate the discovery and understanding of corresponding data and AI assets

Access control metadata
: ACLs, applicable filtering and masking functions, and other permissions that govern access to data and AI assets

Lineage metadata
: Data that indicates the relationships and dependencies between various assets, tracking their journey throughout their lifecycle

Audit metadata
: Records of who performed which action and when, fueling security and compliance efforts

Permissions Model

Earlier, we mentioned that although the data and AI assets are stored in the object storage, as a user you do not need to know where the data is or how to access it from the storage. This is because Unity Catalog abstracts out these details using data access configuration objects—*storage credentials* and *external locations*:

Storage credentials
: A Unity Catalog wrapper object for credentials that authorize Unity Catalog service to access the corresponding cloud storage object, that is:

  - An IAM role that grants read/write access to an S3 bucket
  - An MI or an SP that has `STORAGE BLOB DATA CONTRIBUTOR` role on an ADLS Gen2 account
  - A service account that has STORAGE LEGACY BUCKET READER and `STORAGE OBJECT` ADMIN permissions on a GCS bucket

External locations
: A Unity Catalog wrapper object that maps the storage credentials to a specific cloud storage URI. A single storage credential can be associated with multiple external locations that are pointing to the same cloud storage object.

In terms of the object hierarchy, storage credentials and external locations are created directly under the metastore. You can grant privileges to principals on these objects, like the securables that fall under the catalog hierarchy. Figure 5-7 shows the positioning of storage credentials and external locations in the Unity Catalog object hierarchy.

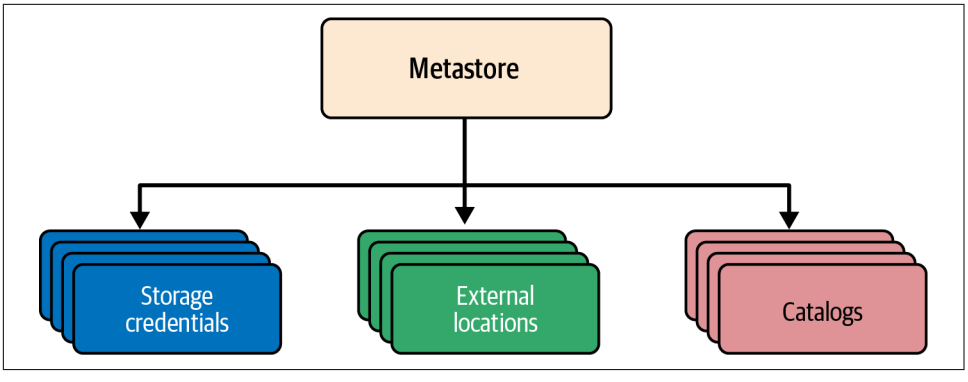


Figure 5-7. Positioning of storage credentials and external locations in the Unity Catalog object hierarchy

As storage credentials and external locations are necessary configurations for data access, you create them first before you create any other Unity Catalog securables. In other words, you design the storage layout and configure access from Unity Catalog. You then design the catalog layout and map the storage credentials and external locations to catalogs, schemas, tables, or volumes. We will look at the storage and catalog layout later in this chapter.

Similar to a workspace, we can categorize Unity Catalog users into three main categories:

Admin
: A metastore admin has the highest privilege on the corresponding Unity Catalog metastore. The admin can grant privileges on and change ownership of any Unity Catalog securable to any account-level principal.

Owner
: A Unity Catalog securable has an owner, who is the creator by default. The creator of the securable gets full privileges on it by default. They can grant privileges on and change ownership of the securable to any account-level principal.

User
: An account user by default has no access to any securable. The user needs to be granted specific privileges on a securable to use it or to create new securables.

There are specific sets of privileges at each level of the Unity Catalog object hierarchy. These privileges are granted to principals, and these are inherited down the hierarchy. For example, if you have `ALL PRIVILEGES` on a catalog, then you inherit `ALL PRIVILEGES` on all the objects within that catalog. The principals will inherit the privileges on all current and future objects within the hierarchy.

Granting privileges on a securable can be done only in the following cases:

- You are a metastore admin.
- You are the owner of the securable.
- You are the owner of the parent object of the securable.
- You have `MANAGE` privilege on the securable.
- You have `MANAGE` privilege on the parent object of the securable.

You can drop a securable only if you have the ownership of and/or `MANAGE` privilege on the securable or its parent. This also applies to metastore admins. However, a metastore admin can change ownership of the object to themselves or grant themselves the `MANAGE` privilege and then drop the securable. Note that `ALL PRIVILEGES` does not include `MANAGE` privilege and having `MANAGE` privilege does not automatically grant `ALL PRIVILEGES`.

Access Controls on Nontabular Data

ML and AI engineers at Nexa Boutique were often troubled by the lack of ways to properly secure nontabular training datasets, such as audio/video files and ML models. Problems included where to store them, how to properly version them, how to map a model to the dataset it was trained on, and, more importantly, how to configure proper access controls without annoying their colleagues. Although these engineers were supposed to focus on model development and fine-tuning, they ended up spending too much time on application development, platform operations, security, and governance. That's one of the reasons the unified approach to securing and governing both tabular and nontabular assets appealed to them a lot. Such experiences are not unique and are quite prevalent across organizations. While some of the challenges specific to ML use cases are mitigated through frameworks such as MLflow, the security and governance requirements are not fulfilled.

Take the example of Databricks itself, before Unity Catalog was created and made available to customers. The dataset and models were stored in the context of a workspace, and the access controls were applied only at the workspace level. This made it challenging to share datasets and models across Databricks workspaces. Moreover, the nontabular data such as files had to be stored in DBFS, a distributed file system used as an interface to the cloud object store. DBFS does not support granular access

controls, so anyone who has access to the Databricks workspace can access any file on the DBFS. In essence, there was no option to properly govern nontabular datasets and models.

With Unity Catalog the story is very different. As we've mentioned, you can use a volume to store nontabular datasets, and ML models have their own object type *model* within the Unity Catalog object hierarchy. A *volume* is a logical representation of a cloud object storage location. It catalogs a collection of directories and files in a given cloud storage path and abstracts out the location and access details from the user. Volumes might sound quite similar to external location objects, defined previously. You might be wondering why we have two similar objects. The difference is that external locations are fundamental data access configuration objects, whereas volumes are user-facing abstractions on top of it. You need an external location defined, which you can use to create volumes.

If we refer back to Nexa's retail sales data example from "Unity Catalog Access Controls" on page 138, you can access and list the files in the volume `report_extracts` using the catalog and schema reference as follows:

```
LIST '/Volumes/data_products/sales/report_extracts/';
```

You can access the contents of a volume if you have been granted at least `READ VOLUME` privilege on it and `USE CATALOG` and `USE SCHEMA` permissions on the corresponding parent catalog and schema. You don't need to know the exact location of the files on the cloud storage, nor do you need to have any access at the cloud storage level. Note that if you have been granted access on a volume, you will have access to all the directories and files within that volume. Currently, access controls on individual directories and files within a volume are not supported.

The nature of volumes makes them quite flexible in terms of what they can store. You can use volumes to store not just datasets but also libraries, config files, certificates, and so on. This implies that you can isolate project or team-specific resources, no matter the type, into specific catalog structures and apply access controls accordingly.

Take Nexa's project for inventory optimization, which is termed *singularity*. If you create a catalog for this project with the same name, `singularity`, you can create schemas within that catalog and store all the related structured data as tables or views, and store all the related unstructured data as files, code libraries, config files, and so on in volumes, and save models as Unity Catalog models. You create project-specific IdP groups like `singularity_engineers` and grant privileges to that group on the `singularity` catalog and objects inside it. You'd have all data and related assets in one place, and this simplifies discoverability, accessibility, and governance.

Managed and Unmanaged Datasets

When you catalog datasets in Unity Catalog, you need to decide if you want to store them as a Databricks-managed dataset or not. This applies to only two types of securables: tables and volumes. Managed tables can only be either Delta or Apache Iceberg format. As the name implies, in the case of *managed datasets*, Databricks manages certain aspects of the dataset, namely location and optimization.

It is very important to note that, by using managed datasets, that is, managed tables/volumes, you are not giving up control over your data because you can choose which storage unit it is stored in. Therefore, you retain full control over your data. When we say Unity Catalog manages the data location, it means Unity Catalog takes care of the path prefixes and requires the user to not access the data using the cloud URI and instead use the logical Unity Catalog abstraction, that is, `catalog.schema.table/volume`. In essence, you are not getting into a vendor lock-in by using managed tables/volumes.

To be able to create managed and unmanaged tables and volumes, we should first understand how data access configurations are associated with Unity Catalog securables. As mentioned previously, the data access configurations are defined using storage credentials and external locations. You can associate storage locations backed by storage credentials and external locations with different hierarchy levels of Unity Catalog securables. This is depicted in Figure 5-8:

- When you create a table or a volume, you can specify a storage location. This allows you to precisely control where the contents of the table or volume are stored on cloud storage. You can achieve this by using the `LOCATION` keyword while creating the object. The tables and volumes created this way are correspondingly termed *external tables* and *external volumes*.
- You can specify a storage location while creating a schema. All the managed objects created within that schema will use the specified storage location as the storage root.
- You can specify a storage location while creating a catalog. All the managed objects created within that catalog will use the specified storage location as the default storage root.
- You can set a storage location at the metastore level, which acts as the default storage root for all the managed objects created within that metastore, unless you have specified one at the catalog or schema levels.

We do not recommend setting a storage location at the metastore level as it might result in many of the managed assets, even the unintended ones, getting stored in a common shared location. It might also result in hitting some limits on the cloud storage at scale. Note that, if you do not specify a storage location at the metastore level, you have to specify a storage location when you create a catalog.

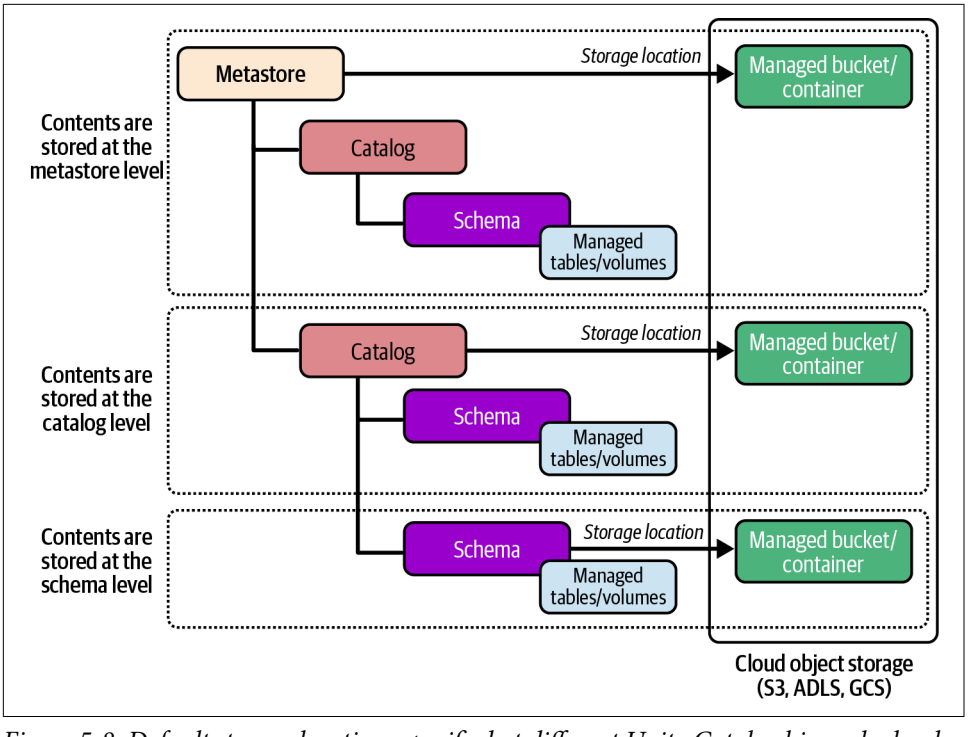


Figure 5-8. Default storage locations specified at different Unity Catalog hierarchy levels

In essence, external tables and external volumes have a storage location directly associated with them and are used to store their contents. For managed tables and volumes, the storage location defined at the closest parent is used. To elaborate:

1. If you have a specified storage location at the schema level, then it will be used to store the contents.
2. If not, the storage location specified at the catalog level will be used.
3. If no storage locations are specified, either on the schema or the catalog levels, then the contents are stored at the metastore level storage location.

This is illustrated in Figure 5-9.

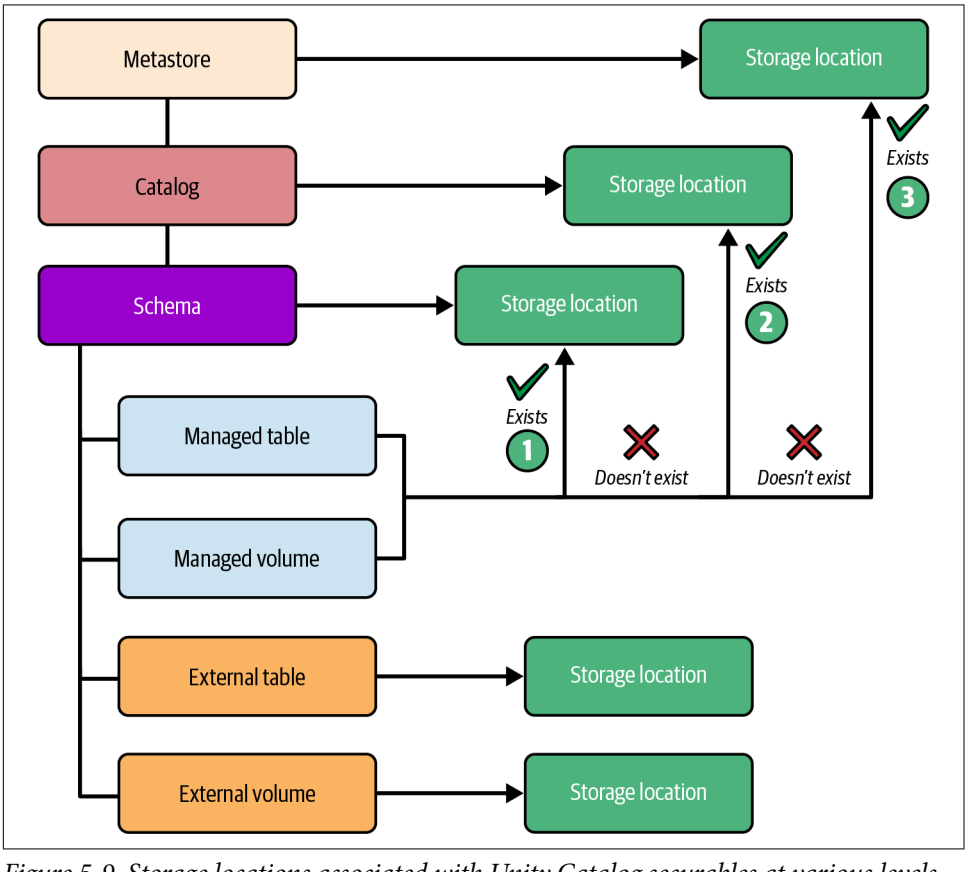


Figure 5-9. Storage locations associated with Unity Catalog securables at various levels of object hierarchy

In simple terms, the storage locations associated with the metastore, catalog, or schema are managed locations. The datasets stored in these locations are managed datasets. External tables and contents of external volumes are unmanaged datasets. Managed and unmanaged datasets can coexist under a catalog/schema. When you drop a managed table or a volume, the underlying files are automatically purged by Databricks within 30 days. However, when you drop an external table or volume, only the metadata cataloged in Unity Catalog is purged; the underlying files stay intact and need to be manually deleted.

External Locations: Things to Keep in Mind

- External locations are part of Unity Catalog's data access configuration and are used to specify the storage path for managed assets as well as external tables/volumes. The keyword "external" often leads to confusion and might lead you to believe that it is relevant for only external tables/volumes.
- You cannot have two external locations pointing to the same cloud storage path. In other words, you cannot have overlapping external locations.
- To avoid accidental overlapping paths, we recommend you don't create external tables/volumes at the root of an external location. Instead, create them in subdirectories within an external location.

Managed tables and volumes are stored in a path that is constituted using the metadata information of Unity Catalog objects, and this depends on the storage root. For example, a managed table with a storage root at the catalog level has the following path structure:

```
<CLOUD_STORAGE_URI>/<OPTIONAL_SUBPATH>/__unitystorage/catalogs/
   <CATALOG_ID>/tables/<TABLE_ID>
```

`CLOUD_STORAGE_URI`

This is the storage root and the subpath in the cloud object storage.

`OPTIONAL_SUBPATH`

If set, further specifies where the managed objects are stored.

`CATALOG_ID`

This is the Universal Unique Identifier (UUID) of the catalog.

`TABLE_ID`

This is the UUID of the table.

A managed table with a storage root at the metastore level storage location has the following path structure:

```
<CLOUD_STORAGE_URI>/<METASTORE_ID>/tables/<TABLE_ID>
```

`CLOUD_STORAGE_URI`

This is the storage root and the subpath in the cloud object storage.

`METASTORE_ID`

This is the UUID of the metastore.

`TABLE_ID`

This is the UUID of the table.

We mentioned the path structure to show how it differs from the content path of an external table/volume. There is no point in trying to access the managed datasets using the file paths, as it is clearly inconvenient, they can change as they are managed by Databricks, and they just aren't meant to be used that way.

If you're wondering whether to store your datasets, especially tables, as managed or unmanaged, you're not alone. This is a question that we come across quite often. On the one hand, using managed tables results in less maintenance work as the tables are automatically optimized in terms of storage and performance. On the other hand, with external tables, you have a clear human-readable path for the content files associated with the tables. Until recently, the latter would have been useful for accessing data externally as files, that is, outside of Databricks without using Databricks compute. However, now you can access the contents of a table using Unity Catalog REST APIs and you don't need to fetch the files directly. Therefore, managed tables have a clear advantage. Chapter 9 explains the Unity Catalog REST APIs and other ways to access data from Databricks. As for using external tables, the only reason to choose to use them is when you have strict necessity to store data in formats other than Delta Lake or Iceberg.

Advanced Access Controls

Since its inception, Unity Catalog has been evolving rapidly and in the right direction. We have seen more and more features being released that allow us to incorporate complex governance requirements while designing solutions based on Unity Catalog. We can refer to them as *advanced* access controls since these features add more dimensions to the core access control mechanism. In other words, these features provide you with more options and flexibility in developing and enforcing access control policies. This is an area where Unity Catalog needs to evolve further to be a full-fledged governance solution. In our opinion that is happening, and the challenge for us is to keep up with the latest advancements and feature releases. We will focus on the features that are currently available and usable but will hint at the future state of affairs that seems inevitable and mention things we think would be useful for you to know so you can design solutions accordingly.

Binding Unity Catalog securables with workspaces

We saw earlier that Unity Catalog securables exist at the metastore level and can be accessed through any Databricks workspace associated with that particular metastore. You can control this by *binding* a Unity Catalog securable to specific workspaces. This binding feature is applicable to catalogs, storage credentials, and external locations. For example, when you bind a catalog to a specific workspace, you can access that catalog from only that particular workspace. For regular users who have access to that

catalog, they will no longer be able to access it from workspaces apart from the one that is bound to it. Even if you are a metastore admin, although you can still see the catalog in the Catalog explorer tab of other workspaces, it is grayed out and cannot be accessed.

Binding a catalog to a workspace is a good way to isolate project- or business unit-specific resources. Suppose you have a set of workspaces dedicated to a specific organization unit (OU), and you have created a set of catalogs to store data and AI assets that belong to that OU. Binding those workspaces to the corresponding catalogs makes sure that the assets can be accessed within only that OU. You can further restrict access based on the operational tiers of the development lifecycle—for example, restricting production catalogs to be accessed only by production workspaces. Basically, this feature helps prevent accidental cross-OU data access as well as isolating operational tiers to enable a clean and secure development process.

Isolating operational tiers sounds good in theory, but very often you need to access production assets in other environments. For instance, analysts might want to run some ad hoc analysis on the latest sales numbers, or ML engineers might want to build a model to forecast the demand for certain products. In such cases, you need to provide the users access to production data. However, it might not be necessary to grant them access to the entire production environment. This is where read-only binding of catalogs to workspaces helps. If you bind the production catalogs to development workspaces in read-only mode, the analysts or ML engineers can access the required data from development workspaces. Since the binding is set to read-only, the users cannot write to the production catalogs, preventing any accidental mishaps.

Figure 5-10 shows an example of workspace catalog binding. The production catalog is bound to the production workspace as read-write and to the development workspace as read-only. The development catalog is bound to only the development workspace.

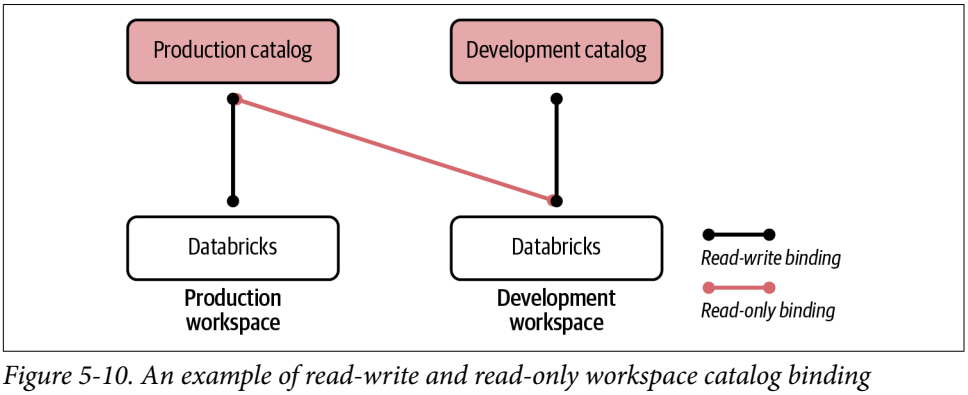


Figure 5-10. An example of read-write and read-only workspace catalog binding

Fine-grained access controls

If you have a CSV file stored in an S3 bucket, what is the difference between applying access controls directly at the file level and abstracting the CSV file as a table and then applying access controls on the table? The answer depends on what level of controls you wish to enforce. If you care about controlling access only to the entire CSV file contents, then there is not much difference. But if you wish to grant access only to selected columns or rows within the CSV, then file versus table matters, as it is not possible to do that level of access control at the file level.

Unity Catalog allows for controlling access at the row and column level of a table. Given a principal, you can decide which columns and rows the principal is allowed to access. This functionality is broadly termed *fine-grained access control* (FGAC). This feature is quite useful if you have use cases with sensitive data, such as PII, or use cases where you want to simply filter the records based on certain column values, such as geographical regions.

Figure 5-11 shows an example of what FGAC looks like. In this case, a user, Aly, who is part of the HR group, is allowed to access an entire table consisting of employee information, and another user, Bob, who is not in the HR group, is allowed to see only some of the data with filtered rows and masked columns.

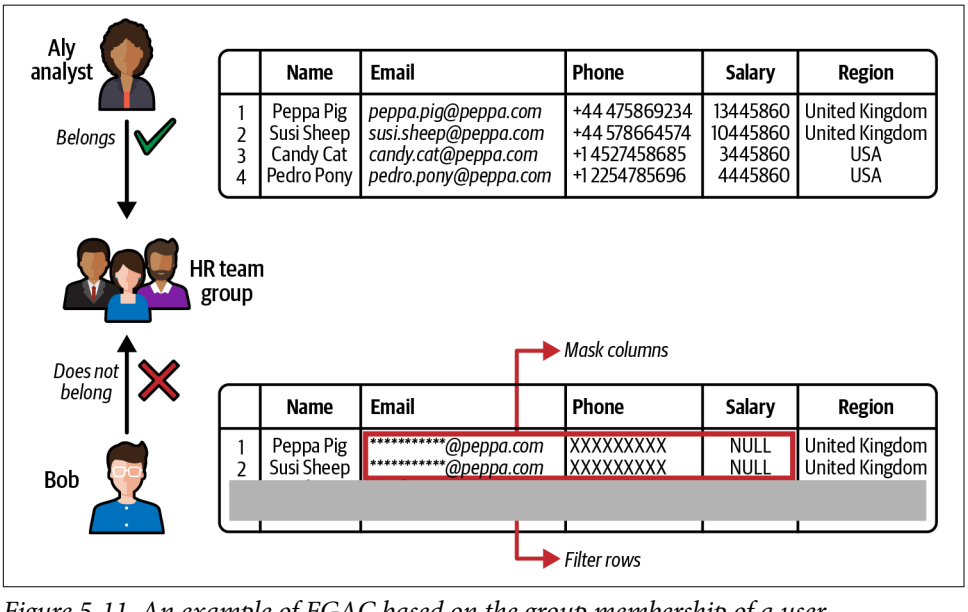


Figure 5-11. An example of FGAC based on the group membership of a user

Databricks Unity Catalog lets you define functions with logic that will determine which principal is allowed to access which rows and columns. In other words, you write functions to filter rows or mask columns and apply that function to a table. The part that we really like is that the filtering or masking function is itself a securable in Unity Catalog. You can view the function definition alongside the table and control access to the function just like you do on the table. Figure 5-12 depicts an example of a row filter function cataloged along with the table on which it is applied.

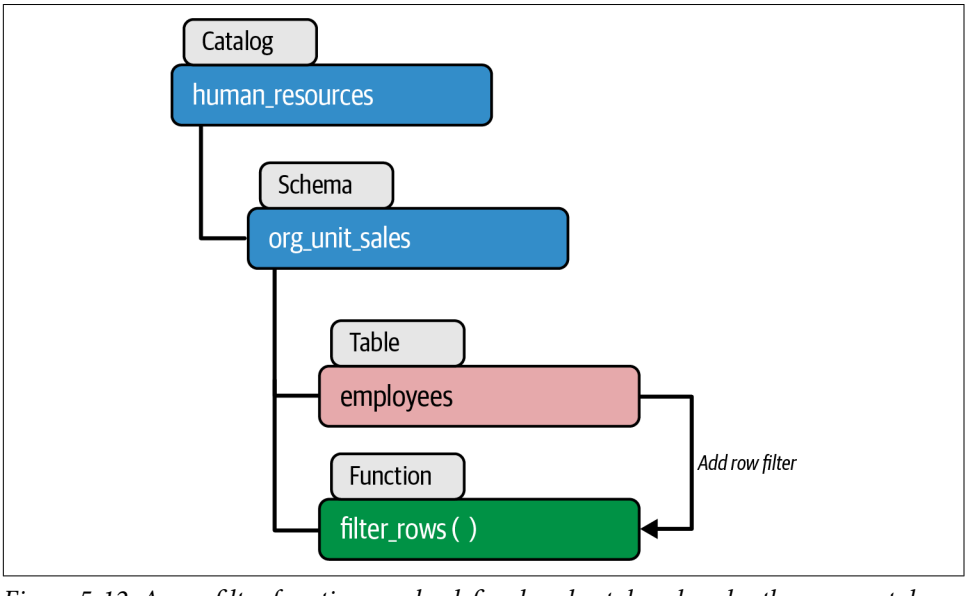


Figure 5-12. A row filter function can be defined and cataloged under the same catalog and schema where the table exists

Applying FGACs does not encrypt or anonymize the data at rest. It only happens dynamically when the data is accessed via a cluster or a SQL warehouse on a Databricks workspace. In other words, these controls are dynamic in nature. When a user queries a table with row filters and column masks, the rows are filtered and columns are masked on the Databricks compute layer, before the contents of the tables are returned to the user. This means the Databricks compute plays a key role in enforcing these controls. If you access a table in Unity Catalog using the REST API, for instance, FGACs are not enforced, as there is no Databricks compute involved. Chapter 9 discusses more about how to access data in Unity Catalog using the APIs. Figure 5-13 shows the high-level query lifecycle for accessing a table with row filters or column masks.

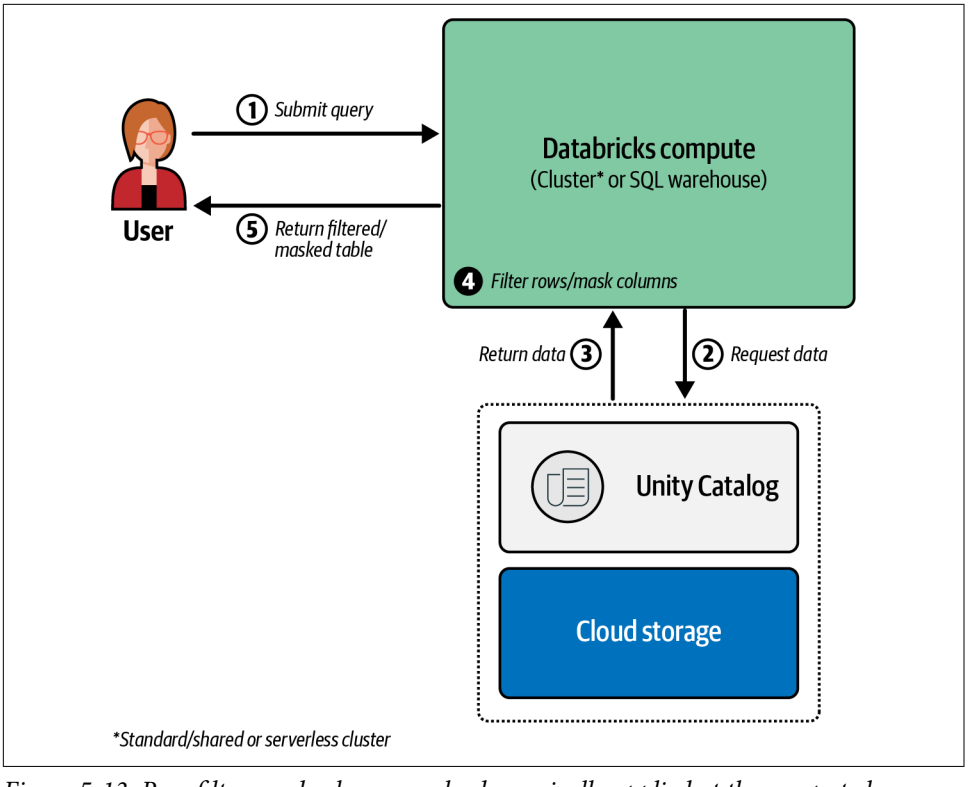


Figure 5-13. Row filters and column masks dynamically applied at the compute layer

The query lifecycle has the following stages:

1. The user submits a query to access a table with a row filter or column masks.
2. The Databricks compute interacts with the Unity Catalog service, which validates the metadata, checks user permissions on the table, and returns a list of paths/files and scoped-down temporary tokens. The compute instance fetches the data from the cloud storage using these paths/files and temporary tokens.
3. The cloud storage returns the data to the compute instance.
4. The row filter and column mask functions are applied to the table.
5. The filtered contents of the table are returned to the user.

Note that the query lifecycle in Figure 5-13 applies to Databricks standard/shared clusters, SQL warehouses, and serverless compute. The lifecycle differs for a dedicated/assigned cluster, which uses an additional background service to apply row filters and column masks. We won't go into much detail about that in this book.

Attribute-based access controls

We have seen different ways of controlling access to Unity Catalog securables. These are sufficient for most small and, to a certain extent, medium-sized organizations. However, for large organizations, especially with complex governance requirements, the core access controls mechanism and the ownership model of Unity Catalog might not be sufficient and can pose challenges in scaling the governance model. For example, the owner of a Unity Catalog table can grant permission to any other Databricks user to access it (the receiver of the permission can only access the table if they have `USE` privilege on the parent catalog and schema). Some organizations want to restrict the sharing of data and AI assets and want to define centralized policies that enforce such restrictions. Such rules or policies typically incorporate various attributes such as user roles, environmental factors, and resource sensitivity that need to be dynamically evaluated to determine if a user requesting access to a securable is allowed to access it. Moreover, these policies should be applied to multiple securables at different object hierarchy levels, in order to scale.

Consider the healthcare sector, where organizations often deal with highly sensitive information in different domains such as patient care and drug research. These are highly regulated and often have complex policies that dictate who can access certain data, when and at which stage of the project, and so on. For example, a policy in the patient care domain can state that *Nurses* should be allowed to access only the records of patients assigned to their ward and only during their scheduled shifts. A policy in the drug research domain can specify that *Researchers* should be allowed to access only the research data of the project they have been approved for and only from the company-owned secure device during business hours.

ABAC allows organizations to fulfill such requirements. It is offered by major cloud providers as well as governance specific solutions such as Immuta.

In 2014, the National Institute of Standards and Technology (NIST) published a paper titled "Guide to Attribute Based Access Control (ABAC) Definition and Considerations" that defined how enterprises should implement ABAC. The NIST publication defines three key concepts for ABAC:

Attributes
: Attributes are characteristics that define subjects, objects, and the environment.

Policies
: Policies are the rules that govern access control decisions.

Policy enforcement
: Policy enforcement is the process of evaluating the attributes against the policies to make access control decisions.

Databricks' current release of ABAC features includes enhanced tagging and scalable row filtering and column masking techniques. The ABAC implementation of Databricks Unity Catalog has three core concepts: *attributes, policies,* and *inheritance,* and it seems to be in line with NIST guidelines. Attributes are derived from the context of the securable and the principal trying to access that securable, as well as other environmental and temporal factors. Examples of *attributes* are the identity of the principal, tags associated with the securable, and time of access. Policies use these attributes to dynamically control access. A policy associated with a securable is inherited down the Unity Catalog object hierarchy and coexists with existing object permissions.

The current release of Databricks ABAC has two main features: governed tags and ABAC policies.

Governed tags

We will briefly discuss tagging in Chapter 7 in the context of the discoverability of data and AI assets. Unity Catalog tags, in general, are free-text key-value pairs that can be assigned to Unity Catalog securables. However, without any controls on how tags are created and applied, their functionality is limited. Ensuring that tags are consistently applied with proper naming while conforming to the organizational standards helps to enforce centralized policies. Governed tags allow admins to create/specify tags that have an associated tag policy to enforce its rules. It allows defining a set of allowed values for each tag and controlling which principals can manage these tag definitions and can assign them to Unity Catalog securables.

Governed tags are defined at the account level and apply across all the workspaces within that account. You create a governed tag by specifying a tag key and zero or more allowed values. For example, you can create a governed tag with a tag key *domain* and specify the values such as *finance, sales, marketing*, and so on. You can then apply the tag to a Unity Catalog securable, such as a catalog or a table, and specify one of the allowed values. Once created, you can also decide who can apply and manage the governed tag by setting the appropriate permission, `ASSIGN` or `MANAGE`, to principals.

Since governed tags enforce governance on tags, they can be used to implement data classification, cost management, discovery of assets, and so on. Moreover, governed tags are a necessary feature for applying ABAC on Databricks. Governed tags also enable trusted automation, as you can rely on tags to have only allowed values and to be applied only by allowed principals.

With the introduction of governed tags, we now have three types of tags in Unity Catalog. Figure 5-14 illustrates different types of Unity Catalog tags:

Discovery tags
: Tags without any policies

Governed tags
: Tags with tag policies applied

System tags
: Databricks predefined governed tags

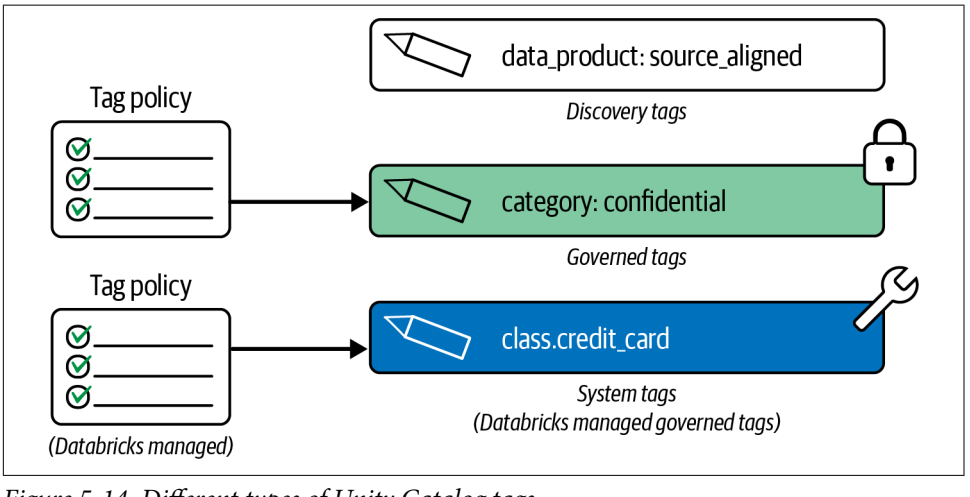


Figure 5-14. Different types of Unity Catalog tags

ABAC policies

FGACs that we discussed previously are applied per table. This means, if you want to mask the email field for a hundred different tables, you have to apply the masking function on each of these tables separately. This approach is hard to scale. Moreover, if your organization wants to define and enforce centralized policies for handling sensitive data, you will have a hard time implementing and managing them. Databricks ABAC provides a different approach for FGAC. Instead of applying the row filters and columns masks on every table, you can define a policy, which you can apply on a catalog, schema, or table.

ABAC policies applied at the catalog or schema level will be enforced on all the relevant child objects. An ABAC policy is applied based on the governed tags. In other words, you first define governed tags and apply them to Unity Catalog objects such as tables and columns. You then define an ABAC policy and apply it at the catalog, schema, or table level. At the time of this writing, the supported policy types include row filters and column masks.

A row filter/column mask policy includes the following components:

- A securable on which the policy is applied (catalog/schema/table)
- Principals for whom this is applicable (users/groups/service principals)
- Exceptions for principals on which the policy is not applicable
- Row filter or column mask UDF
- Tables that match a specific governed tag
- Columns that match a specific governed tag

ABAC policy, once applied, is automatically enforced on new objects that match the policy criteria. For example, if you have a column mask policy applied on a catalog to mask the columns tagged with `PII` in tables tagged as `confidential`, then the policy is enforced on any new tables with the tag `confidential` and columns with `PII` tags. The policy is also automatically applied to any tables/columns if you add the corresponding tags. In essence, you can control which objects the policy applies to using the governed tags, without modifying the policy.

This makes it much easier to manage centralized policies. Moreover, since the policy can be applied at the catalog or schema level, the policy is enforced on all the tables/columns with proper tagging, hence, easier to scale. Figure 5-15 shows an example of an ABAC column mask policy applied to the catalog `data_products`. The policy is effective for all Databricks users in the account, except for the certified admin group. The policy automatically masks the columns that are tagged with the governed tag `[Category, PII]` in the table with the governed tag `[Category, Confidential]`.

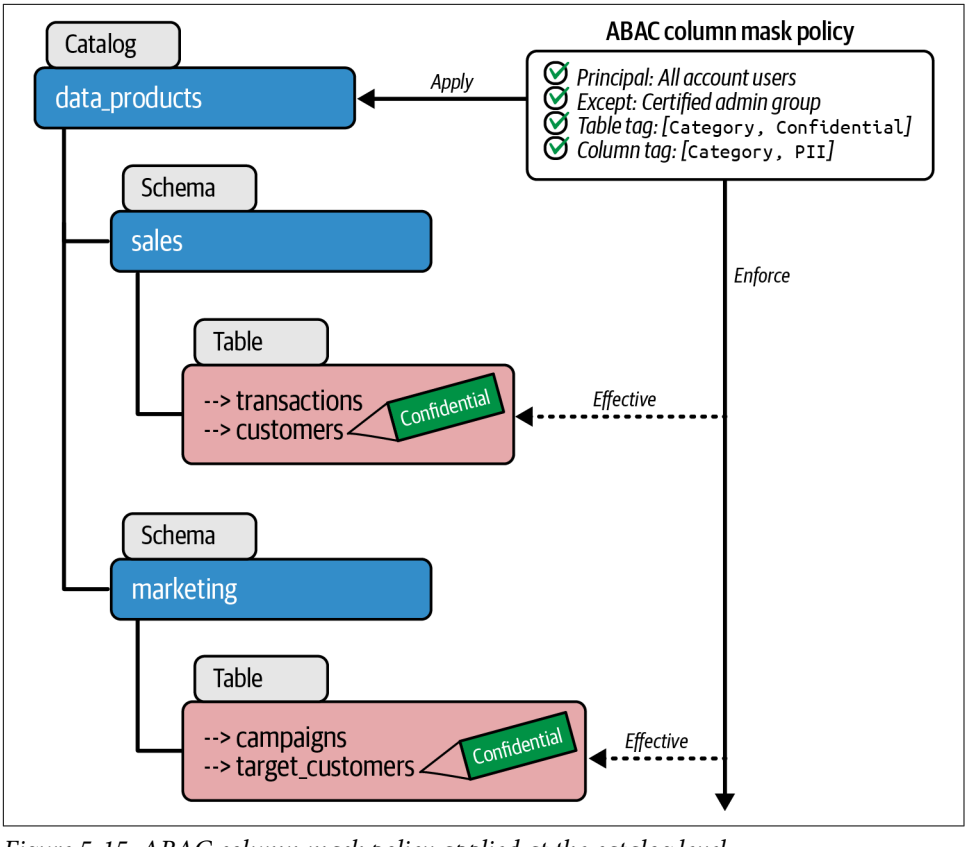


Figure 5-15. ABAC column mask policy applied at the catalog level

Yet another benefit of ABAC policies is that you can apply multiple policies with different conditions on the same table. You can do this by assigning multiple governed tags and defining policies based on corresponding tags to fulfill different governance requirements. For example, you anonymize a subset of columns with PII for all Databricks users except for the certified admin group, using a policy. Now, imagine that you are required to anonymize additional columns that are not PII but need to be confidential. In this case, you create a new governed tag(s) and apply it to the corresponding table and columns and define a new policy that masks these columns for all Databricks users except for any allowed groups. Now you have two policies applied to the same table, which you can independently update if needed. Note that at the time of this writing, only one column mask or row filter can be applied to a given column or row within the object hierarchy.

Data Governance Models

When we hear the word *governance*, the word *government* tends to spring to mind. And actually, we can use government examples to illustrate some aspects of data governance. For example, France and Germany stand as two of the most influential nations in Europe and globally, both exemplifying democratic governance with robust electoral systems. To the casual observer, these countries may appear remarkably similar, given their shared status as economically prosperous, modern Western democracies with deep cultural and religious foundations. Additionally, both nations exhibit a strong sense of linguistic pride. However, a more thorough examination reveals significant distinctions between France and Germany in terms of their political structures, governance approaches, and cultural nuances.

The French political landscape is characterized by a semi-presidential system with a strong centralized tradition, while Germany operates under a federal parliamentary model emphasizing decentralized governance. In France, up to 80% of the public spending happens at the national level, whereas in Germany it's around 50%, and the states have constitutionally protected powers. In Germany, *municipalities*, that is, local governments, handle two-thirds of the public investment, whereas in France, municipalities depend on Paris, the capital, for funding. Germany embraces policy-making through cooperative federalism, whereas France follows a vertical power structure.

Adding Switzerland, which has a cantonal system, to the mix reveals yet another governance model. *Cantons* (states) and municipalities hold significant autonomy; for example, they set their own tax rates. National decisions often require citizen referendums or cantonal approval, such as COVID-19 responses.

Basically, these are three neighboring countries with three different governance models. France follows a centralized model, Switzerland has a decentralized or distributed model, and Germany embraces a federated governance model. This is a perfect analogy for the data governance models that you could implement in your organization.

Centralized Data Governance

Centralized data governance is a top-down approach, similar to that of France, in which a central authority, such as the IT department or a CDP team, controls all the policies, standards, and decision making. This approach typically leads to a uniform set of policies across the organization and simplifies the compliance journey. However, it also results in the central team becoming the bottleneck and not having enough flexibility for domain-specific needs. This approach is typically adopted by highly regulated industries, as their priority is regulatory compliance.

Centralized data governance can be implemented with Unity Catalog, where the central team would have to create and manage all the Unity Catalog securables up to the leaf objects of the Unity Catalog object hierarchy. In other words, Databricks users will have access only within the schemas and very limited privileges. At the workspace level, the central team could create and provide workspaces and workspace-level objects, such as clusters and SQL warehouses, and grant permissions to the teams to use them.

However, it can be challenging to enforce centralized governance because it is difficult to control all aspects of the platform. For example, if a user creates a securable in Unity Catalog, they automatically become its owner and share it with any other user. If you have a central policy that forbids unapproved sharing of assets, it is not possible to enforce that policy. Some of the limitations around central policy enforcement should be addressed by the upcoming ABAC feature. There are also other challenges. For example, the Data Definition Language (DDL) and Data Manipulation Language (DML) permissions are combined into the `MODIFY` privilege. So, it is not possible to allow users to just insert into a table without allowing them to update the schema of the table. Therefore, if you have specific policies that allow DML operations but not DDL operations, it is not possible to enforce them.

For other aspects of governance, such as audit log analysis, controlling feature enablement, and so on, you can control them using the admin roles, and it perfectly fits into the central governance model.

Distributed Data Governance

In a *decentralized* model, authority is dispersed across autonomous entities without a central hierarchy, similar to the Swiss model. In this model, entities within an organization are fully independent in terms of policies, standards, and execution. This is the most flexible option and is potentially suitable for tech startups prioritizing innovation. Compliance can be hard because of the lack of a central authority. Of course, you can be successful with a distributed data governance model, provided that you have self-sufficient teams capable of securing the platform and assets from internal and external threats. The concern with a decentralized approach is the lack of central oversight and duplicated efforts across entities. Organizational structures are usually hierarchical. No matter how autonomous the teams are, and how flat the hierarchy is, you will have a boss, and your boss has to report to their boss, on up until the CEO. Therefore, a lack of central oversight on the data governance aspects where data strategy is crucial for the organization's success can be a problem.

You can implement distributed data governance with Unity Catalog to a certain extent. At the metastore level, you can delegate most of the privileges to the respective teams so that they can create and manage all the securables. Each team can also create its own workspaces and independently manage workspace-level objects. However, a

fully distributed setup is also not possible, as there are central admin roles. Metastore admin, being optional, can be removed, but you need the account admin who needs to perform a few activities every now and then. Although most of the activities that need account admin privileges can be automated, you would still need an account admin for situations such as handling escalations, enabling the latest features at the account level, and so on.

Federated Data Governance

Federated governance is a hybrid model that combines centralized oversight with entity-level autonomy. Like the German governance model, there is a nice balance of power between the central body and the local entities. Some of the policies, especially around interoperability and compliance, are set at the central level with local implementation. Each entity or domain will have autonomy over aspects such as access controls, data quality, pipelines, and so on. This approach is suitable for large organizations since it can scale well. You can implement this model of governance with data architecture patterns like data mesh. This model fosters innovation while facilitating regulatory compliance. Hence, this is our favorite.

Unity Catalog is best suited to implement the federated data governance model. Similar to the distributed model, each entity can create and manage Unity Catalog securables as well as Databricks workspaces on their own. The central team could provide a template for the catalog design as well as workspace setup. They can define security best practices such as the use of storage firewalls, policies for preventing data exfiltration, granting only certified employees access to sensitive information, and so on. The central roles such as metastore admin and account admin can play a crucial role in enforcing the central policies and guidelines. Moreover, the central team could have oversight by monitoring all the events using the system tables, information schema, and lineage data—all of which we will discuss in Chapter 7. You can also set standards for interoperability and data sharing and leave the implementational details to individual entities.

Governing a country is very different from governing data, though. While the governance structures of France, Germany, and Switzerland offer illustrative parallels to centralized, federated, and distributed data governance, respectively, it's important to recognize the distinct challenges and contexts of data management within organizations.

The efficacy of a particular governance model at the national level does not necessarily translate directly to organizational data governance success. For instance, Switzerland's prosperity under a highly distributed system does not automatically imply that a distributed data governance model would be optimal for all organizations. Each organization must carefully evaluate its unique needs, regulatory environment,

organizational culture, and strategic objectives when selecting a data governance approach.

The following sections discuss the design of physical and logical storage layouts and data sharing from the perspective of the federated data governance model.

Data Storage and Distribution

One of the first steps you take when you start using Unity Catalog is to define the data layout, that is, structure the catalogs and schemas and organize your data and AI assets. It is of course different from how you would have done it on an RDBMS or on most of the data warehouses. On an RDBMS, you need to be concerned only about the logical data layout because the storage layer is completely abstracted away by the system. That is the case with most of the data warehouses. Therefore, when it comes to designing the data layout, you just need to do it on the logical layer, that is, defining databases and data assets below them.

With Databricks Unity Catalog, you have the option to design the layout of your storage layer as well. In other words, you can choose the cloud object storage units, such as S3 buckets, where you want to store your data and AI assets. However, in practice, there might already be an infrastructure-driven storage layout to which you might have to adhere to. For each of these object storage units, you need to configure the data access by defining the storage credentials and external locations.

For the sake of simplicity, we will refer to the cloud storage layer and the data access configurations objects together as the *physical storage layer* and the catalogs, schemas, and other securables below the schemas as the *logical storage layer*. We have depicted this in Figure 5-16.

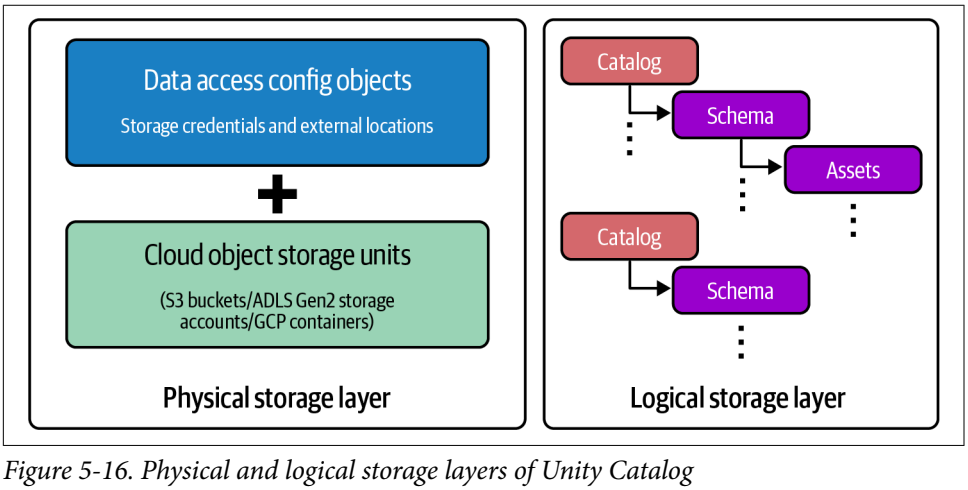


Figure 5-16. Physical and logical storage layers of Unity Catalog

If you're starting from scratch—that is, if you have just started with your journey with Databricks and planning to design your data layout—then it's best to start with the design of the logical storage layer. You define how you want to structure your catalogs and schemas, and it will inherently map to a physical storage layer. We will elaborate on this in a bit. On the other hand, if you have already been using Databricks and are now migrating to Unity Catalog, you might want to consider the existing data layout—that is, the existing data files stored in the cloud storage and how it is registered in the HMS, to minimize the migration efforts.

There are various factors that you need to take into account while designing the data layout. The main factors are organizational structure and policies, domains, data sources and processing requirements, and data consumption patterns. Properly organizing your data and AI assets can simplify the ownership model as well as significantly enhance operational efficiency by minimizing data movement and data duplication. It would also make it easier to enforce centralized organizational policies. Although you should consider as many factors as possible, you cannot be perfect. Therefore, it is important to make sure that you do not end up spending too much time designing the perfect layout. You can in fact change or update the design in the future, although this would need additional effort. Next, we'll cover some guidelines and best practices so that you have a head start on the design phase.

Catalog Layout and Nomenclature

Should you have one catalog per business unit or one per project? Or just have a catalog per operational tier? How many schemas and tables can you have in a catalog? There are quite a few questions that occur when you start designing the catalog layout. Thinking back to the RDBMS systems, we had the databases and tables or views within them. The idea of the database is to group together all the related tables and views. In Unity Catalog, you have yet another hierarchy level. This gives an additional dimension to work with. However, the idea is the same: to group related assets.

As mentioned before, the design should be a reflection of several factors, such as organizational structure and policies. You can have a catalog or a set of catalogs for a business domain, a data domain, or even a project. The deciding factor is how big is this unit? In other words, the volume of data, the number of users who will be accessing the assets within the catalog, the data access patterns, and so forth. Irrespective of which unit you choose, the naming pattern of the catalogs should be uniform.

Now let's look at the most common patterns of catalog layout within an organization. We will use the term *organizational unit (OU)* to indicate a unit within an organization that is independent and large enough to command a dedicated catalog. This can be a project, team, data domain, business domain, or business unit. In this

chapter, we focus on the data storage and distribution within a single Unity Catalog metastore. In Chapter 8, we will discuss the cross-metastore data distribution. From the governance perspective, we will focus on a federated data governance approach.

An important technical factor that has a major impact on the data layout design is whether you want to use managed or external tables/volumes. If you decide to go with the managed securables, it will simplify your design because you can strive to have a one-to-one mapping between the logical and physical storage layers by specifying managed locations at the catalog level. Of course, you can have external tables/volumes within a catalog where you have specified a managed location. However, that can result in a complex mapping between the logical and physical layers. Therefore, to keep things simple, we will discuss the catalog layout design only with managed securables.

The simplest catalog layout for an OU is illustrated in Figure 5-17. The details of this setup, along with other related Databricks and cloud entities, are as follows:

- Three dedicated catalogs for the OU, one per operational tier of the development lifecycle.
- Dedicated cloud object storage units to store all the data and AI assets that belong to the OU.
- A managed location pointing to a dedicated cloud object storage unit is specified at the catalog level.
- Optionally, bind these catalogs to the OU's corresponding Databricks workspaces. We recommend this if you have different workspaces per environment.
- Dedicated OU account groups that are granted access to the OU workspaces and OU catalogs.

Following a federated governance approach, the admins within the OU should manage all the resources at the cloud level, such as virtual networks, workspaces, storage, and so on, as well as Unity Catalog securables. such as catalogs and data access configuration objects. Members of these organizational units typically get read-only or read-write access on the schema or asset level, within these catalogs.

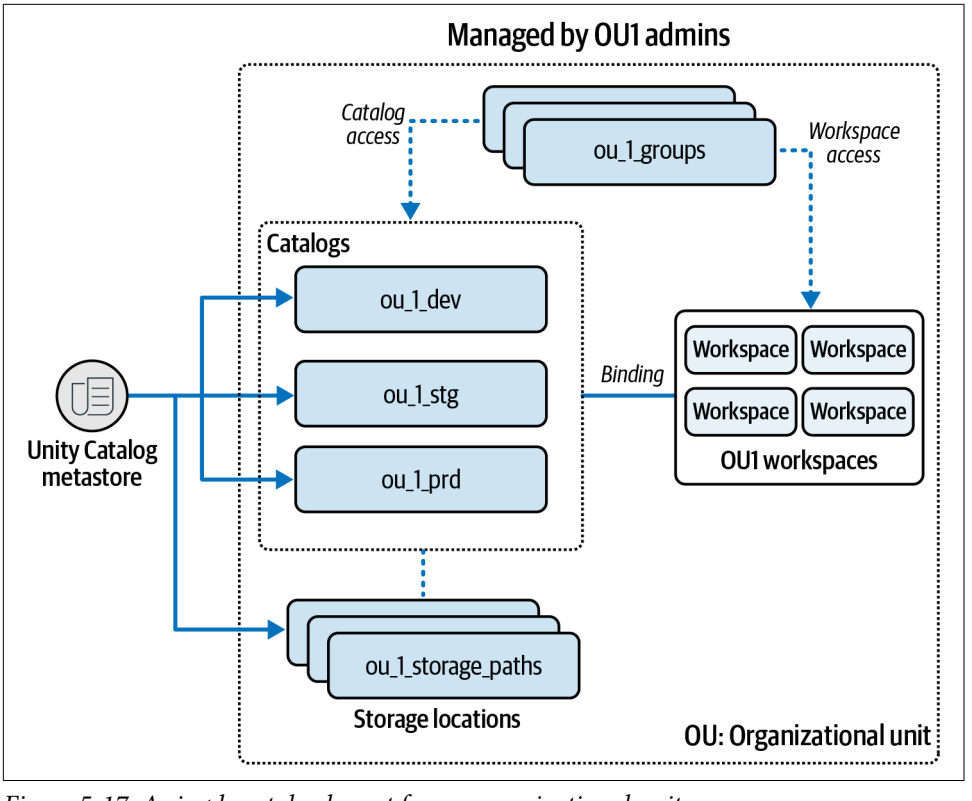


Figure 5-17. A simple catalog layout for an organizational unit

You can scale this catalog layout to a large number of OUs unless you hit the Unity Catalog limits on the number of catalogs or other securables. The layout is exactly the same if you have one or more OUs. Figure 5-18 shows the catalog layout for two OUs.

You might be wondering why we would incorporate the operational tiers of the development lifecycle into the catalog names? Why do we need to have a separate catalog for each operational tier? First of all, we should all agree that we need a way to isolate data and AI assets at operational tiers. With Unity Catalog, the best way to do it is at the catalog level. You might be thinking: why don't we do it at the metastore level and have different metastores for each operational tier? Also, why don't we isolate assets of OUs at the metastore level? We have come across these questions from a lot of Databricks users and customers, especially in large organizations. If you recall from our discussion in Chapter 3, in a given Databricks account, you can have only one metastore per cloud region. And why is that? Let us explain.

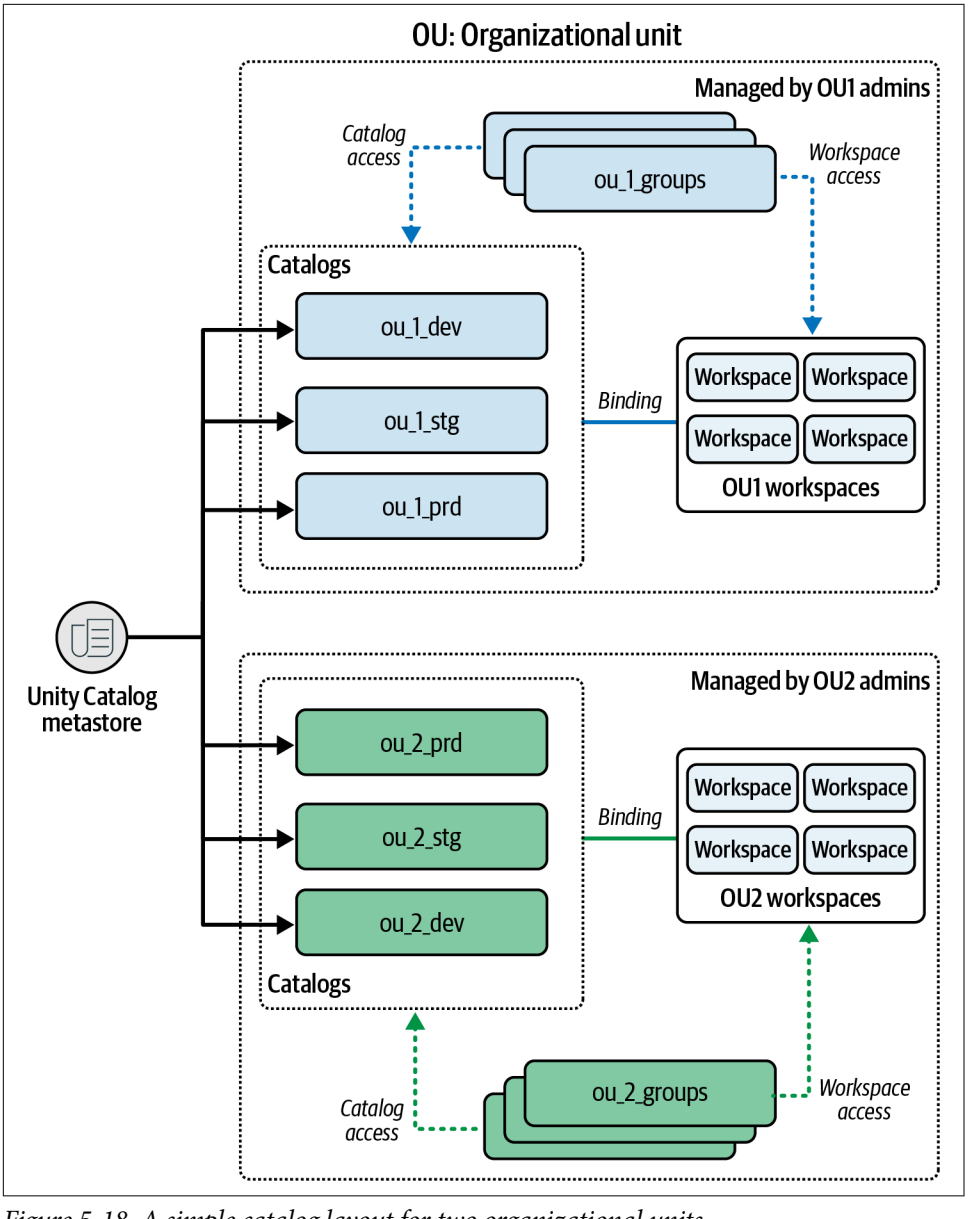


Figure 5-18. A simple catalog layout for two organizational units

What does a Databricks Unity Catalog metastore actually refer to? For a given Databricks account, a regional metastore can be thought of as an abstract entity. When you create a new metastore, it does not mean that you are creating a new instance of a resource. Rather you can think of it as creating a new logical entity that allows you to associate all the workspaces in that region and act as a metadata layer for

all the data and AI assets. Moreover, a single metastore allows for simplified data sharing. Users with proper permissions can access all the data and AI assets cataloged in a metastore using any workspace that is associated with that metastore. This will become much clearer when we discuss data sharing and distribution next. Even if you somehow manage to get multiple metastores in the same cloud region, you end up increasing the complexity of your data landscape as you rely on different mechanisms such as Delta Sharing to share data across metastores. Therefore, to sum up, the best way to isolate the data and AI assets of OUs within a given geographical region and operational tiers of the development lifecycle is at the catalog level, all within the same regional metastore. The flexible three-level namespace hierarchy facilitates this and results in simple data sharing and distribution models.

You can also have a more fine-grained grouping of your assets by creating catalogs for subunits within an OU, especially when the OU is large and you need further isolation among the subunits. This means you will have more catalogs but better isolation. The catalog nomenclature should reflect this pattern. Figure 5-19 shows an example of such a catalog layout.

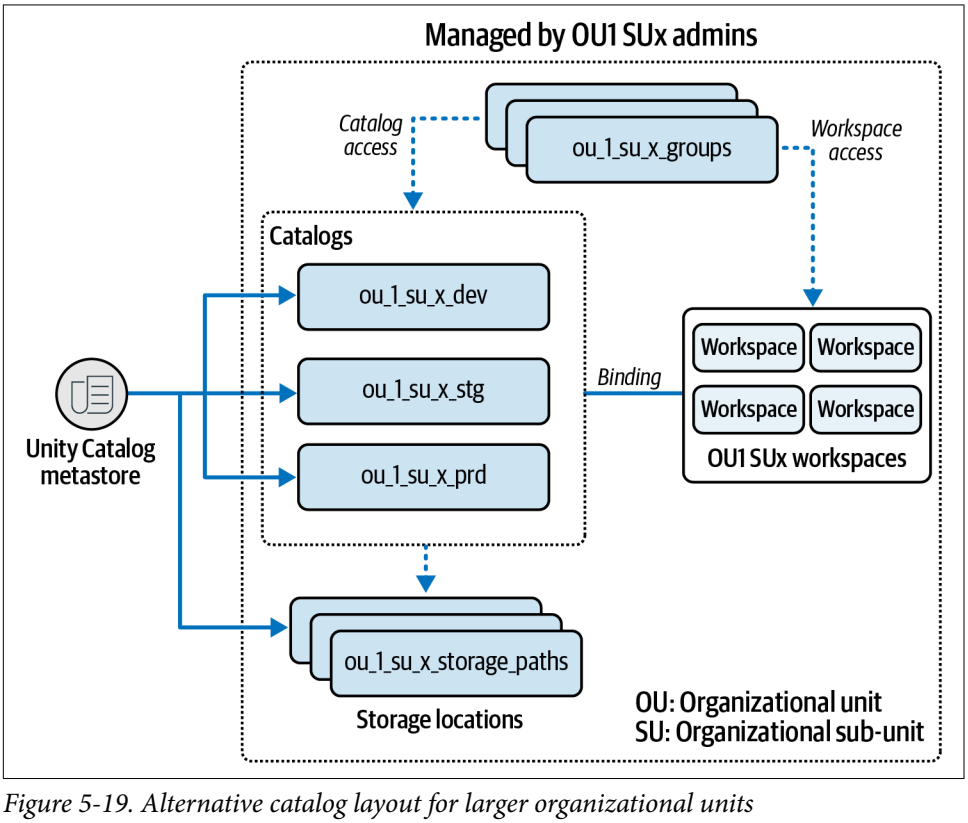


Figure 5-19. Alternative catalog layout for larger organizational units

We have seen a large variety of patterns among Databricks customers, ranging from unnecessarily sophisticated layouts on the one hand to overly simple setups on the other. Sophisticated layouts, unless well planned, can lead to increased maintenance, especially in cases where you have a large number of catalogs but few cloud storage units. On the flip side, overly simplified layouts can mean insufficient isolation. Therefore, a good design to the catalog layout is of paramount importance.

We have also come across catalog layout design where the isolation of assets was done at the schema level rather than at the catalog level. In other words, you would have a common catalog and dedicated schemas for each OU. This is indeed an option, as you can also set managed locations at the schema level. However, isolating assets at the catalog level gives you the ability to restrict access to specific workspaces through binding, which is an added benefit from the security perspective. Therefore, this approach, schema-level organization, is suitable for small organizations, especially unregulated ones, that do not have any strict requirements for isolating assets.

Who Should Be the Metastore Admin?

A metastore admin has access to all the data and AI assets cataloged in that metastore. If you're sharing a metastore across several OUs, some of which can be different legal entities, it might be challenging to find the right individual or a group of individuals who should take on the role of the metastore admin. This is indeed a challenge for large organizations.

First, since metastore admin is an optional role, you could simply remove it from the metastore. However, based on our experience, sometimes it can be convenient to have a metastore admin for activities such as handling escalations, central operations, monitoring, compliance checks, and so on. The role is also beneficial in cases where the operational admins involved in metastore creation may not have any purview over the data space.

If you decide to retain the metastore admin role, the following are the five main options to fill the role:

- If there is a significant presence of Databricks in the technology landscape of your organization, then it might be worth investing in a dedicated team that would assume crucial admin roles. This can be a new:
  - — Databricks Platform operations team
  - — Databricks center of excellence (CoE)
  - — Databricks governance team
- An existing admin team could assume the role of the metastore admin, usually a team such as:
  - — The cloud team

— Identity management team (for example, Entra ID admin team)

— Platform operations team

— Data governance team

- A virtual team consisting of individuals from different OUs who would represent the interests of their unit.
- The metastore admin role could be assigned to an empty group. An individual is elevated to the metastore admin role by temporarily adding them to the group and revoking access once the required activity has been completed or the permitted time is over. This could be seen as a "break glass procedure" and is also good for preventing accidents.
- An SP is given the metastore admin role, and all the activities at the metastore level are automated via APIs or IaC scripts. Furthermore, the automation workflow can be integrated with the internal IT ticketing system of the organization.

In all cases, the metastore admin role should be assigned to a group instead of an individual. Moreover, every action performed by a user as a metastore admin is logged. Therefore, using any of the options we've mentioned here and the audit logs, you can keep the powerful role of being a metastore admin secure and accountable.

Data Sharing and Distribution

So far, we have discussed the catalog layout and asset access per OU. In most cases, however, you would want to share the data and AI assets that you create within an OU with other OUs. In this section, we will discuss different ways to achieve this.

In-place publishing

The simplest way to share the assets with other OUs is by granting direct access to those assets. Typically, you would restrict this access to read-only. Say you want to share a table. In terms of Unity Catalog privileges, this would mean you grant the `SELECT` privilege on the table along with `USE CATALOG` and `USE SCHEMA` on the corresponding catalog and schema. If you have bound your catalogs to corresponding OU workspaces, then you have to bind your catalog to the workspaces of the OU that you are sharing the data with. Usually, this binding should also be read-only. If we refer back to the catalog layout for two OUs, the asset sharing would be like that in Figure 5-20.

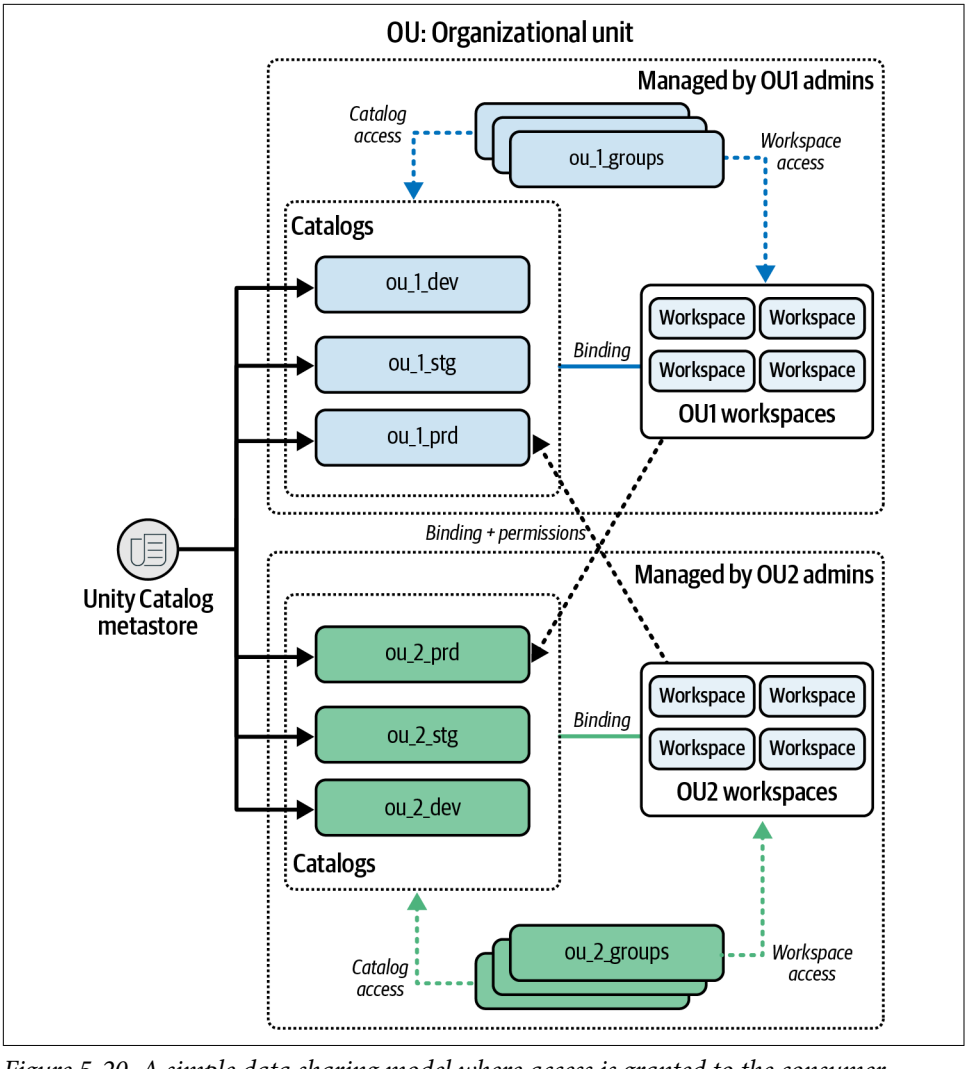


Figure 5-20. A simple data sharing model where access is granted to the consumer directly on the production catalog of the source OU

Dedicated publishing catalog

You may be thinking that granting direct access to a `production` catalog to other OUs isn't the best idea. Or binding your `production` catalog to workspaces outside of your OU might not be something that you or your peers favor. If that's the case, you can consider having a separate catalog dedicated to publishing the assets that you intend to share. You grant relevant permissions to the consumers on the `published` catalog and bind their workspaces to it. Figure 5-21 shows this approach for data sharing.

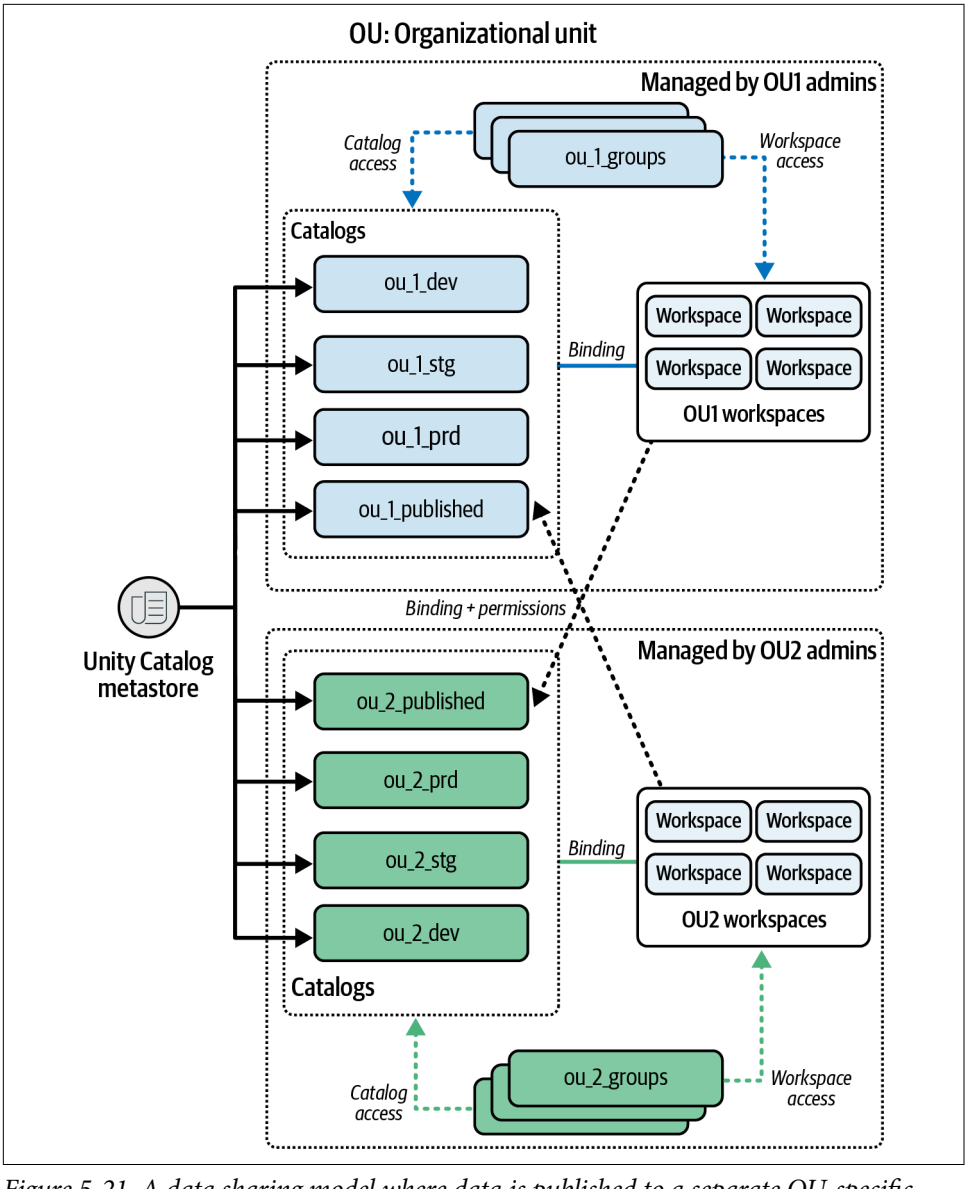


Figure 5-21. A data sharing model where data is published to a separate OU-specific "published" catalog and access to the consumers is granted on this catalog

Whenever we discuss this approach, a common question we get is: should we copy the assets from the `production` catalog to the `published` catalog? The answer is no. The idea here is to directly create/update the assets that you wish to share, directly in the `published` catalog. The publishing process using the medallion architecture

is shown in Figure 5-22. An example approach of medallion architecture that we discussed in Chapter 1 has different layers, namely bronze, silver, and gold. Usually, you share the assets in the gold layer. Therefore, in terms of orchestrating the data transformations, you create the bronze and silver layer assets in the `production` catalog and the gold layer assets directly in the `published` catalog.

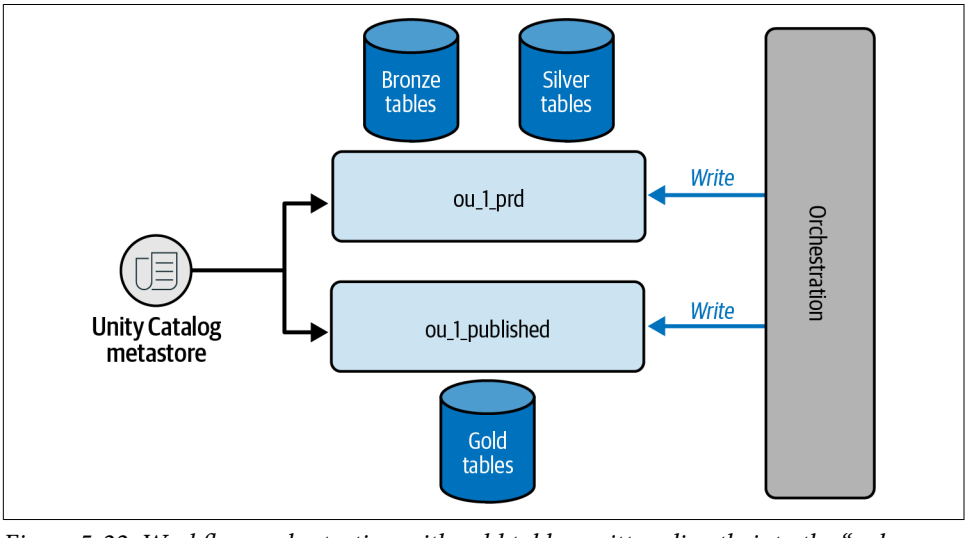


Figure 5-22. Workflow orchestration with gold tables written directly into the "published" catalog preventing data duplication

This setup can be termed *distributed publishing* because every OU creates and maintains a `published` catalog. It is best suited if you do not have any CDP teams or if the OUs have very specific assets that are of interest to only one or a couple of other OUs. On the other hand, if you have a CDP team and if there is a significantly high interest and practice of sharing and consuming assets across OUs, a centralized publishing approach is more suitable.

Centralized publishing

In this approach, you create and maintain a central `published` catalog that is external and independent of all the OUs—all the OUs willing to share the data publish their assets to the central `published` catalog. Publishing to a common catalog can mean each OU having write access to the catalog, but this can be hard to sustain. Therefore, an alternative approach would be to dedicate one or more schemas to a particular OU where it has write access, `MODIFY` privilege to be specific, and grant selective access to other OUs. In addition, all the OUs should be granted the `BROWSE` privilege on the `published` catalog, which facilitates the discovery of assets. The potential

consumers can then search, discover, and request access. Chapter 7 goes deeper into discoverability.

Figure 5-23 illustrates the centralized publishing model. Note that, similar to the previous setup, publishing the assets to the central catalog does not necessarily mean that you copy the assets from your `production` catalog to the `published` catalog. You can simply create the assets directly in the central `published` catalog. In addition to the OUs, if the central team also acts as a source of assets that it intends to share with other OUs, then it can publish those assets to the same `published` catalog.

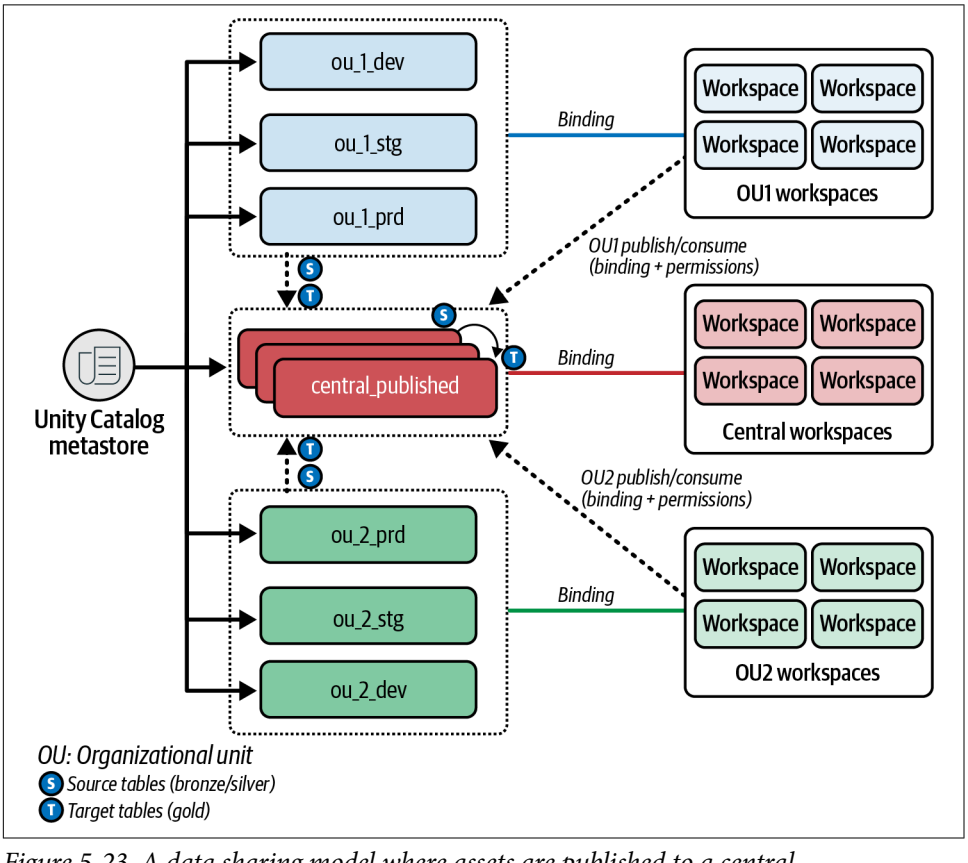


Figure 5-23. A data sharing model where assets are published to a central "published" catalog

The advantages of centralized publishing over distributed publishing are as follows:

- Individual OUs do not have to maintain a separate `published` catalog and to a certain extent save storage costs as you will be creating and updating the assets directly in the central `published` catalog managed by the CDP team.
- Since a central catalog independent of the OUs holds all the published assets, it is relatively easy to search and discover those assets.

However, on the flip side, if you don't provide clear guidelines and impose strict restrictions, the central catalog might end up becoming a dumping ground. Therefore, it's important to have proper processes to publish the assets.

We've discussed various catalog layouts and data sharing models. But to be honest, we haven't seen anyone implement the exact same setup, and we don't believe anyone will in the future. These are ideal setups and, of course, you can strive for them, but in the real world, it's really hard to enforce such clear and clean boundaries. You will end up creating way more than just three catalogs per OU. You will have more than one `published` catalog. You will at least have a hybrid approach to everything that we discussed and more, and that is alright as long as you have clear responsibilities and ownership of assets as well as robust security practices such as the principle of least privilege.

If you're a data architect or a platform owner, you'll see that the complexity of managing the assets will rise significantly with the rapid increase in the volume of data, which in turn results in more catalogs and other securables. This is typically followed by an increase in the number of users who want to have access to everything. The best way to deal with the complexity of managing the data and AI assets at scale is to treat them as products. Establishing a framework for data products with data contracts and certification processes can help build a clean and sustainable practice for data sharing and collaboration. Unity Catalog's data layout options, governance features, and sharing capabilities are well-suited for developing data products and implementing data mesh-like architectures.

Bringing It All Together

Designing the data architecture in the form of physical and logical storage layout and creating a framework for permissions and ownership model for your data and AI assets can be a daunting task, but it is crucial. Your data and AI assets are treasure that you need to protect from external and internal threats. You can leverage the various access control options available within the Databricks Platform to design and implement a robust and scalable governance model. You might have to spend some time discussing and iterating on the design to get it right, but it's worth it.

Take the example of Nexa Boutique. Before Unity Catalog, the CDP team controlled most of the data and AI assets and how these were consumed. This meant Nexa was leaning more toward centralized data governance, although they did not have any formal processes for enforcing policies. If one team is in charge of ingesting, transforming, and providing data to the entire organization, then the governance, if any, cannot be distributed or federated. This was not by deliberate design. It was the result of organic growth.

If you're a startup, one whose revenue is not dependent on selling software, you would not be worried too much about the data architecture or technical debt. You'd focus on growing and expanding your core business, and everything else just grows with it. So was the case with Nexa Boutique. The one data platform did everything from the early times of the company until it grew into a multinational organization. Then when they had to migrate to Unity Catalog, they saw an opportunity to revisit their data architecture.

Nexa decided that it was time to modernize its approach to data and related assets. The CDP team was tired of being the bottleneck and having to deal with a negative image within the company. They became like the political party that gets most of its votes from the older generation and are vehemently hated by the young folks for not being modern and not catering to their needs. The CDP team along with representatives from multiple business domains discussed, deliberated, consulted with Databricks and its partners, and came up with a data mesh kind of architecture with several data domains and a central hub for publishing some of the commonly used assets. It was a hybrid topology that combined a harmonized data mesh with the hub-and-spoke model. Moreover, they replicated this approach across different geographies. Figure 5-24 depicts this hybrid data mesh architecture.

In the harmonized topology, data domains create and publish domain-specific assets. These assets are consumed directly by other data domains. In the central hub topology, the data domains create and publish assets to the central hub from where other data domains consume them. In the hybrid topology adopted by Nexa, some of the assets are published within the data domains and some assets that are deemed useful for several other data domains are published to the central hub.

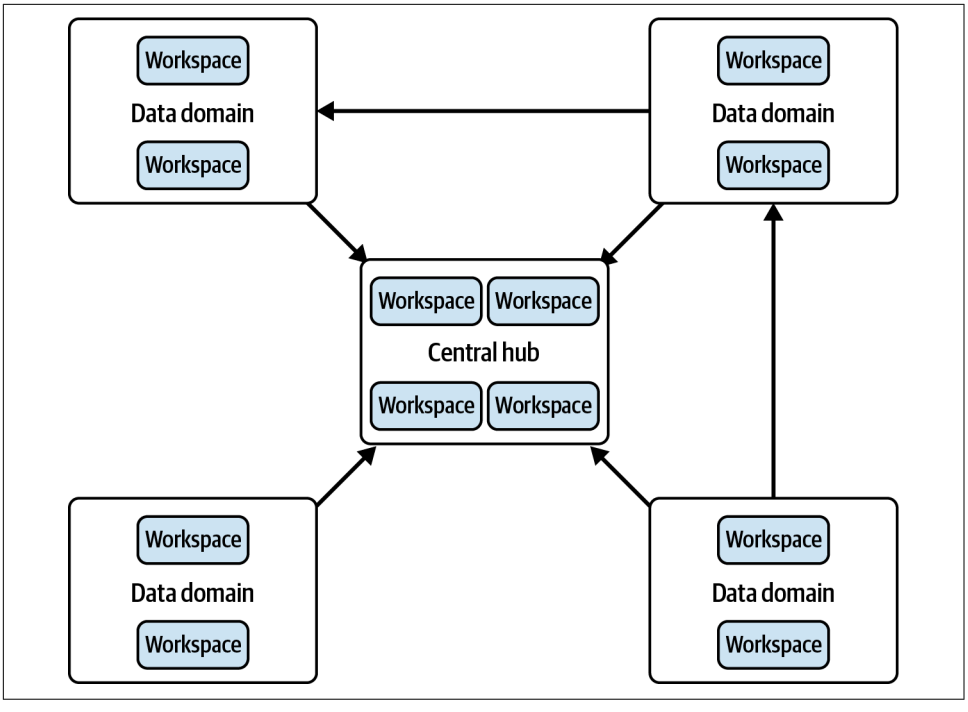


Figure 5-24. A hybrid data mesh topology with a central hub and harmonized publishing of data and AI assets

In terms of the catalog layout, each domain consists of one catalog per operational tier (dev/stg/prd) and a `published` catalog. The central hub managed by the CDP team also has the same setup. The major difference in the new architecture is that each data domain, which maps to the corresponding business domains such as sales and marketing, will ingest and transform data from the sources they are closest to. For example, sales teams use Salesforce extensively and therefore the data domain corresponding to sales will now ingest the data from Salesforce, transform it, and create data products. The data products closest to the source are termed *source-aligned* data products. Each data domain will create source-aligned data products that are published either within the domain-specific `published` catalog or the `published` catalog within the central hub. The central hub will continue to ingest data and process the data from nondomain-specific data sources. Figure 5-25 shows this setup along with the catalog layout design.

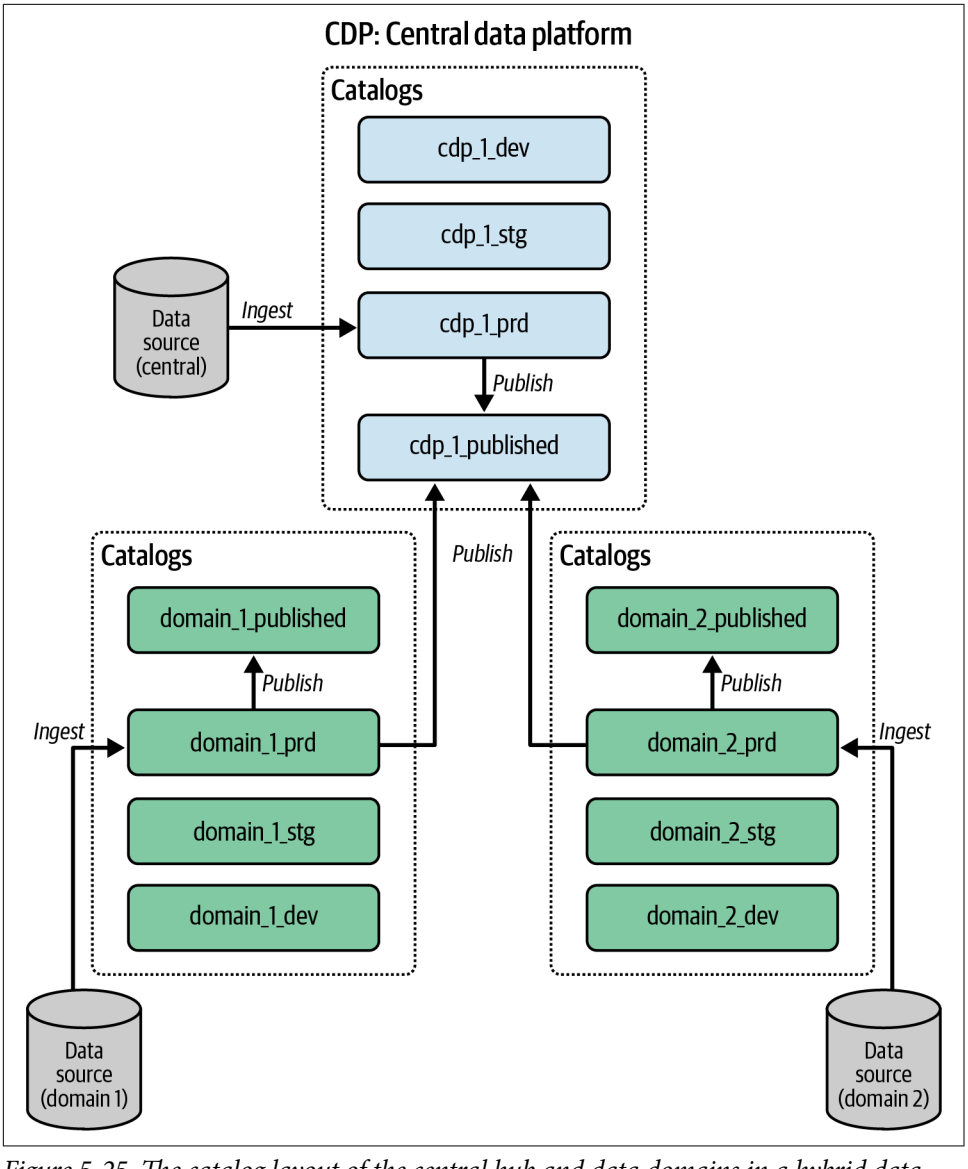


Figure 5-25. The catalog layout of the central hub and data domains in a hybrid data mesh topology

Nexa always had a security-first design approach and the company continued with it while designing the permissions model for the data and AI assets within Databricks. Following are some of the key design decisions and practices that Nexa employed:

- Everyone should get only the minimum set of privileges so that they can perform their job efficiently—no more and no less. This is the principle of least privilege.
- Nexa also followed a zero-trust model. Even if someone is authenticated into the system, they will not have access unless explicitly granted.
- Admins in the central hub as well as the data domains create the first-level objects in the metastore object hierarchy: catalogs, storage credentials, and external locations.
- The users get privileges only within the catalog and sometimes even only within the schemas.
- By default, no human has access to production workspaces or catalogs. SPs do all the processing in the production environment. A human would be granted access only to change configurations or debug issues.
- Only managed tables/volumes are allowed by default in production and published catalogs, and external tables/volumes are only allowed as exceptions.
- Access is always granted to account groups and never to an individual.
- A group consisting of SPs owns all the securables in production environments.
- Even at the workspace level, users were given only necessary privileges such as `USE` on a `SQL Warehouse` instance and `CAN RUN` on a `Query` object.

Each domain had the autonomy to manage its workspaces and Unity Catalog securables. There were, however, some guidelines and policies that we centrally curated and sometimes enforced. Nexa also decided to have a metastore admin, which was a group consisting of individuals from CDP as well as each data domain. There were many other decisions that Nexa took to solidify its data and AI security and governance, some of which we will touch upon in the following chapters.

Summary

In this chapter, we saw that applying proper access controls both at the workspace level as well as Unity Catalog metastore level is crucial for securing your data and AI assets. We also explored the catalog layout design approaches and discussed how best to share your assets. These approaches are applicable to organizations of all sizes. And we took a look at how Nexa Boutique completely revamped its data architecture and turned the journey of migrating to Unity Catalog as a way to modernize its data landscape. In Chapter 6, we will explore ways to secure and govern AI, which is a relatively new topic.

CHAPTER 6
Governing AI

Technology is morally neutral until we apply it.
—William Gibson

AI-powered has become the most popular adjective across organizations and products. Much of that is pure hype, certainly, but it is true that more companies are indeed trying to leverage their data and build something useful using the latest AI advancements. Just like the digital transformation, the AI transformation is real, and it is happening.

The biggest challenge, however, is to build an AI-based system that is fully governed and secure. You might have experienced this firsthand: AI projects, even simple, classic ML-based ones, can be difficult to productionize. That's often because the typical origins of such projects trace back to the laptops of data scientists or ML/AI engineers, who are normally more focused on mathematical and statistical aspects of the project and less focused on how to deploy them. In most AI projects, security is typically an afterthought, and governance aspects are nonexistent. Moreover, with the advancements in generative AI (GenAI) offerings, notably GPT models from OpenAI, new security and governance challenges have emerged.

Some of these challenges seem comical. On January 18, 2024, a user on X (formerly Twitter) posted his conversation with an AI-powered customer service chatbot from parcel delivery company DPD. The chatbot, in addition to not being as helpful as the user expected it to be, when prompted, generated a poem on how useless it is and how terrible DPD is. This incident, after it went viral on social media, resulted in DPD immediately taking action to rectify the AI element, but the damage to its reputation couldn't be reversed. A more serious incident from 2023 took place at Samsung when one of its employees posted sensitive internal source code to OpenAI through ChatGPT, which resulted in Samsung banning the use of GenAI tools among

its employees. These are just two of many similar incidents that are attributed to the rapid adoption of AI across various organizations.

With the latest trend of using generic, general purpose GenAI models often hosted by third-party vendors, the surface area for potential security issues for organizations expands—for example, organization-specific data being uploaded to third-party model API providers. This vulnerability underscores the need to think about securing and governing systems at different levels: the application, the network, and the infrastructure layers. For example, on the network layer, you need to consider cross-region data transfers required to use a model hosted in other cloud regions, which might not be allowed in some cases for compliance reasons. Even within an organization, it can be hard to control who can access trained AI models, how to reproduce ML experiments and results, and so on.

The lack of proper data governance significantly impacts the sustainability of AI projects, a fact that surprises many organizations. This is evident by the high failure rates in AI projects. As per Gartner, GenAI projects are often abandoned after the proof-of-concept phase due to poor data quality, lack of clear business value, insufficient risk controls, and increasing costs. Gartner predicts that by 2027, 60% of organizations will fail to realize the anticipated value of their AI use cases due to incohesive ethical governance frameworks.

Due to rapid advancements in AI, many bigger concerns have been raised, such as job displacement, increasing economic inequality, bias, discrimination, misinformation, and more. Addressing these issues at the organizational, governmental, and societal levels is crucial.

What Is AI Governance?

As AI is increasingly integrated into critical aspects of business operations and daily life, an essential requirement for organizations and societies is to manage both the opportunities and risks associated with it by establishing robust governance mechanisms. *AI governance* encompasses the policies, structured frameworks, and practices designed to ensure that AI technologies are developed and deployed in safe, responsible, and ethical ways. The scope of AI governance is broad, and as with data governance, there is no global central authority that decides its boundaries. It is influenced by policy, procedure, and framework trends and limitations, each playing a significant role at different levels—international, governmental, and industrial.

International entities set principles and global standards that influence national and regional policies. For example, the United Nations (UN) has its AI Advisory Body, and the Organisation for Economic Co-operation and Development (OECD) has developed its intergovernmental AI Principles. The EU is shaping AI regulatory approaches worldwide through its EU AI Act. Major technology companies such

as Microsoft, Meta, Google, and standards bodies such as the International Organization for Standardization (ISO) with ISO/IEC 42001, are highly influential in operationalizing governance frameworks and best practices.

At its core, AI governance represents a multidisciplinary approach that combines technical, legal, and ethical considerations to create a structured environment and frameworks for developing and using AI technologies. This approach applies to both AI system providers and users. It consists of processes, standards, and guardrails designed to ensure that AI systems operate safely, ethically, and in compliance with relevant regulations. It encompasses the entire AI lifecycle, from development and deployment to ongoing monitoring and management of AI technologies.

This chapter focuses on only certain aspects of AI governance, especially on the implementational details in the context of the Databricks Platform. We will dive into the governance aspects of AI project components such as training data and models, model serving, guardrails for accessing and using AI models, and observability. We won't discuss the legal, ethical, and societal aspects of AI governance that typically fall under the category of responsible AI.

AI Model Lifecycle

The most important artifact of an AI project is the model, which is trained using data. Therefore, in very simple terms, a *trained model* is a representation of the dataset it has been trained on. Moreover, if we consider an AI use case, the input and output of a model are data. Therefore, AI governance is basically a part of data governance. Figure 6-1 shows the elements of an AI model.

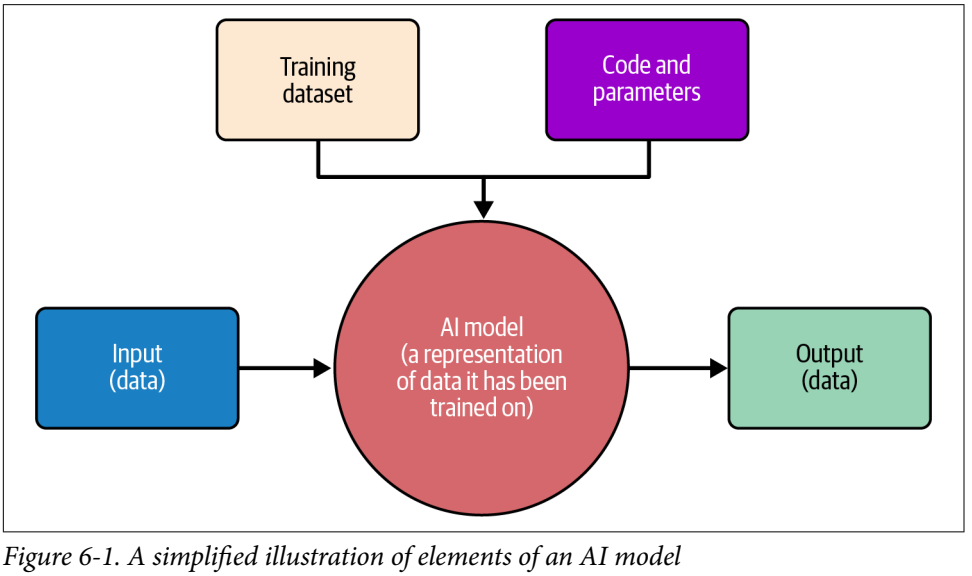


Figure 6-1. A simplified illustration of elements of an AI model

An AI model's lifecycle has two main phases: the *training* phase, where the model is created, and the *serving* phase, where the model is used. The training phase is relatively simple for classic ML, but with GenAI it is much more complex. In this book, we focus mainly on the simplified concepts of ML/AI. The complexities and nuances of the AI model training are out of scope for this book. For more details on model training, you can refer to this book, *Designing Machine Learning Systems* (O'Reilly) by Chip Huyen.

To keep things simple, we will refer to all kinds of AI-related projects, including classic ML and GenAI projects, as AI projects. Moreover, the entire chapter is focused on the security and governance aspects, so we will treat most AI concepts on a very high level.

Model Training

In the training phase of an AI project, you have to mainly focus on the datasets that you wish to use for training. Within the context of the Databricks Platform, you ingest the data, be it structured or unstructured, and catalog it in Unity Catalog as tables or volumes. You can use Delta tables cataloged in Unity Catalog as feature tables. You can leverage the managed MLflow framework available as part of the Databricks Platform to build ML models and GenAI applications. The integrations of MLflow with industry-standard tools such as OpenAI and Hugging Face Transformers expand its capabilities to manage and deploy large language models (LLMs). We will dive into LLMs a bit later in this chapter.

The key advantages of implementing AI projects on the Databricks Platform are not just that it is much simpler than other data platforms, but also that you get security, scalability, and reliability. This is made possible mainly by the platform's security-first approach and unified access controls provided by Unity Catalog. That may not sound very important, but if you look deeper into security issues that are emerging out of the new GenAI paradigm, you will see that these aspects are significant, especially when it comes to productionizing AI models.

When we talk to Databricks customers and users who plan to implement an AI project using an external model, the biggest concern we hear is about organizational data being sent to a third party, such as OpenAI. Most users automatically assume that whatever they send to an LLM will be used for training purposes, which is sometimes true, but you often have the option to opt out of that through contractual arrangements with the vendor. Obviously, training or hosting your own LLM is far more secure than using a third-party hosted solution, because you're in charge. This might also make sense if you want to customize a model to your organization's specific datasets.

There are also certain aspects that you need to take into account while training or customizing an LLM. Some users tend to blindly use every dataset and file that they can get their hands on to train an LLM. The intention is to leverage all the data available within the organization to build a model that can be very helpful. However, that might not be such a good idea. For example, you need to make sure that you do *not* use any PII or other sensitive data for training LLMs unless explicitly allowed. If you blindly use anything and everything, some sensitive information might end up in the LLM, and from there lead to leaks. Imagine a file containing the salary amounts of individuals stored in a nonsecure SharePoint directory. Although it is already an issue that the directory is not secure, the impact of having such info in a SharePoint directory, which anybody might happen to look at, would be very different if it were used as training data for an LLM, which provides much easier access to information. Therefore, a platform that allows you to secure the end-to-end process of model development, deployment, and serving is necessary to prevent potential leaks.

Model Serving

Once you train a model or choose an existing one, your next step is to make it available for use. An AI model in its passive state is a file or a set of files, typically in a format understood by only the AI framework that was used to train the model. You can use the model to get the desired output using *batch inference*, also known as *offline inference*, by loading the model into memory and generating predictions for a typically large set of data at once, with high throughput. Examples would include periodically updating product or content recommendations and bulk image classification. Another way to get the output is *real-time inference*, where you load the model and generate predictions for individual data points with very low latency. Examples include obstacle detection during autonomous driving and fraud detection during credit card transactions.

In addition to serving a trained model for real-time inference, Databricks also allows you to use a model that is hosted outside of the platform. These are termed *external models*. An example of such an external model is Azure OpenAI. The way that works is that you create a model-serving endpoint in the Databricks workspace with an Azure OpenAI model being the backend that does the inference. You post the input to the Databricks model-serving endpoint, and Databricks forwards that to Azure OpenAI, gets the inference, and returns the output to you through the model-serving endpoint.

The advantage of this approach is that the users have a uniform experience irrespective of which model is being used in the backend. You could decide, for instance, to switch the model to use Hugging Face in the backend, and the UX wouldn't be affected. Moreover, you can leverage other platform-specific security features such as AI Gateway, covered later in this chapter.

Figure 6-2 shows the important security and governance aspects to be considered for an AI project. Note that all these are applicable end-to-end and not just at specific stages.

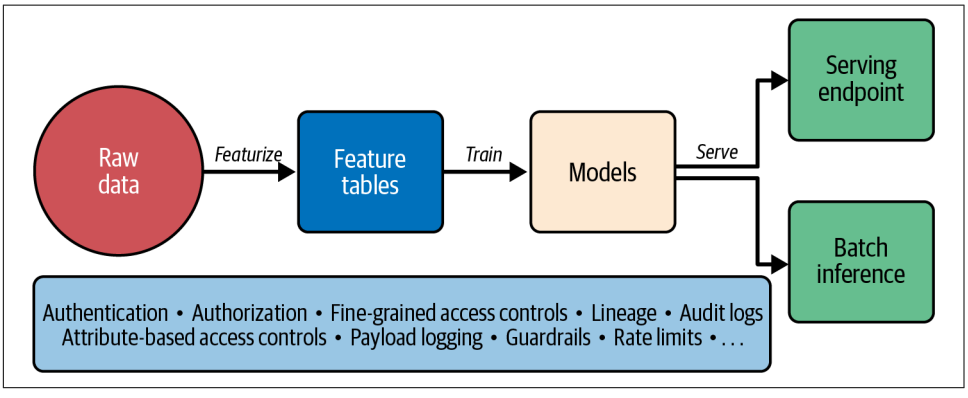


Figure 6-2. Simplified AI project elements and corresponding security and governance aspects to be considered

Governing AI Systems on Databricks

In Chapter 1, we discussed how silos within organizations can lead to different governance models. In the case of AI projects, this is exacerbated for two main reasons:

- The rapid growth of large-scale AI initiatives has outpaced the development of standardized governance models, resulting in fragmented approaches.
- Most platforms that provide a secure and governed framework for structured data do not provide the same for AI assets, namely files, functions, and models.

Our focus here is precisely on addressing these challenges with the Databricks Platform and Unity Catalog. Let's start with a simple example. Suppose you are an ML engineer who is part of a project called Detecting Nemo, where you need to build a model to detect a fish in images that looks like Nemo, the clownfish character in the famous 2003 Disney movie *Finding Nemo*. Your organization is using Databricks as one of its data platforms. Suppose you're lucky and you don't have to provision things on your own, since there is a platform team that sets things up for you. As an ML engineer who is part of the latest ML project, you get `ALL PRIVILEGES` on a newly created schema, `detecting_nemo` under the catalog `ml_projects_dev`, on which you get `USE CATALOG` permission.

You have a training dataset, which consists of a bunch of JPEG files and a CSV file that has a mapping of the image ID, which is part of the image filename, and whether Nemo is present in the image or not. The first thing that you need to do is

to ingest this dataset into the Databricks Platform. Since the dataset is fixed, for now, you have two approaches. The first approach is to use provider native tools to copy over all the files into cloud storage where you have an `EXTERNAL LOCATION` defined. You can then create an external volume, and you will be able to access all the files through Unity Catalog. Although this sounds like the obvious approach, there is a problem with it: the data ingestion part is not governed by Unity Catalog, since you need to have direct access to the storage account to copy the files. However, this is a suitable approach if you have very large files that you wish to ingest. If you choose this approach, it is recommended to make sure that you write files into the storage in an automated manner and make sure that only very limited principals have direct access to the storage.

The second approach, which is our preferred one, is to use the Files API to ingest the files into the Databricks Platform. In this approach, you create a managed volume and use the Files REST API with the volume as your target securable. The data ingestion part is also secure and governed. Now that you have your dataset, albeit in the raw state, cataloged in Unity Catalog, you can apply access controls to it. Note that, since your files are in a volume, you can apply access controls on only the volume itself and not on individual files within the volume. You can create a table from the CSV file that has the mapping. In this way, you will have a *securable*—that is, tables on which you can apply access controls—and you get lineage and other metadata associated with it. You won't have this metadata information on the file level within a volume.

In the next step, you train multiple models using the MLflow framework and ensure that they are persisted in your project's catalog and schema. You then evaluate the models and choose the best-performing one. For detecting Nemo in batches of images, you can create a function in Unity Catalog that calls the `prediction` function that returns the result using your best-performing model. Or you can spin up a model-serving instance that builds your model and serves it through a REST API endpoint, which you can use to detect Nemo in new images in real time. On top of that, you can build a Databricks Apps app where the user can log in with their Databricks credentials or SSO, upload an image, and check whether Nemo is present in that image.

We won't go into details of training, evaluation, and prediction because the focus of this book is not ML or AI. Our focus is governance, and in this simple example, we saw that the end-to-end process from ingesting your raw dataset until serving the model can benefit from the security and governance offered by the Databricks Platform. We illustrate this end-to-end process in Figure 6-3.

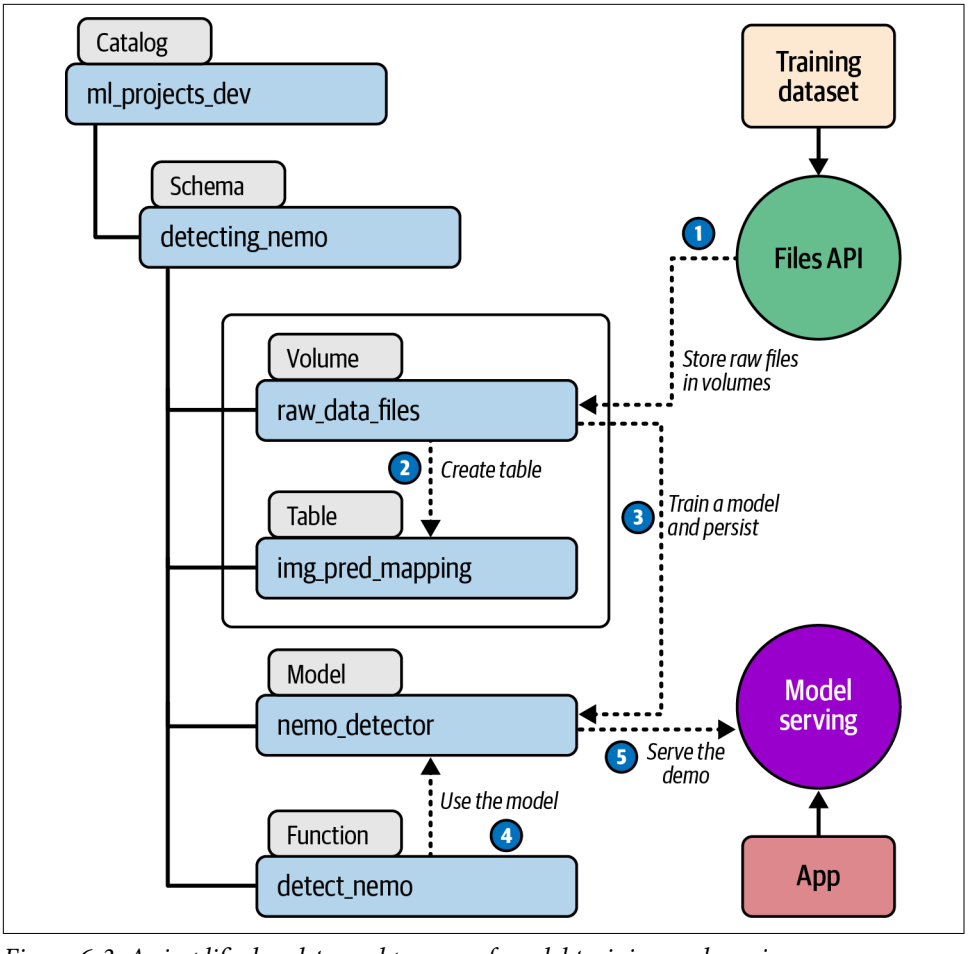


Figure 6-3. A simplified end-to-end process of model training and serving

MLOps

The need to deliver high-quality software products quickly, while rapidly incorporating feedback from stakeholders, has led to a software development methodology termed *DevOps*. It combines software development (dev) and IT operations (ops) into a single, collaborative approach and increases the reliability of software products through automation. DevOps is a reflection of the cultural shift in software development where the software developers and engineers take over the responsibilities of testing, deployment, and managing the infrastructure and platform to a certain extent. This is a result of the rapid rise of cloud platforms as well as technologies such as Kubernetes. Gone are the days when a software developer focused only on coding and there were dedicated teams for different stages of the software development lifecycle.

DevOps practices have been widely adopted and naturally, as ML projects entered the mainstream, the field warranted something similar. Typical software applications consist mainly of code, config files, and libraries or binaries that the code depends on. Therefore, in typical software projects, your primary focus is on version control, packaging, testing, and deploying the code along with the associated artifacts across different operational *tiers* or *environments*, known as *development*, *staging*, and *production*. By contrast, a significant part of ML projects is constituted by datasets and models. Therefore, the regular DevOps practices are not sufficient for ML projects. This has led to *machine learning operations* (MLOps), a set of practices and methodologies aimed at streamlining and automating the end-to-end lifecycle of ML models from development and training to deployment, monitoring, and maintenance in production environments. In essence, MLOps combines DevOps for code, DataOps for data, and ModelOps for models.

Proper implementation of MLOps helps you achieve the following:

Collaboration
: MLOps fosters strong collaboration among data scientists, ML engineers, DevOps, and IT teams to streamline the transition from model development to deployment and operations.

Automation
: It involves automating workflows for data preparation, model training, testing, deployment, and monitoring. This reduces manual errors and accelerates processes.

Continuous integration and deployment
: You apply CI/CD principles to ML, enabling versioning, iterative updates, and reliable rollouts of updated models.

Monitoring and governance
: You monitor the models in production and automatically detect any issues, such as data drift, and take measures to mitigate them. Compliance with regulatory requirements is also a key aspect, as you might need to adapt your processes, workflows, and so on for continued compliance with changing regulations.

Scalability and reproducibility
: The frameworks and processes you put in place for building ML solutions help you rapidly scale across the organization and ensure the reproducibility and auditability of models and results. The setup and practices should also scale in terms of data volume.

MLOps on Databricks follows a data-centric approach, and you can rely on the core platform features, such as MLflow and Unity Catalog, to achieve the key aspects mentioned in the preceding text. Let's look at how you can achieve each of these on the Databricks Platform:

Collaboration
: The collaborative features of the Databricks workspace, such as Notebooks, Git integration, and a hosted MLflow environment, help ML engineers and data scientists codevelop models within a workspace. Unity Catalog enables collaboration across workspaces by providing access to datasets and models.

Automation
: You can leverage Databricks Platform features such as *Lakeflow Jobs* to automate model development and deployment end-to-end. Furthermore, you can employ Databricks Asset Bundles (DABs), which is a templated IaC approach to managing complex projects on Databricks. It simplifies implementing the MLOps best practices on Databricks.

Continuous integration and deployment
: Git integration, model versioning, MLflow features such as experiment tracking, and other platform features enable CI/CD processes. DABs can further simplify the setup and maintenance.

Monitoring and governance
: Unity Catalog covers the governance aspects, namely, cataloging datasets and models, access controls, audit logs, and lineage. You can use Databricks Lakehouse Monitoring, a native platform feature, to monitor the statistical properties and quality of the data. You can also monitor model performance and potential drift by monitoring *inference tables*, which are basically logs of requests and responses from Mosaic AI model-serving instances, in the form of Delta tables.

Scalability and reproducibility
: Databricks Platform features collectively help you scale the ML solutions framework within your organization, and since Databricks is designed for processing big data, scalability in terms of data volume is inherent to the platform. Platform-specific tools, such as managed MLflow, and automation tools, such as DABs, simplify reproducibility.

Separate workspaces and catalogs are usually used to isolate operational tiers (aka environments). On Databricks, you can continue with the same approach for ML projects, with dedicated workspaces and catalogs for development, staging, and production environments.

In a typical software project, propagating code across environments is straightforward. You write the code in the development environment, test it in staging, and deploy to production. However, the propagation of data and AI assets across environments can be quite tricky, as it depends on various factors. For example, if you wish to train a model, in almost all cases, you need to use the production data. This means, if you are using the development environment, you need to access the production data in the development environment. Another example: if you train the model in

the development environment, should you propagate the training code to staging and production, since it is not used in these environments? You also need to think about how to roll out the models to production to accommodate additional requirements from the business, such as A/B testing and gradual rollout. An extensive deep dive into such MLOps implementation is out of the scope of this book. You can find detailed implementation steps and best practices recommendations in *The Big Book of MLOps*.

Large Language Models

In 2010, when Karthik started his career, the terms *machine learning* and *artificial intelligence* were niche and usually referred to neural networks, statistical models such as linear regression, logistic regression, and ensemble techniques such as random forests. *Natural language processing* (NLP) techniques were based on rules, statistics, and traditional ML and performed fairly well at structured, well-defined tasks such as sentiment analysis and text classification. However, they were inflexible and hard to scale. Today's LLMs are basically NLP models that are significantly more powerful and highly performant. They are primarily used in tasks such as open-ended question answering, summarization, text generation, code generation, and so on. The nature and number of use cases in which LLMs are used are increasing rapidly.

Although LLMs are ML models, the way they are trained and used is very different from traditional ML models. As their name implies, they are *large*, which means they are trained on significantly large volumes of data and require massive compute power, including GPUs. This requires a huge investment, so it does not make much sense for every company to train their own LLMs. Most firms end up using a generic LLM available from a vendor.

There are mainly two categories of LLMs:

Closed source
: Typically available for use by APIs and UIs hosted and managed by the vendors that provide them. Examples include models from OpenAI and Anthropic.

Open source
: These are models that anyone can download, update, and host themselves. Examples include Llama models by Meta and Mistral models by Mistral AI.

Most of these LLMs, closed source and open source, are being used extensively. However, these are generic models and naturally perform better on generic tasks than those where you need organization-specific context. To use an LLM for such a use case, you would either have to fine-tune an open source LLM with organization-specific data or train an LLM from scratch. The former is more feasible and cost-effective. Most security-conscious organizations also prefer to host LLMs themselves instead of sending data to a third party.

In the past, *model serving* meant that you instantiated a model that you trained and made it available for use via REST APIs. Now, model serving can mean:

- Serving self-trained or fine-tuned and self-hosted models
- Serving pre-trained and self-hosted models
- Serving externally hosted models

The first two are not new, but the third one is, and most organizations struggle with it as it is quite challenging to secure and govern the use of externally hosted models. There are various considerations:

Data security
: Data that is being sent to third-party vendors hosting the models should be secured with, for example, TLS 1.2+ encryption. If you wish to send sensitive data to a third party, special measures should be taken, such as field-level encryption.

Data retention policies
: Verify data retention policies of the vendor and choose the ones that guarantee zero-data retention policies.

Filtering inputs and outputs
: Preprocess the inputs to filter out any sensitive information—for example, PII and credentials—before sending it to the vendors. Postprocess output to redact risky content.

API governance
: Secure API keys by storing them in secret vaults and not hardcoding them in scripts. Enforce proper authentication and authorization, rate limiting, request logging, IP whitelisting, and firewalls to prevent malicious payloads. Enforce the principle of least privilege for using the APIs.

Operational governance
: Logging requests/responses and monitoring is crucial to detect anomalies such as unusual traffic and sensitive data leaks. You should also have an incident response plan in place, such as steps to revoke credentials, to quickly act in case of security breaches.

Vendor risk management
: It is vital to ensure that the vendor adheres to certain security standards and complies with regulatory requirements. You should also require them to provide third-party audit reports. Always negotiate terms for data retention, data ownership, breach notification timelines, and liability for compliance violations.

Of course, this is not an exhaustive list of what you need to do to secure and govern access to externally hosted models. However, it should give you an idea of how to think about and approach it.

The Databricks Platform has a model-serving service called Mosaic AI Model Serving. The term *Mosaic* comes from MosaicML, a company that Databricks acquired in 2023. You can think of it as a layer that acts as a generic interface to access both Databricks-hosted and external models, as shown in Figure 6-4. What we like about this Mosaic AI model serving is that it provides a uniform UI, allows for access controls, and leverages Databricks' core security and governance options. In addition, you can enable the Mosaic AI Gateway, which helps you manage API usage.

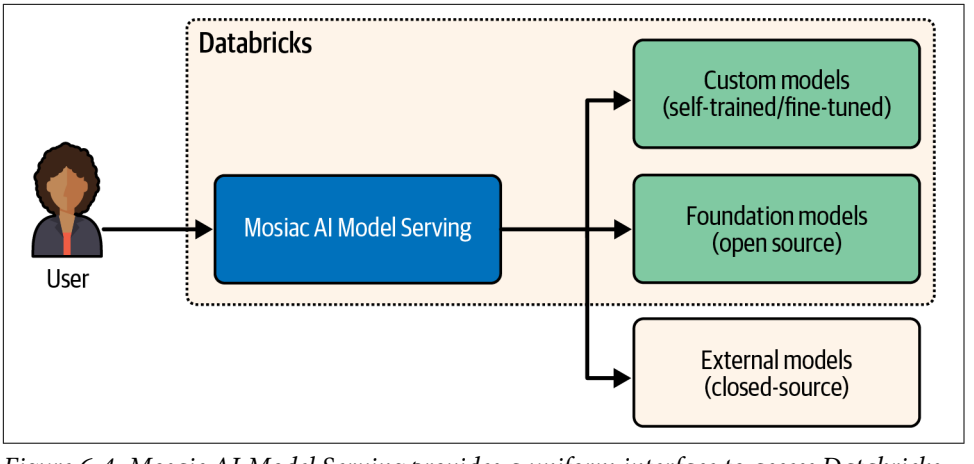


Figure 6-4. Mosaic AI Model Serving provides a uniform interface to access Databricks-hosted and external models

This trend of using common models through an API, embraced by the dawn of LLMs, has also affected MLOps practices, in which training a model is less frequent and the focus is on hosting/serving a model or using a third-party hosted model via APIs. Operational activities of LLMs are also sometimes referred to as *LLMOps*.

Mosaic AI Gateway

When you create a model serving endpoint on Databricks, you have the option to enable AI Gateway features, including the following:

Rate limiting
: Limit the number of queries per user and per endpoint.

Payload logging
: Log the inputs to and outputs from a model as Delta tables, for monitoring and auditing.

Usage tracking
: Capture the operational usage on endpoints and associated costs as part of the system tables, for monitoring.

AI guardrails
: These automatically detect and block unsafe or harmful content, such as references to violent crime, self-harm, or hate speech. You can also choose to detect, block, or mask PII content. The guardrails can be enabled independently for both model inputs and outputs.

Fallbacks
: To minimize production outages during and after deployment, you can choose to send your request to other served models on the endpoint as a fallback.

Traffic splitting
: You have the option to load balance traffic across models, common for A/B testing models and other validation methods.

These features are available out of the box and address most of the challenges that organizations face while dealing with production model serving. In addition, they are a common component applicable to all kinds of models, such as Azure OpenAI, Amazon Bedrock on AWS, Meta Llama 3, etc., that you choose to use on or through Databricks.

Components of an AI System

Until now, we've looked at various aspects of AI governance and talked about different components in turn. Let's now zoom out for a bird's-eye view of an AI system on Databricks, as illustrated in Figure 6-5.

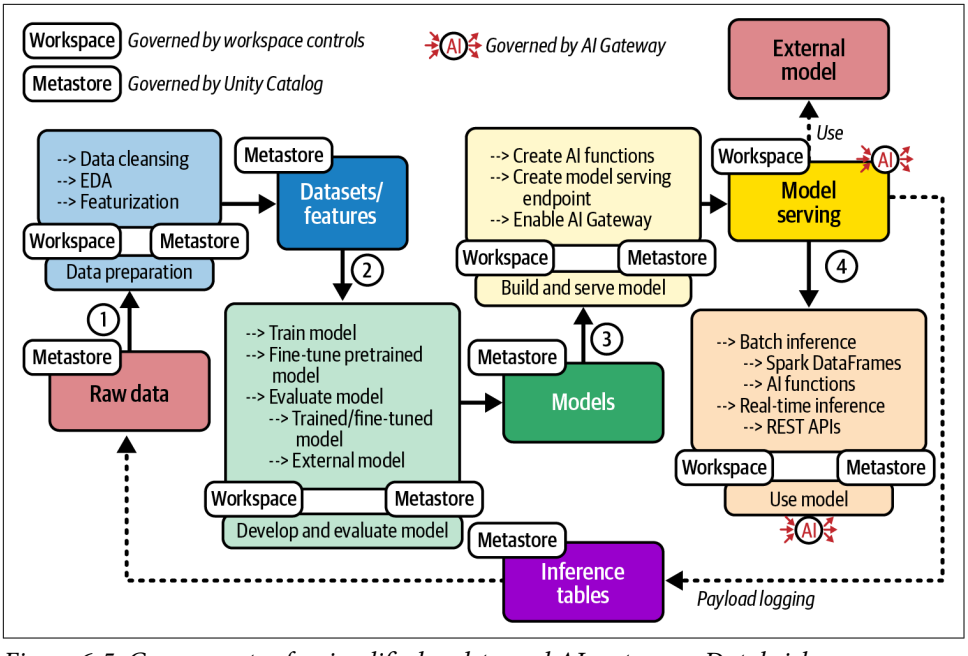


Figure 6-5. Components of a simplified end-to-end AI system on Databricks

The four phases in the simplified end-to-end lifecycle of an AI system on Databricks are as follows:

1. You start with raw data, which can be cataloged as volumes with Unity Catalog. You clean the data, indulge in exploratory data analysis (EDA), and create features for training and testing a model.
2. You run various experiments to train a model, evaluate it, and choose the one that performs best. If you are fine-tuning an existing model, the steps are more or less the same. You can also choose to use an external model, in which case, you skip the training part and focus just on the evaluation part and find the one that best fits your use case. You can use various public benchmarks during the selection process.
3. Once you have the model ready, you go ahead and use it, either for batch inference or for real-time inference.
4. Once the model is served, you can integrate your AI applications to use the model-serving endpoint to fetch predictions. For offline batch inference, you either use Spark DataFrames or create AI functions that you could invoke from SQL queries.

If you want, you could log the payload as inference tables, which you could then use for analysis and model improvement.

In terms of governance, Unity Catalog covers all the assets, namely, raw data in volumes, datasets, and features in the form of tables, trained or fine-tuned models, and functions. Model serving is governed by workspace controls and the AI Gateway.

Treating an AI system as opaque wouldn't be much help; instead, the best approach to securing any AI system is to divide it into key components, identify risks associated with each component, and devise a strategy to mitigate them. This is also a good way to evaluate a platform and understand whether you will be able to leverage its features to easily and effectively handle the risks. For a detailed analysis of risks and corresponding mitigation strategies associated with an AI system on Databricks, refer to the Databricks AI Security Framework (DASF).

Implementing an AI System

Like many other companies, Nexa Boutique wanted to jump on the AI bandwagon and develop something appealing. They already had several teams of data scientists and ML engineers implementing traditional ML techniques for various use cases. But for the first time, the top-level executives were interested in what AI can do and how they could benefit from it. The CTO, CDO, and even the CEO were aligned on this, and there was some budget made available for AI initiatives. The teams were also equally enthusiastic to capitalize on this opportunity. Several teams at Nexa, being in the retail industry, envisioned multiple use cases that could benefit from the latest advancements in AI. But clearly there was a lack of understanding and experience on how to develop end-to-end AI systems that are secure, reliable, and maintainable. So, they decided to focus on just one use case and, if successful, templatize the approach and use it for other use cases.

Since there was a clear need for an AI-powered chatbot that could help their end customers on their websites and apps, they decided to go with that use case. The decision was also influenced by the tech teams, who knew many other organizations had built or were building similar solutions, and various resources were available for reference. Yet another decision was to leverage an LLM model that they could simply use without customizing it, at least in the first iteration.

In terms of the tech stack, since Databricks had a large footprint within the organization and most of the data scientists and engineers were already familiar with the platform, Nexa started exploring its capabilities. It did not take them long to realize that the Databricks Platform had almost all the things that they needed to develop and run their AI system. Yet another key reason to go with Databricks was the fact that they already had most of the data available in Databricks; hence, building AI systems on Databricks would be much more convenient, as they did not need to copy over the data to another platform. Moreover, although not perfect, Databricks offered an end-to-end security and governance model that no other platform did.

In terms of the GenAI solution, Nexa chose to go with Azure OpenAI. One reason was undoubtedly the popularity of OpenAI, and the other was their existing relationship with Microsoft, which meant they did not have to go through the long process of security approvals that would be involved for any new vendor.

In terms of technique, they chose *retrieval-augmented generation (RAG)*. RAG is a technique that enhances LLMs by combining their generative abilities with real-time information retrieval to generate more accurate, up-to-date, and contextually relevant responses. This approach is best suited for answering questions about proprietary, frequently changing, or domain-specific information—hence, quite appropriate for Nexa for their chatbot use case. For faster information retrieval, RAG relies on the *Vector Search*, a technique that uses *vectors,* numerical representations of data, such as text, images, or audio, and a bit of algorithmic mathematics to find items that are similar in meaning or context, rather than relying on keyword matches.

On Databricks, you can use MLflow with LangChain, a software framework for integrating LLMs into applications. Mosaic AI Vector Search is a Databricks service that you can use to build vector search indices for faster retrieval of *embeddings*, i.e., encoded vectors. Nexa used these options to build the RAG chain on Databricks. High-level implementation steps were as follows:

1. Choose to use an existing Databricks workspace to build the AI system for the chatbot application.
2. Set up Databricks model-serving endpoints for external embedding and chat models using the AI solution offered by the cloud provider.
3. Create a vector search endpoint.
4. Set up a vector search index using the vector search endpoint and the model-serving endpoint that uses the external embedded model.
5. Create and register the RAG chain model in Unity Catalog using MLflow and LangChain.
6. Create a RAG chain model-serving endpoint to orchestrate the flow.
7. Create a Mosaic AI agent evaluation review app for evaluating the system.
8. Use MLflow tracing for observability of the RAG chain tasks.
9. Once evaluated and approved, integrate the RAG chain model endpoint with the chatbot application using a service principal with an OAuth token.
10. The governance aspects are inherently managed by Unity Catalog, workspace controls, and the Mosaic AI Gateway.

Figure 6-6 illustrates this setup. The detailed specifications of all the steps are not relevant to this book.

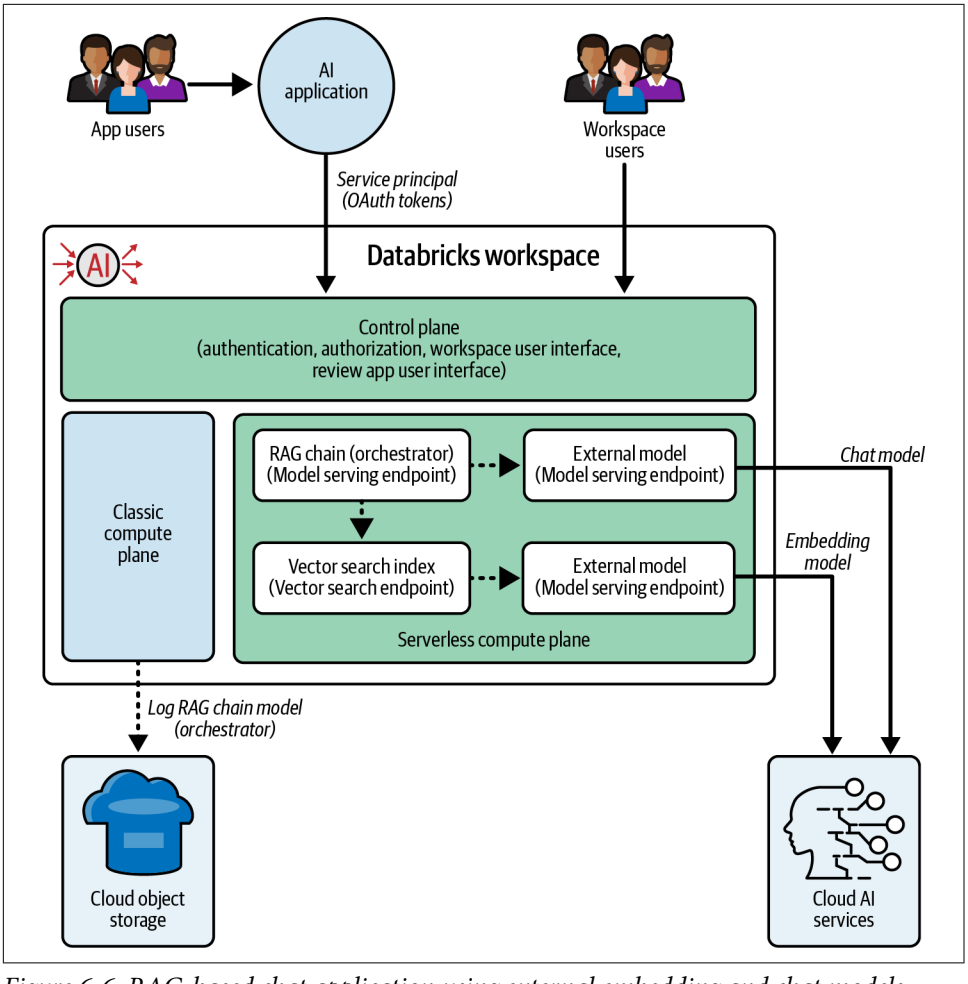


Figure 6-6. RAG-based chat application using external embedding and chat models

The chatbot use case Minimum Viable Product (MVP) was implemented within a few weeks and gained traction across the organization. It was a great success. A small team without much knowledge or experience in GenAI topics managed to pull off one of the high-stakes use cases, which had the attention of C-level, with minimal effort and cost. Of course, it took some time to productionize the system because they had to implement major changes in the chatbot application, fallback scenarios, incident handling, compliance checks, and so on. This use case setup was templatized, and all the following use cases followed the same pattern. The only major variable was the LLMs themselves. Some teams preferred to go with Databricks' foundational models, and some preferred externally hosted closed source models, which basically illustrates the plug-and-play model of their setup.

Summary

In this chapter, we examined the major factors that necessitate strong security and governance measures while implementing AI use cases. We focused on the key aspects of AI governance and how to achieve them on the Databricks Platform. We also examined what an end-to-end AI system on Databricks looks like and how Nexa Boutique implemented a RAG chain-based chatbot application using Databricks. In Chapter 7, we will delve into the discoverability of data and AI assets on Databricks and outline available options for implementing observability.

CHAPTER 7
Observability and Discoverability

People spend 60% to 80% of their time trying to find data. It's a huge productivity loss.
—Dan Vesset, Group Vice President, IDC

Before joining Databricks, Kiran worked on an on-premises data platform running Apache Spark for almost four years. One of the key issues he faced while running the platform, supporting multiple users who did regulatory reporting on the data, was data quality. Working on near real-time data processing use cases, particularly those that rely on message processing systems such as Apache Kafka, Azure Event Hubs, or AWS Kinesis, further aggravated the problem. The risk of missing a message in transit from the message processing queue without anyone knowing its absence was exceptionally high. Almost 95% of the time, the issues were attributed to low data discoverability in the data platform. The support team developed, deployed, and monitored custom tools to prevent data loss issues. The team used multiple monitoring tools for different platform aspects, which complicated the overall process. Building and deploying these tools helped curb data issues but came with the overhead of managing the infrastructure and additional costs.

When building a business application using data, it's essential to prioritize data quality from the outset, rather than treat it as an afterthought. Data quality is just one part of the equation. Once you have solved it, the next critical issue concerns data discoverability and platform observability, which answers some key questions about your data:

- What data assets do you have in the platform?
- How do you find who has access to the data assets?
- What is the status of your data ingestion pipeline?

- How much money is your team spending on the data platform?
- Is there a way to optimize your workload for cost and efficiency?

The number of questions on the data and AI platform is endless. How does Databricks solve this, and in particular, what does Unity Catalog provide you for your data discoverability and observability? Let's find out how exactly Unity Catalog helps you.

System tables are Databricks-hosted Delta tables that capture the data platform's operational data, including auditing, billing, and workflows, which appears as a catalog in the metastore. We look at the different aspects of using system tables to build your observability of the platform.

The platform offers different capabilities to build data quality monitoring for your data assets. We discuss the various ways of assessing data quality using the Lakehouse Monitoring tool.

The platform's data discoverability is enabled by AI-based metadata generation, semantic tags, the `BROWSE` privilege, and a context-aware search. We discuss how you can make your data assets discoverable to end users, enabling them to deliver value from the data faster.

Unity Catalog System Tables

According to Barr Moses, CEO and cofounder of Monte Carlo, a company specializing in observability, the five pillars of data observability are as follows:

1. Freshness: It is all about the freshness of your data and ensuring it's up to date.
2. Quality: Is your data within the accepted range of inaccuracy?
3. Volume: This refers to data completeness and health.
4. Schema: Is your data organized in the right way?
5. Lineage: This gives details on the upstream and downstream data flow.

Without a centralized observability solution, responding to issues becomes reactive rather than proactive. However, building an observability solution for a data platform is complex. When the data platform comprises multiple tools for data ingestion, ETL, governance, ML, and AI workloads, the observability becomes more complicated, as illustrated in Figure 7-1. Organizations usually use different solutions for different platform layers, often leading to fragmented observability, resulting in productivity and revenue loss.

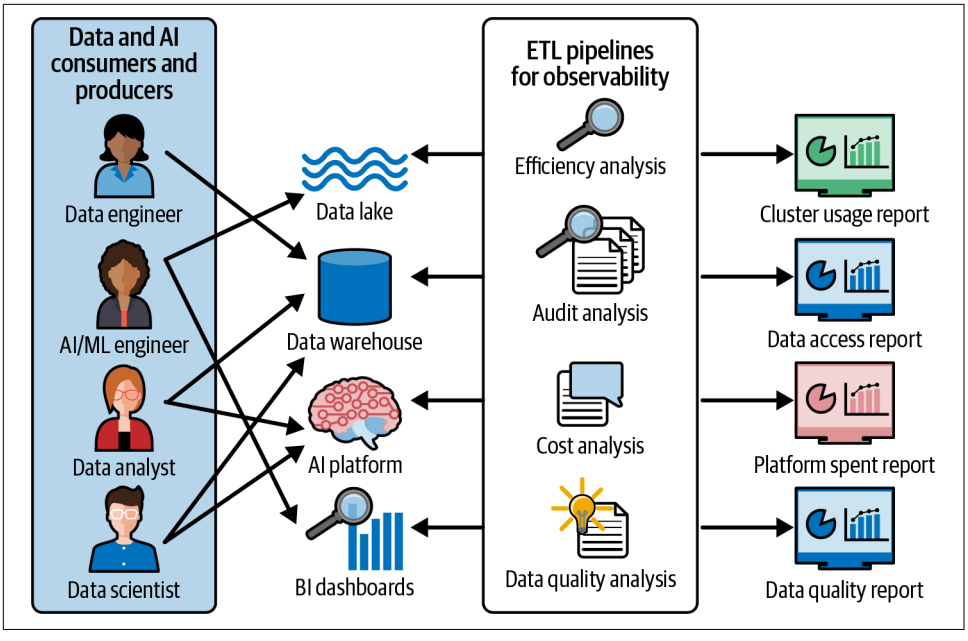


Figure 7-1. Data and AI observability is complex due to multiple tools in the platform

The three foundational elements of observability—logs, metrics, and traces—offer a comprehensive view into the platform's performance and the behavior of the services that operate within it. System tables in Unity Catalog enable observability by providing a centralized snapshot of the functioning and use of the Databricks platform. It powers one of the pillars of Unity Catalog, AI-powered monitoring and observability. System tables are a Databricks-hosted analytical store for all operational data, enabling the platform to provide historical observability. This data includes the following:

Cost and usage analytics
: Understanding the $DBU(Databricks Unit used to denote processing power for measurement and pricing purposes) spent and insights on using different product SKUs on the Databricks Platform. All cost-related metrics come in the Unity Catalog schema `system.billing`.

Efficiency analytics
: Understand the compute usage and optimize it for efficiency and cost. All compute-related metrics come in the Unity Catalog schema `system.compute`.

Audit analytics
: Find who accessed the data and prevent security breaches. All audit-related metrics come in the Unity Catalog schema `system.access`.

Service level agreement (SLA)/service level objectives (SLO) analytics
: Insights on key metrics for workflow runs to meet key SLO/SLA. All workflow-related metrics come in the Unity Catalog schema `system.lakeflow`.

At Nexa, the CDP team built an observability tool for their Databricks platform before Unity Catalog using Grafana and Prometheus, as shown in Figure 7-2. The setup reads audit logs and Apache Spark cluster logs to provide insights on data access and cluster performance. Even though this worked well for the CDP team, it required a dedicated team to host, set up, and support the monitoring solution.

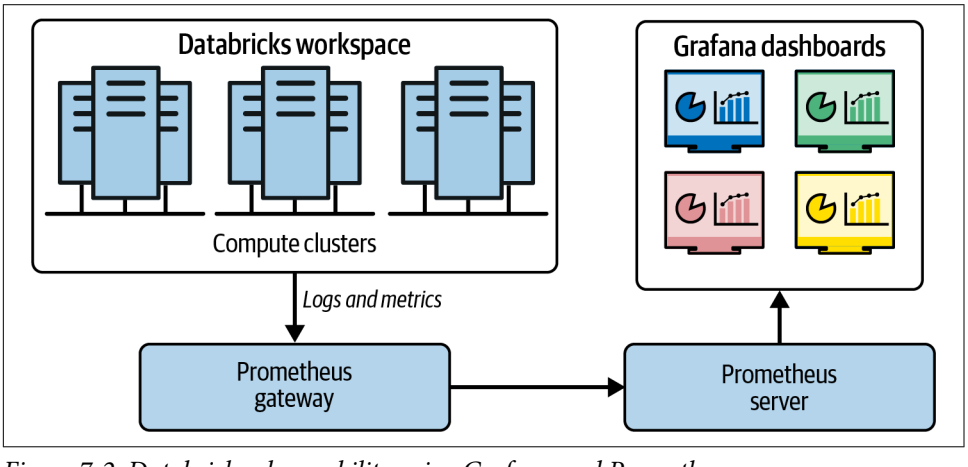


Figure 7-2. Databricks observability using Grafana and Prometheus

Definition: Prometheus

Prometheus is a freely available, open source solution for monitoring and alerting, first created at SoundCloud in 2012. Over time, it has evolved into a highly popular and self-sustaining initiative backed by a vibrant community of contributors and adopters. In 2016, Prometheus became a part of the Cloud Native Computing Foundation.

Definition: Grafana

Grafana, an open source platform, empowers you to unlock insights from your metrics, logs, and traces, no matter where they reside. The platform's flexible plug-in ecosystem allows you to seamlessly connect to a broad range of data sources, including relational and NoSQL databases, IT service management tools like Jira and ServiceNow, and CI/CD platforms like GitLab.

Imagine having all these features at your fingertips, including prebuilt dashboards that track your Databricks usage and spending and user-level audit logs out of the box. The Unity Catalog governance solution combines audit and operational logs with system tables. System tables are read-only Delta tables that an admin user can access from any Databricks workspace enabled with Unity Catalog. By carefully designing an access model, these tables can also be shared with users for analysis. You can build customized reports on the data captured in system tables using the Databricks AI/BI dashboards or any BI tool of your choice.

Definition: Databricks AI/BI Dashboard

AI/BI is a BI product that understands your data's semantics and lets you build interactive dashboards using a compound AI system. It provides a low-code authoring experience for visualizations and supports natural language-based interactions when creating reports.

Architecture

System tables work across workspaces in a Databricks account, enabling you to monitor your Databricks account-level, cross-workspace operational metrics. Databricks hosts system tables, and all the data is stored in a Databricks-managed storage account in the same region as your Unity Catalog metastore. The data itself is encrypted and inaccessible to end users.

Once the metrics are captured and stored in the storage layer as a Delta table, the system tables are shared with your Unity Catalog metastore via Delta Sharing. Sharing the tables via the Delta Sharing protocol enables zero-data copy share across Databricks workspaces. These tables become accessible from any Databricks workspace you have that is Unity Catalog enabled.

Figure 7-3 illustrates how the metrics are captured from the Databricks workspaces and made available as Delta tables in the system catalog in Unity Catalog via Delta Sharing.

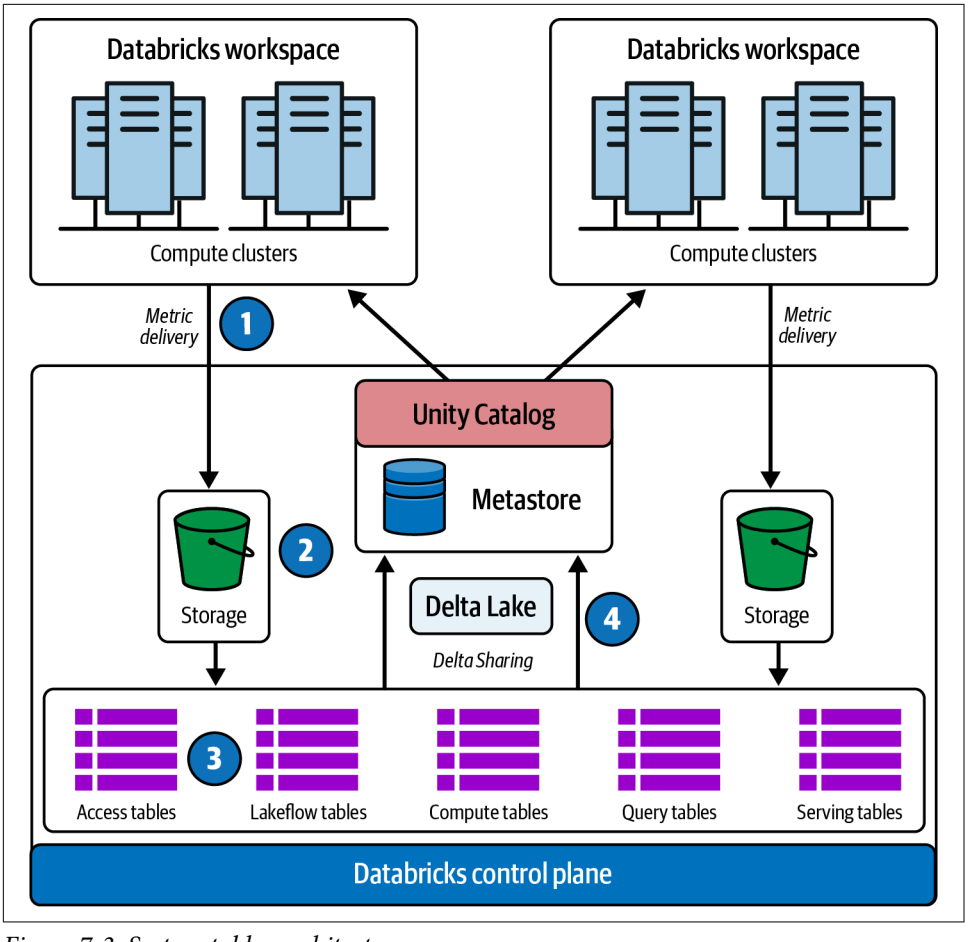


Figure 7-3. System tables architecture

Here is how it works:

1. Automated processes from each workspace under the Databricks account capture the key metrics related to a Databricks workspace. Since a metastore is regional, all operational metrics for workspaces attached to the same metastore will be available centrally in the metastore.
2. The data is ingested and stored in the Databricks-managed storage in the control plane.
3. Transformed and modeled data is added as Delta tables in the control plane managed by Databricks.
4. The tables are shared with the metastore of the workspaces using the Delta Sharing protocol.

Audit Observability

The *access* schema in system tables holds audit and lineage information across your workspaces, providing a centralized repository for critical access logs. Specifically, the `system.access.audit` table captures detailed audit logs, including user identity, source IP address, and timestamp, which are essential for maintaining accountability and traceability from a governance and security perspective. This information is crucial for security teams investigating data breaches, as it helps them understand the audit trail for data access. In fact, many organizations rely on these audit logs to comply with regulatory requirements. Previously, analyzing these logs required custom tools or specialized log analytics products, as most of the logs were written to a storage layer that required further processing to be usable.

In contrast, Unity Catalog-enabled system tables automate the loading of this information, making it easily accessible for a user who is a metastore admin as well as an account admin. As in the following example, you can use a SQL query against the `audit` system table to analyze which users have accessed a table within the last seven days. Table 7-1 shows a sample output from the query against the audit table. Organizations can streamline their audit log analysis by leveraging system tables and improving their security posture:

```
SELECT
 user_identity.email as `User`,
 IFNULL(
   request_params.full_name_arg, request_params.name
 ) AS `Table`,
 action_name AS `Type of Access`,
 event_time AS `Time of Access`
FROM
 system.access.audit
WHERE
 (

  request_params.full_name_arg = : table_full_name


   OR (
     request_params.name = : table_name
     AND request_params.schema_name = : schema_name
   )
 )
 AND action_name IN (
   'createTable', 'getTable', 'deleteTable'
 )


 AND event_date > now() - interval 7 day
ORDER BY
 event_date DESC
```

Table 7-1. Audit table showing which users accessed a table within the last seven days

User	Table	Type of access	Time of access
kiran.sreekumar	c360.bronze.customer	getTable	2025-06-18T04:xx

Nexa's networking team blocks internet access from their Databricks workspaces to prevent unauthorized data downloads and potential data breaches. When a user attempts to access the internet from a workspace, the system blocks the request and logs the attempt as an audit event. The `access.outbound_network` system table captures this information, recording the DNS event, the user-accessed domain name, and the event's timestamp, thereby enabling thorough audits of internet access attempts.

Definition: DNS

Domain Name System is the global network that translates human readable domain names into IP addresses. This is the backbone of the internet that enables communication between devices.

Databricks provides an AI-powered assistant to help you answer your platform queries, fix code issues, and generate and optimize code. The context-aware assistant can help you accelerate the work you are implementing in the Databricks Platform. Tracking and auditing the assistant's usage is essential, and the `assistant_events` system table does the same.

An export of a Databricks AI/BI dashboard for assistant usage is available on GitHub. You could import it to your Databricks workspace to instantly get a dashboard that can provide the following:

- Active users by day and month
- Active users per workspace
- Top users overall
- Submission data: per workspace and total

All of those are built from the `assistant_events` system tables.

Databricks Clean Rooms enables multiple parties to share sensitive enterprise data on a secure platform without directly accessing each other's data. As will be discussed in more detail in Chapter 8, this feature is designed to facilitate collaboration and data sharing while maintaining the highest levels of security and control. Auditing is vital since it involves sharing sensitive and business-critical information with trusted collaborators. The `clean_room_events` system table provides a comprehensive record of all events related to your Databricks Clean Room, including event time, event type, and other activities. By leveraging this table, you can gain visibility into all Clean Room activities, ensuring you can track and audit all interactions with your sensitive data. This enables you to maintain your data's integrity and security while complying with regulatory requirements and internal governance policies.

Lineage Observability

The *access* schema contains table and column lineage information in the `system.access.table_lineage` and `system.access.column_lineage` system tables. Databricks Unity Catalog enables one of its most critical features: capturing run-time data lineage for queries executed on Unity Catalog data. The lineage data tracks the impact of queries on upstream and downstream notebooks, workflows, dashboards, and tables. While the workspace visualizes this lineage data for easier analysis, the lineage tables also store the raw data, making it readily accessible for users to build custom use cases and reports. As a key element of Unity Catalog, data lineage is automatically visualized in the workspace user interface for every workload processed through Unity Catalog, as illustrated in Figure 7-4.

You can write a Python function to find the lineage from the lineage system table for a Unity Catalog external table:

```
def getLineageForTable(table_name):

   df = spark.read.table("system.access.table_lineage")
   return df.where(
       (df.source_table_full_name == table_name)
       | (df.target_table_full_name == table_name)
       | (df.source_path == table_path)
       | (df.target_path == table_path)
   )
display(getLineageForTable("table_name"))
```

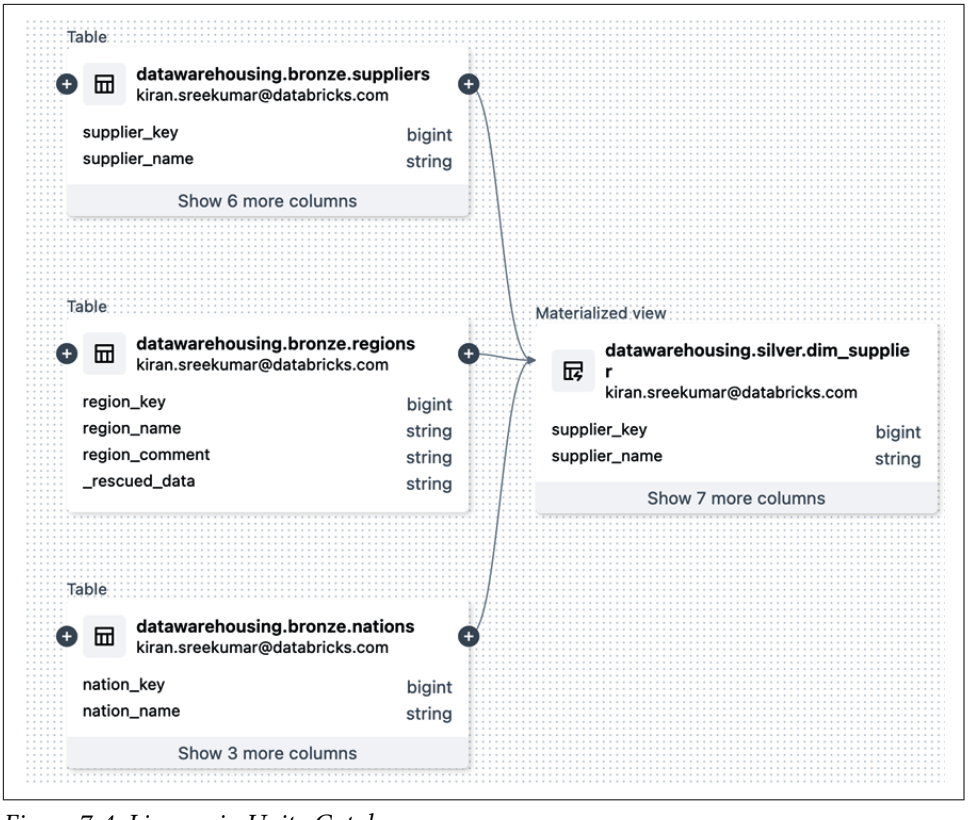


Figure 7-4. Lineage in Unity Catalog

Cost Observability

Nexa tracks and reports its IT spending budget to the finance and executive leadership teams. With Databricks, the company can track its spending using the `system.billing` system table. The `billing` schema consists of two tables—`usage` and `list_prices`—and captures centralized information across all regions where Databricks workspaces are deployed, providing a global view of the account's usage. The usage table records detailed information, including usage quantity, date, product SKU, start and end time, and other relevant data, giving you a comprehensive overview of your actual spending on the platform.

The `list_price` table stores historical SKU pricing, which, when combined with actual consumption data from the *usage* table, enables you to calculate the exact cost of using the Databricks Platform. By updating information almost every hour, the platform provides near real-time spending insights, also visualized in a built-in AI/BI dashboard available in the account console under usage. With up-to-date cost information, you can effectively budget for new solutions and growth in existing ones, ensuring control over your spending. Using SQL queries, you can answer specific questions, such as this one:

```
"What is the spending difference between Databricks PREMIUM_ALL_PURPOSE_COMPUTE
 SKU and PREMIUM_ALL_PURPOSE_COMPUTE_(PHOTON) over the last 3 days?"
```

The results can then be plotted on a bar chart in the Databricks AI/BI dashboard, as Figure 7-5 illustrates, from a query like this:

```
select
 usage_date as `Usage Date`,
 usage.sku_name,
 sum(usage_quantity) as dbus,
 pricing.default as list_price,
 sum(usage_quantity * pricing.default) as `Dollar DBU at list Price`
from
 system.billing.usage
 left join system.billing.list_prices on list_prices.sku_name = usage.sku_name
 and list_prices.cloud = usage.cloud
 and list_prices.price_start_time <= usage.usage_start_time
 and (
   list_prices.price_end_time > usage.usage_start_time
   OR list_prices.price_end_time IS NULL
 )
where
 usage_date between now() - interval 3 day
 and current_date
 and (
   usage.sku_name = 'PREMIUM_ALL_PURPOSE_COMPUTE'
   or usage.sku_name = 'PREMIUM_ALL_PURPOSE_COMPUTE_(PHOTON)'
 )
group by
 usage_date,
 usage.sku_name,
 list_price
order by
 Sku_name
```

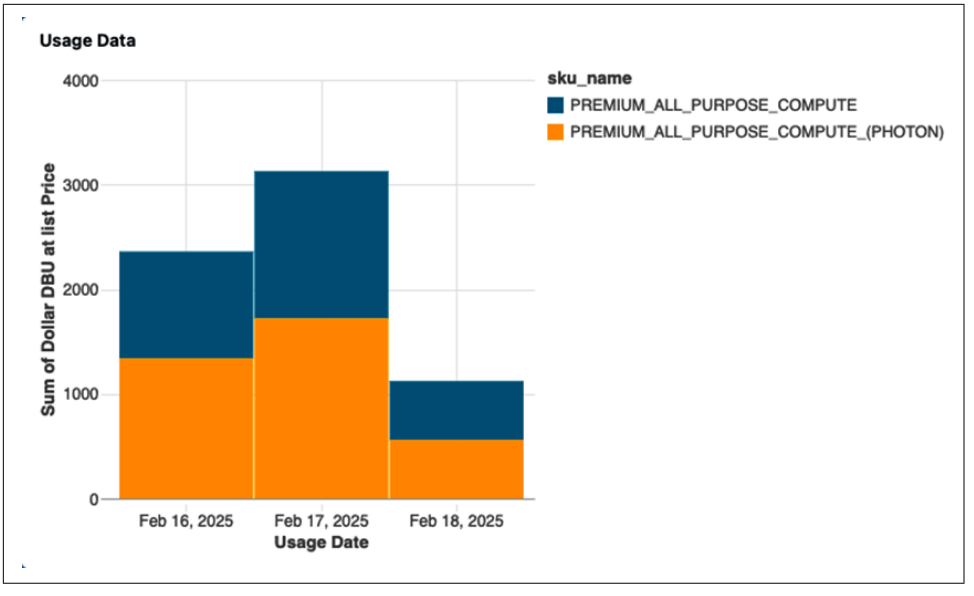


Figure 7-5. Usage of SKU plotted in a bar chart

Compute Observability

The `system.compute` schema provides a comprehensive view of your compute usage in your Databricks account, covering all-purpose compute, jobs compute, and SQL warehouse instances. The clusters and `node_type` tables offer detailed hardware-related information about your compute instances, including instance type, number of workers, DBR version, number of cores, and memory.

For a deeper dive into cluster performance, the `node_timeline` table is a valuable resource. This table captures key CPU, memory, and disk usage metrics, giving insight into your cluster's performance over time. By analyzing these metrics, you can identify the underutilization of compute resources and optimize your infrastructure by downsizing to a lower-tier VM, leading to significant cost savings over time. For instance, if your workloads consistently use only 20% of the available capacity, you can consider migrating to a smaller VM instance, reducing your costs while still meeting your performance requirements.

If you use SQL data warehouses for BI reporting, the `warehouses` and `warehouse_events` tables are essential resources. The `warehouse_events` table, in particular, can help you answer important questions about your SQL warehouse usage, such as this:

```
"Which warehouse had the most uptime hours over the last 30 days?"
```

Table 7-2 shows a sample output from the query against the warehouse events table:

```
SELECT
 warehouse_id,
 SUM(
   TIMESTAMPDIFF(MINUTE, start_time, end_time)
 ) / 60.0 AS uptime_hours
FROM
 (
   SELECT
     starting.warehouse_id,
     starting.event_time AS start_time,
     (
       SELECT
         MIN(stopping.event_time)
       FROM
         system.compute.warehouse_events AS stopping
       WHERE
         stopping.warehouse_id = starting.warehouse_id
         AND stopping.event_type = 'STOPPED'
         AND stopping.event_time > starting.event_time
     ) AS end_time
   FROM
     system.compute.warehouse_events AS starting
   WHERE
     starting.event_type = 'STARTING'
     AND starting.event_time >= DATEADD(
       DAY,
       -30,
       CURRENT_DATE()
     )
 ) AS warehouse_uptime
WHERE
 end_time IS NOT NULL
GROUP BY
 warehouse_id
ORDER BY
 uptime_hours DESC
```

Table 7-2. `warehouse_events` table showing warehouses with the most uptime hours over the last 30 days

warehouse_id	uptime_hours
8ca9df56d19022e8	945.266667

By leveraging the `system.compute` schema, you can better understand your compute usage and optimize your Databricks account for improved performance and efficiency.

The availability of compute utilization metrics in system tables enables you to build alerts that help prevent overspending and stay within budget. For example, at Nexa, the CDP team previously relied on custom tools to read usage logs and generate alerts on outliers. However, with Databricks SQL alerts and system tables, they can now create automated email alerts based on compute metrics, eliminating the need for custom solutions. The CDP team can build better FinOps practices and ensure more effective cost control by leveraging system tables. In summary, using system tables for alerting and cost control provides a more efficient, automated, and scalable solution, enabling organizations to optimize their compute utilization and stay within budget.

Definition: FinOps

FinOps, a term derived from Finance and DevOps, refers to a cultural practice that brings together teams from across the enterprise to achieve better control and predictability of cloud spending. By adopting FinOps, organizations can gain greater visibility into their cloud costs, optimize resource utilization, and make data-driven decisions to drive business growth.

Jobs Observability

The Databricks workflow UI provides a comprehensive view of job runs within the platform for a defined retention period. However, this information is limited to each workspace level. To access account-level information across workspaces, you must exert additional effort. This is where the `system.lakeflow` system tables come in, providing all workflow-related information as a Delta table across your Databricks workspaces. The `jobs` table, for instance, includes a list of all the workflows defined across your workspaces, including the name of the job, the creator, and the workspace ID where the job is deployed. The tasks defined within your workflows and their dependencies are captured in the `job_tasks` table.

The job and task execution details are captured separately in the `job_run_timeline` and `job_task_run_timeline` tables. These tables capture key metrics about your job runs, including the start and end times, the status of the run, and any termination code for further analysis if required. By leveraging the `system.lakeflow` system tables, you can better understand your workloads in Databricks and pinpoint issues with your workloads. For example, you could write a SQL query against the `lakeflow` system tables combined with your `billing` system tables to identify the most expensive jobs over the last 30 days:

```
with list_cost_per_job as (
 SELECT
   t1.workspace_id,
   t1.usage_metadata.job_id,
   COUNT(
```

```
    DISTINCT t1.usage_metadata.job_run_id
  ) as runs,
  SUM(
    t1.usage_quantity * list_prices.pricing.default
  ) as list_cost,
  first(identity_metadata.run_as, true) as run_as,
  first(t1.custom_tags, true) as custom_tags,
  MAX(t1.usage_end_time) as last_seen_date
 FROM
  system.billing.usage t1
  INNER JOIN system.billing.list_prices list_prices on
  t1.cloud = list_prices.cloud
  and t1.sku_name = list_prices.sku_name
  and t1.usage_start_time >= list_prices.price_start_time
  and (
    t1.usage_end_time <= list_prices.price_end_time
    or list_prices.price_end_time is null
  )
 WHERE
  t1.billing_origin_product = "JOBS"
  AND t1.usage_date >= CURRENT_DATE() - INTERVAL 30 DAY
 GROUP BY
  ALL
),
most_recent_jobs as (
 SELECT
  *,
  ROW_NUMBER() OVER(
    PARTITION BY workspace_id,
    job_id
    ORDER BY
      change_time DESC
  ) as rn
 FROM
  system.lakeflow.jobs QUALIFY rn = 1
)
SELECT
 t2.name,
 t1.job_id,
 t1.workspace_id,
 t1.runs,
 t1.run_as,
 SUM(list_cost) as list_cost,
 t1.last_seen_date
FROM
 list_cost_per_job t1
 LEFT JOIN most_recent_jobs t2 USING (workspace_id, job_id)
GROUP BY
 ALL
ORDER BY
 list_cost DESC
```

Table 7-3 shows a sample output from the query against lakeflow and billing tables.

Table 7-3. Lakeflow and billing table showing the most expensive jobs over the last 30 days

name	job_id	workspace_id	runs	run_as	list_cost	last_seen_date
data_etl	111	222	31	user	3.6	2025-06-18T08:xx

In summary, the `system.lakeflow` system tables provide a powerful tool for analyzing and optimizing your Databricks workloads, enabling you to make data-driven decisions and improve your overall workflow efficiency.

A Databricks AI/BI dashboard export for job monitoring is available on GitHub. You could import it to your Databricks workspace to instantly get a dashboard that can provide the following:

- Job performance tracking
- Failure monitoring
- Resource utilization

All of which are built from the `lakeflow` system tables.

Marketplace Observability

Databricks marketplace is an open forum to exchange your data products, including tables, AI/ML models, notebooks, and files. As a data provider, you can monetize your valuable data by listing it on the marketplace, reaching a broad audience of potential consumers. The `marketplace` system table provides a range of features to help you manage and analyze your listings. The `listing_access_events` table, for example, records all consumer information for any data requests for your marketplace listing, including the consumer's email ID, name, and company details. This information can help track interest in your listings and identify potential customers.

In addition to access events, the `listing_funnel_events` table records the impressions and actions a consumer takes on your listing. The event type is critical information for analytics, as it can help you understand how consumers interact with your listings. The event type can have several values essential for understanding the consumer's journey and optimizing your listings for better performance. By analyzing these event types, you can gain valuable insights into consumer behavior and make data-driven decisions to improve your listings and increase sales. Again, using simple SQL queries, you can get insights like top listing by number of requests:

```
SELECT
 listing_name,
 consumer_cloud,
 Count(*) AS requestCount
```

```
FROM
 system.marketplace.listing_access_events
GROUP BY
 listing_name,
 consumer_cloud
ORDER BY
 requestcount DESC
```

Table 7-4 shows a sample output from the query against the listing access table.

Table 7-4. `listing_access_events` table showing the top listing by number of requests

listing_name	consumer_cloud	requestCount
Stock Market Data	aws	22

Model Serving Observability

The GenAI space is rapidly evolving, with more robust and cost-effective models being launched frequently. As these models become increasingly intelligent and sophisticated, the need for governance and monitoring has never been more pressing. In an enterprise setup, it is imperative to ensure that these models are used safely and responsibly, with proper guardrails in place to prevent misuse.

To address these challenges, Databricks provides the Mosaic AI Gateway, a powerful tool for generative GenAI model monitoring discussed in detail in Chapter 6.

The Mosaic AI Gateway's usage tracking functionality, enabled via system tables in Unity Catalog, provides a powerful tool for monitoring and analyzing GenAI model usage. The `system.serving` schema contains two key tables: `served_entities` and `endpoint_usage`. The `served_entities` table records details about hosted models, including endpoint names and tasks like embeddings and chat. The `endpoint_usage` table, on the other hand, records usage information for hosted endpoints, including the requester, time of request, input and output token count, and more.

By leveraging these tables, enterprises can gain insights into how their GenAI models are being used, including which models are most popular, how they are being used, and what kind of requests are being made. For example, a company might use the `endpoint_usage` table to identify which of its chatbot models is being used most frequently and then optimize its performance to improve UX.

Enterprises can use the Mosaic AI Gateway and system tables to ensure their GenAI models are used safely, responsibly, and effectively while driving business value and innovation. For instance, Nexa uses the usage tracking functionality to identify areas where its models are underutilized and then develop targeted marketing campaigns to increase adoption. By providing detailed monitoring and analysis of GenAI model

usage, the Mosaic AI Gateway and system tables can help enterprises unlock the full potential of their AI investments and drive real business results.

A Databricks AI/BI dashboard export for Mosaic AI Gateway is available on GitHub. You could import it to your Databricks workspace to instantly get a dashboard that can provide the following:

- Aggregated daily insights per endpoint
- Total tokens per endpoint by week
- All usage tracking requests for all endpoints

All of those are built from the `serving` system tables.

Query History and Storage Observability

The `query.history` system table in Unity Catalog provides a comprehensive record of all queries executed against either your SQL warehouse or your serverless generic compute for notebooks and jobs. This table offers a centralized location to view all query history information across your workspaces in the same region, making tracking and analyzing query performance easier. The `query.history` table captures a wide range of metrics, including who executed the query, the statement executed, the number of files read, the total duration, the data spilled to local storage size, and execution status. This information can help optimize query performance, identify bottlenecks, and troubleshoot issues. For example, Nexa uses the `query.history` table to determine which queries are taking the longest to execute and which are suffering from increasing I/O times. By analyzing this data, Nexa can identify optimization opportunities and optimize those queries to improve performance and reduce latency.

It's worth noting that the query history data can grow to huge volumes based on the number of active users in the Databricks Platform and the average queries executed per day. To manage this data, the `query.history` table retains data for only 90 days. However, if you want to keep the data longer, you can materialize the history table data into a separate Delta table based on a schedule. This allows you to store the entire history of query execution for later analysis, which can help track trends and patterns in query performance over time. By leveraging the `query.history` system table, you can gain insights into query performance and optimize your Databricks workflow for better efficiency and productivity.

Running maintenance operations like `OPTIMIZE`, `VACUUM`, and `ANALYZE TABLE` on your Delta tables is essential to ensure optimal performance.

The book *Delta Lake: The Definitive Guide* (O'Reilly) by Denny Lee, Tristen Wentling, Scott Haines, and Prashanth Babu talks extensively about the Delta filesystem and its maintenance operations.

Predictive optimization, however, automates the process of running the maintenance jobs for your Unity Catalog managed tables and ensures the best performance for your queries. System tables also offer a `storage.predictive_optimization_operations_history` table that captures the metrics related to predictive optimization jobs that run in your Databricks workspaces. The table can answer queries like "`What tables had the most bytes vacuumed?`" Table 7-5 shows a sample output from the query against the predictive optimization table.

```
SELECT
 metastore_name,
 catalog_name,
 schema_name,
 table_name,
 SUM(
   operation_metrics[ "amount_of_data_deleted_bytes" ]
 ) as bytesVacuumed
FROM
 system.storage.predictive_optimization_operations_history
WHERE
 operation_type = "VACUUM"
GROUP BY
 ALL
ORDER BY
 bytesVacuumed DESC
```

Table 7-5. `predictive_optimization_operations_history` table showing the top listing by number of requests

metastore_name	catalog_name	schema_name	table_name	bytesVacuumed
customer-east	prod	c360	customer	194507275615

Observability Assistant

AI/BI Genie is a GenAI assistant that understands your organization's data and enables users to interact using natural language. To perform optimally, you can guide and teach the Genie space with general instructions, sample SQL statements, and your unique enterprise jargon, logic, key performance indicators (KPI), etc.

One key knowledge base for Genie is the table metadata from Unity Catalog. You could curate a Genie space in your Databricks workspace specifically for observability using system tables in Unity Catalog. Having a Genie space on system tables gives you

quick insights into your platform, just by interacting with it using natural language. Access controls on the Genie space data will default to the admin user who has read access to the system tables.

You can add general instructions for the Genie space, such as these:

- When asked about the job owner, it is the same as the creator ID in the job table.
- A job cluster has `'job'` in the name.
- An all-purpose cluster doesn't have `'job'` in the name.

You can add sample SQL queries to train the Genie space, such as this one:

```
What are all of the photon SKUs and their list prices over time?
```

paired with this query:

```
select
 sku_name,
 pricing.default,
 price_start_time,
 price_end_time,
 usage_unit
from
 system.billing.list_prices
where
 upper(sku_name) like '%PHOTON%'
order by
 sku_name,
 Price_start_time;
```

or this one:

```
Are there any node types that have more than one GPU?
```

paired with this query:

```
select
 node_type,
 memory_mb,
 core_count,
 gpu_count
from
 system.compute.node_types
where
 gpu_count >= 2
order by
 gpu_count desc,
 core_count desc
```

Once you've configured the Genie space with the relevant system table information, it becomes your dedicated observability assistant. The Genie space generates SQL queries based on natural language input and presents the results in a chart, text, or table format, providing a streamlined way to manage your platform. Genie's advanced deep research capabilities enable it to analyze historical trends in conjunction with multiple queries, allowing it to identify user groups that would benefit from optimization training or recommend cluster configurations tailored to specific workloads. Figure 7-6 shows a Genie space curated to answer your questions on system table metrics.

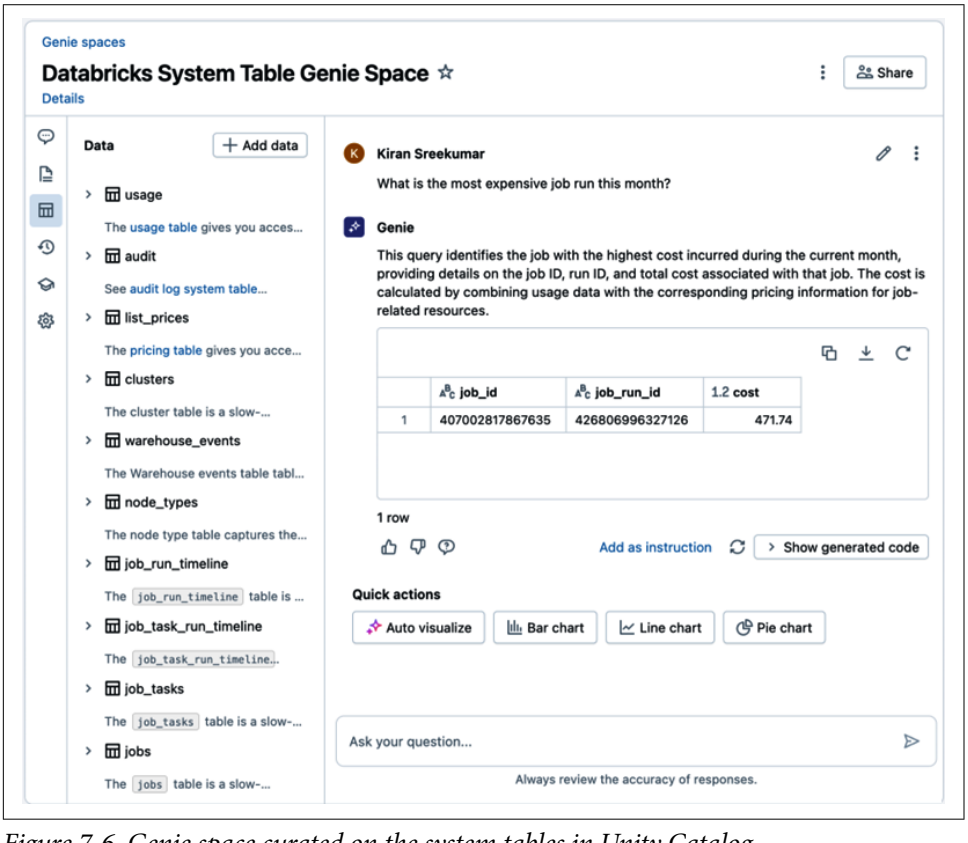


Figure 7-6. Genie space curated on the system tables in Unity Catalog

The CDP team at Nexa took a proactive approach to managing their Databricks platform, harnessing the power of system tables to meet their observability needs. By tapping into these tables, they created AI/BI dashboards that provided visibility into their workflows, compute usage, and spending, instantly flagging any outliers that required attention. The team curated multiple AI/BI Genie spaces to streamline

their operations further, allowing them to query the platform's status using natural language.

This data also fueled multiple dashboards within their Power BI ecosystem, giving them a unified view of their platform's performance. They set up SQL alerting capabilities in Databricks to avoid potential issues, sending notifications based on key metrics from the system tables. The backbone of this observability setup is that the serverless DBSQL warehouse and Unity Catalog work seamlessly to power all their Databricks Platform operations. For a closer look at Nexa's observability setup, check out Figure 7-7. The system tables in Unity Catalog are evolving with each new product category Databricks introduces, and you will see more and more categories coming up in these tables. System tables will remain the standard for observability in Databricks, and you can rely on them to build your platform observability requirements.

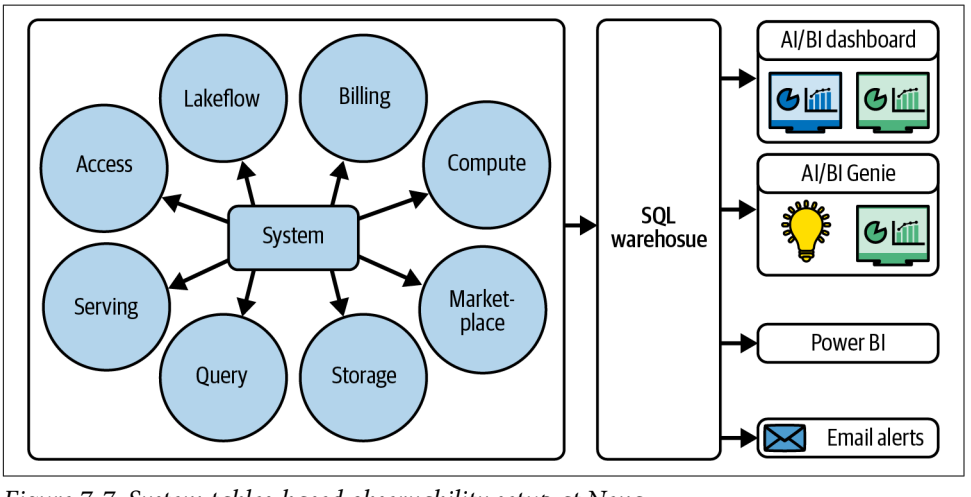


Figure 7-7. System tables-based observability setup at Nexa

Data Quality in Databricks

According to Gartner, "Poor data quality is also hitting organizations where it hurts—to the tune of $15 million as the average annual financial cost in 2017."

As per Dimensions of Data Quality, a research paper published by the Data Management Association, Netherlands chapter, the different aspects of data quality are broken down into six dimensions:

Consistency

Refers to the uniformity of data representation in terms of format, structure, and semantics, encompassing the use of consistent data types, field lengths, and naming conventions. It ensures that data is represented consistently across different datasets and systems, enabling reliable data integration and analysis. At Nexa, data consistency is crucial for providing accurate and up-to-date customer information across all systems and channels. For instance, consider a customer with an online account who is purchasing in-store. The customer's name, address, and contact information should be consistent across all of its systems, including:

1. Online account profile
2. In-store customer database
3. Loyalty program database
4. Customer service records

However, if the customer's name is listed as "Kiran Sreekumar" in their online account profile but as "K. Sreekumar" in the in-store customer database, this inconsistency can cause problems. For example, suppose the marketing team at Nexa wants to send a personalized email campaign to all customers who have purchased in the last 30 days. In that case, the inconsistent data may prevent the email from being sent to the correct customer.

Accuracy

Refers to how close the data is to the actual value. It ensures the data is free from errors, inconsistencies, and inaccuracies. Data accuracy is crucial for ensuring that customer and product information is correct and up to date at Nexa. If the product information, such as prices, descriptions, and inventory levels, is inaccurate, it can lead to customer dissatisfaction, returns, and lost sales. For example, if a product is listed as being in stock but is out of stock, the customer may place an order only to find out later that the product is unavailable. This can damage the customer's trust in the company and lead to negative reviews and word of mouth.

Validity

Refers to the extent to which the data conforms to the defined rules, constraints, and formats. It ensures that the data meets the specified standards and is valid according to the business rules and definitions. At Nexa, data validity is crucial for ensuring that customer and product information is correct and usable. For instance, the company's database contains customer information, including names, addresses, phone numbers, and email addresses. If a customer's zip code is invalid, it can cause problems with shipping and delivery. For example, suppose a customer enters a zip code that is not valid for their region. In that case,

the company's system may be unable to verify the address, leading to delays or errors in shipping.

Completeness
: Refers to the extent to which the data is comprehensive and includes all the necessary information. It involves ensuring that all required data is present and there are no missing or null values. At Nexa, data completeness for a product catalog can enable customers to make informed purchasing decisions. For instance, consider a product page for a gold necklace on its ecommerce website. A complete dataset for this product would include essential attributes such as these:

  1. Product description
  2. Price
  3. Specifications (e.g., color, carat, and purity)
  4. Images
  5. Delivery estimate
  6. Return policy

  The data can be considered incomplete if these essential attributes are missing. For example, suppose the product page does not include a delivery estimate. In that case, customers may be unable to make an informed decision about purchasing the product, as they will not know when to expect to receive it. On the other hand, if the product page includes all the essential attributes but is missing an optional attribute, such as a *product video*, the data can still be considered complete. The presence or absence of the demo video does not affect the customer's ability to make a purchasing decision, and the data is still sufficient to deliver meaningful choices.

Timeliness
: Refers to the extent to which the data is up-to-date and available when needed. It involves ensuring that the data is refreshed regularly and that there are no delays in updating it. At Nexa, data timeliness is crucial for ensuring that customer and product information is up-to-date and available when needed. If the data is not timely, it can cause problems with order fulfillment and customer satisfaction. For example:

  1. If the prices are not updated in real time, the company may charge customers incorrectly, leading to errors and customer complaints.
  2. Suppose the product availability information is not updated in real time. In that case, the company may display products as available when they are out of stock, leading to customer frustration and lost sales.

Uniqueness
: Refers to the extent to which the data is unique and free from duplicates. Each data point must be unique, with no duplicate records. At Nexa, data uniqueness is crucial for ensuring that customer and product information is accurate and up to date. It stores customer information, including names, addresses, phone numbers, and email addresses. If the data is not unique, it can cause problems with customer engagement and analysis. For example:

  1. If there are duplicate customer profiles, the marketing team may send multiple emails or promotions to the same customer, leading to customer frustration and decreased engagement.
  2. If there are duplicate product records, the website may display the same product multiple times, leading to customer confusion and decreased sales.

Data quality principles are extensively documented in authoritative works such as *Python Data Cleaning and Preparation Best Practices* (O'Reilly) by Maria Zervou and *Data Quality Fundamentals* (O'Reilly) by Barr Moses, Lior Gavish, and Molly Vorwerck.

Next, we'll explore Databricks-specific data quality capabilities enabled by Unity Catalog.

Lakehouse Monitoring

Even when your infrastructure monitoring shows green lights and workflows run smoothly, data quality issues can silently creep into your lakehouse, compromising your BI reports and eventually impacting your business. These issues often manifest as:

- Unexpected `NULL` values
- Duplicate records from multiple processing
- Other data integrity violations

While these problems might not immediately surface in aggregated reports, their impact grows as your dataset expands. Minor discrepancies can evolve into significant reporting inaccuracies over time.

Your lakehouse has robust built-in solutions to uphold the six dimensions of data quality, safeguarding against common data quality issues. For example, constraints such as `NOT NULL` are applied to ingested data to guarantee *accuracy*, preventing errors and inconsistencies. Meanwhile, schema enforcement and evolution

mechanisms ensure data *validity*, verifying that data conforms to predefined rules and standards.

Your lakehouse offers a range of features to ensure *completeness*, including atomicity, data enrichment, and metadata management. These work to ensure that all data is comprehensive and available. Delta's high concurrency and streaming capabilities also ensure *timeliness*, allowing for real-time data processing and analysis. Finally, *uniqueness* is guaranteed through various functions, including `Merge`, `Distinct`, and `dropDuplicates`, which eliminate duplicates and ensure that data is distinct and reliable.

Ensuring data quality, detecting anomalies, and resolving issues in a vast and complex data ecosystem can be challenging. The challenges multiply as the data landscape expands, especially when AI and ML models are integrated into the mix. Monitoring the performance of these models over time, identifying potential degradation, and analyzing the root causes of declining accuracy can be a monumental process to implement. The complexity is further compounded by the need to track model inputs, predictions, and outcomes, making it difficult to pinpoint issues and take corrective action.

The Profiles

Lakehouse Monitoring (LM) provides an automated solution for measuring data quality across your Databricks data intelligence platform. This solution tracks and verifies the accuracy and completeness of your data assets, including tables, materialized views, time series tables, streaming tables, feature tables, and model inference tables. LM offers data profiling capabilities, providing three distinct types of analysis to help you better understand your data:

Time series
: Analyzes tables with time-stamped data, applying a time series profile to uncover insights and trends.

Inference
: Applies to ML model inference tables, examining requests, timestamps, model inputs, and predictions to monitor model performance over time.

Snapshot
: Captures profiling metrics for an entire table, offering a comprehensive view of your data at a specific point in time.

The Baseline Table

When configuring LM for your table, you can establish a reference dataset or a baseline table, which is a yardstick for measuring data deviations and anomalies over time. This baseline table should contain a representative sample of your expected data, allowing you to gauge the quality and integrity of your data streams.

For time series profiles, select a baseline table that embodies a period of stable and normal data behavior where distributions align with your expected norms. This might be a week, month, or year where KPIs were within desired ranges. At Nexa, website traffic patterns were a time series dataset that the team monitored. The chosen baseline period was a summer month with consistent traffic volumes and engagement metrics. By establishing this benchmark, the team was able to detect subtle changes and anomalies in their website traffic data, enabling proactive interventions and data quality improvements. For example, if the website traffic suddenly spiked or dropped, they could quickly identify the cause and take corrective action.

When using an inference profile, utilizing the data used to train and validate your ML model as the baseline table is highly effective. To ensure seamless comparison, the baseline table should mirror the primary table's feature columns and include the same `model_id_col` specified in the primary table's `InferenceLog`, guaranteeing consistent data aggregation. At Nexa, the Logistics and Supply Chain group uses a ML model to predict sales and optimize inventory management. The team uses the model's training data as the baseline table, which includes features like seasonality, product categories, and regional demand. This allows them to monitor the model's performance over time and detect deviations from expected accuracy, precision, and recall. By comparing incoming data to the baseline, the team can quickly identify data and concept drift and retrain the model to maintain optimal sales forecasting and inventory management.

For snapshot profiles, select a baseline table that represents a snapshot of your data when the distribution of values was representative of normal operating conditions. At Nexa, while monitoring the product sales data, it set the baseline as the previous quarter when sales were distributed evenly across different product categories. By establishing this baseline, the team can monitor future sales data and detect deviations from expected norms. This enables them to identify potential product demand, inventory, or pricing issues and take corrective action to maintain optimal sales performance. Figure 7-8 illustrates LM in Databricks Unity Catalog tables.

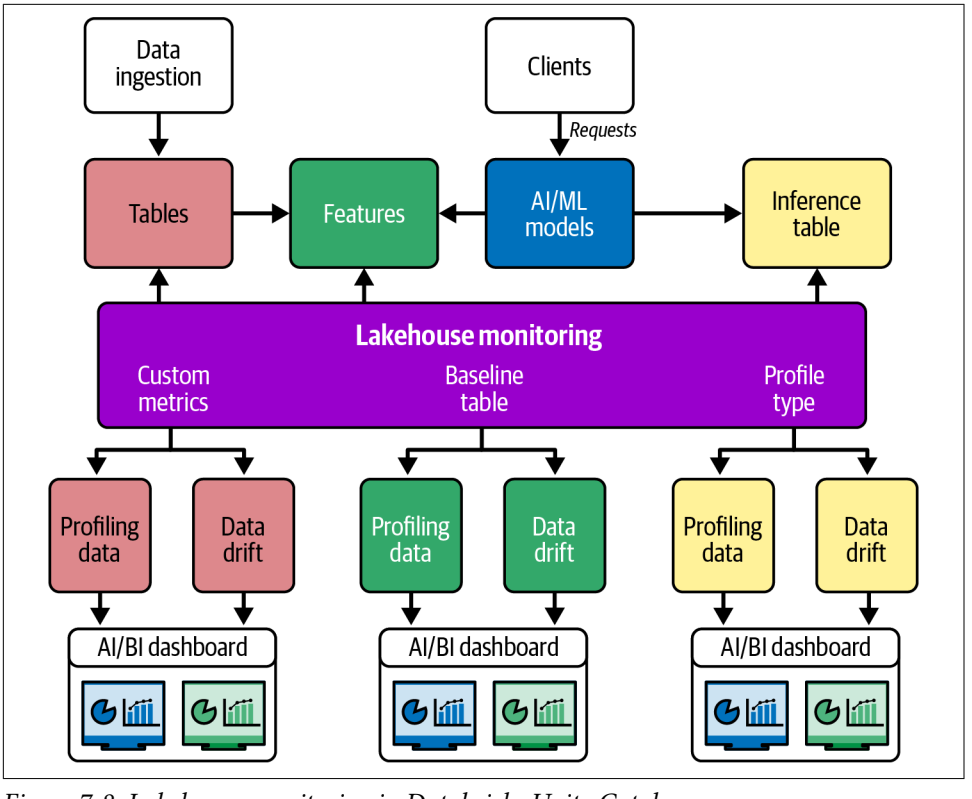


Figure 7-8. Lakehouse monitoring in Databricks Unity Catalog

The Monitoring Artifacts

Once you've configured your LM job, it runs seamlessly on Databricks serverless infrastructure. The monitoring refresh can be scheduled to run as per your requirement from once every minute to once a month.[1] It produces three valuable artifacts that reveal the story of your source table's data quality and integrity:

Drift metrics table
: A detailed table that captures the drift metrics of your data over time, providing a clear picture of how your data is evolving. This table includes:

Numerical drift tests
: Three powerful tests to detect changes in numerical distributions:

1 You also have the option to use the Quartz Cron syntax for extremely flexible scheduling albeit with a steeper learning curve.

KS test (Kolmogorov–Smirnov test)
: Measures the distance between two distributions.

PSI (Population Stability Index)
: Quantifies the degree of change in a distribution.

WD (Wasserstein distance)
: Calculates the distance between two probability distributions.

Categorical drift tests
: Four robust tests to detect changes in categorical distributions:

  Chi-squared test
  : Identifies significant changes in categorical frequencies.

  TVD (total variation distance):
  : Measures the difference between two categorical distributions.

  LID (Local item independence)
  : Detects changes in the relationship between categorical variables.

  JSD (Jensen–Shannon divergence)
  : Quantifies the similarity between two categorical distributions.

Profile metrics table
: A comprehensive table that profiles your data's characteristics over time, including:

  Categorical data insights
  : Count, distinct count, percentage distinct, minimum length, average length, and maximum length.

  Numerical data insights
  : Minimum, median, maximum, standard deviation, average, granularity, and more.

Monitoring dashboard
: A visual dashboard that brings your monitoring metrics to life, providing a clear and intuitive way to explore and understand your data's behavior.

These artifacts work together to completely understand the quality, integrity, and behavior of your data over time. This powerful combination allows you to confidently identify trends, detect anomalies, and make data-driven decisions.

Data Quality Monitoring

The new capability under data quality monitoring, which can be enabled at a schema level, monitors freshness and completeness. Freshness is based on analyzing historical commits made on a table and predicting the next commit time. If the next commit is longer than expected, the table gets tagged as stale. Completeness relies on a historical count of records and compares that to the number of records newly added within the last 24 hours. The table is assumed to be incomplete if any large discrepancy is found.

An automated job generates the monitoring logs and populates a table, which is then used to power a dashboard. Any freshness and completeness status are then fed back into the UI, which visually indicates the table's current status, as illustrated in Figure 7-9.

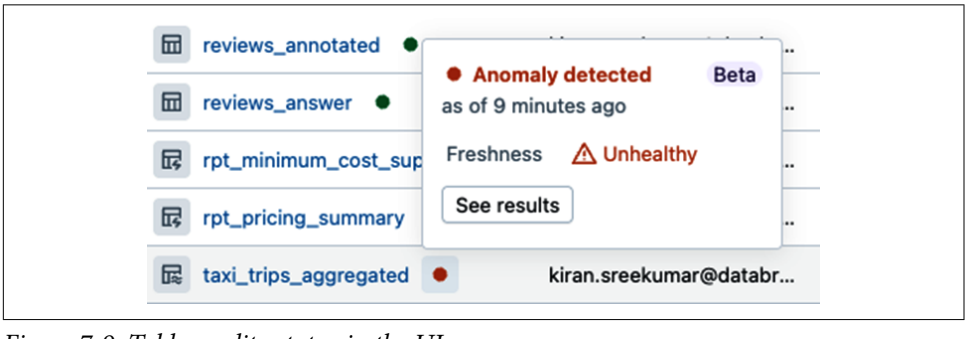


Figure 7-9. Table quality status in the UI

Discoverability in Unity Catalog

The importance of data discovery in unlocking data's value cannot be overstated. Companies like Uber, Airbnb, Netflix, Spotify, Meta, LinkedIn, and Lyft have developed data discovery tools to help users navigate complex data landscapes. These tools, such as Databook, Dataportal, Metacat, Lexikon, Nemo, DataHub, and Amundsen, have been instrumental in empowering users to find and utilize the data they need to drive business decisions.

However, not all companies have the resources or time to build their custom data catalog and enable data discovery for their end users. This is where Nexa's case comes in. Nexa's experience with using a Microsoft Excel sheet for data discovery is a common challenge many organizations face. As the data platform grows, the Excel sheet often becomes outdated, leading to productivity loss and bottlenecks in data innovation.

The statistics from Lyft's study are particularly alarming. Data scientists spend 25% of their time on data discovery, which highlights the significant amount of time

and resources that can be wasted on searching for and discovering data rather than analyzing and utilizing it, as illustrated in Figure 7-10.

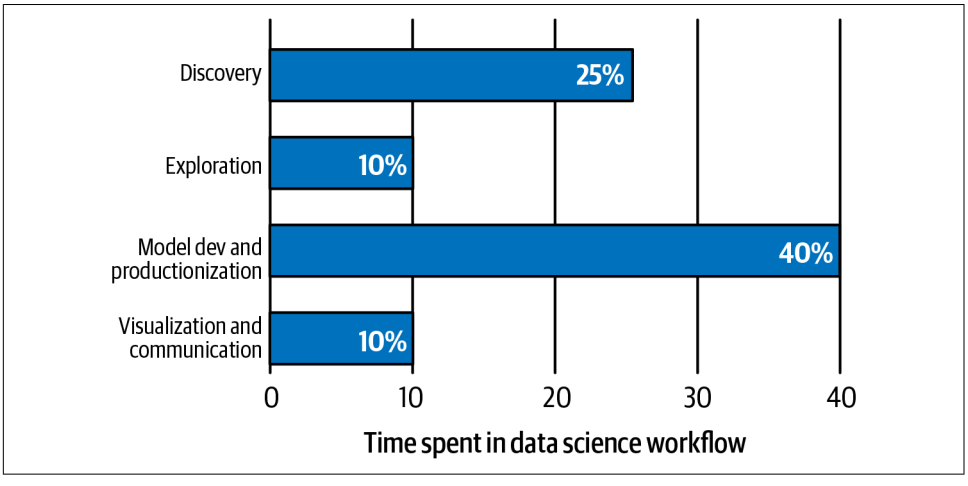


Figure 7-10. Time spent in data discovery as per a Lyft study

Your data discovery platform should be able to answer the common questions your users have about the data, like these:

- I want to build a new model. Does the data I require already exist in the data platform?
- What data does the dashboard I saw from the marketing team use?
- Who else uses the data I discovered? Is it the correct data that I can trust?
- What is the source of this data asset?
- How do I see the column- and table-level lineage?

If this sounds familiar, Unity Catalog is for you. Let's look now at how Unity Catalog enables data discovery.

Asset Description

When users search for a data asset, they typically look at the description to understand the data. This is because the description briefly summarizes the data, including its purpose, content, and context. A good description should be concise, clear, and informative, providing the user with enough information to determine whether the data is relevant to their needs. It should include details such as the following:

- What the data is about
- What type of data it is (customer, sales, etc.)

- What the data contains (columns, rows, etc.)
- Any relevant metadata (data source, update frequency, etc.)

By providing a clear and accurate description, users can quickly determine whether the data is what they are looking for and whether it is suitable for their intended use. This can save time and effort and help users decide which data to use. In a data catalog, the description is often the first point of contact between the user and the data. It is the first thing that the user will see when they search for a data asset, and it will often determine whether they will click the asset to learn more.

In Unity Catalog, a description field is available to capture information about your data asset. The descriptions provided in these fields are used for data discovery. Descriptions can be provided at the catalog, schema, model, table, and column levels. The intelligent AI-powered search engine lets you leverage natural language to find the most relevant data you are looking for. The more accurate descriptions you provide, the better search results you get.

But what if these descriptions were generated automatically for you? AI-generated documentation in Databricks provides an LLM-powered DatabricksIQ engine for adding descriptions and comments to tables and columns in Unity Catalog using models fine-tuned by Databricks for exactly this task.

Comments are audited in the system table `system.access.audit`, which helps you analyze the user or process, adding comments to an asset. Table 7-6 shows a sample output from the query against the audit table:

```
SELECT
 user_identity.email,
 action_name,
 request_params.comment,
 event_date
from
 system.access.audit
WHERE
 event_date BETWEEN now() - interval 3 day
 and current_date
 AND service_name = 'unityCatalog'
 AND (
   action_name in('updateTables', 'createTable')
 )
 AND request_params.comment IS NOT NULL
 AND user_identity.email = 'kiran.sreekumar@databricks.com'
ORDER BY
 user_identity,
 comment,
 event_date DESC,
 event_time DESC
```

Table 7-6. The `audit table` shows the user who added a comment against the table

email	action_name	comment	event_date
kiran.sreekumar@databricks.com	updateTables	Raw taxi trip data	2025-06-16

The AI-generated description for tables and columns can be updated to include your company-specific jargon. Adding additional information that the users will use as a keyword in their search to the description will enable better discoverability of your data. You can also use SQL to add custom comments to Unity Catalog assets:

```
COMMENT ON CATALOG sales_catalog IS 'This catalog contains sales data';
 --replaces existing comments
```

Definition: DatabricksIQ

DatabricksIQ is the intelligent AI engine that powers all of the innovative features in Databricks, including the assistant, AI-generated documentation, enhanced autoscaling, and predictive optimization. It is built as a compound AI system that understands your organization's data to make meaningful decisions to improve the platform.

Tagging

Tags are a powerful feature in Databricks that enable users to add metadata to their assets, making them more discoverable and searchable. By attaching key-value pairs to assets such as catalogs, schemas, tables, table columns, volumes, views, registered models, and model versions, users can provide additional context and specificity to their search results.

From a discoverability point of view, tags are handy because they allow users to narrow their search results to specific assets that match certain criteria. For example, a user can search for tables with a particular tag, such as `application:vendor_management` or `application:crm`, and the search results will show only tables tagged with those keywords. Tags are not limited to data assets but can also be applied to compute and workflows, allowing users to make cost attributions across different teams and business units.

Any compute and workflow tags are not governed tags and are not stored in the Unity Catalog metastore. This is particularly useful for organizations that track and manage costs across multiple departments or teams. A user with the `APPLY TAG` permission is allowed to add and edit tags in Unity Catalog:

```
ALTER TABLE
 catalog.schema.table
SET
 TAGS (
```

```
    'application' = 'vendor_management'
);
```

All tag-related information is available in the `information_schema` that Unity Catalog manages under each of your catalogs. The information schema holds the following tables with their corresponding tag details and serves as a central location to query for tag-related metadata:

- `INFORMATION_SCHEMA.CATALOG_TAGS`
- `INFORMATION_SCHEMA.COLUMN_TAGS`
- `INFORMATION_SCHEMA.SCHEMA_TAGS`
- `INFORMATION_SCHEMA.TABLE_TAGS`
- `INFORMATION_SCHEMA.VOLUME_TAGS`

Tags are audited in the system table `system.access.audit` that helps you analyze the user or process, adding tags to an asset:

```
SELECT
 *
FROM
 system.access.audit
where
 event_date BETWEEN now() - interval 3 day
 AND current_date
 AND service_name = 'unityCatalog'
 and (
   action_name = 'UpdateTagSecurableAssignments'
   OR action_name = 'UpdateTagSubentityAssignments'
)
```

AI-Powered Search

The search functionality in Databricks is powered by the DatabricksIQ intelligence engine, which utilizes comments and tags assigned to assets managed by Unity Catalog to provide a natural language-based search capability. This means that users can search for assets using everyday language, and the search engine will return relevant results based on the search query context. For example, the search query `"What tables contain sales data by region"` focuses on `"sales data"` and `"region"` and finds related terms containing sales attributes such as sales amounts, revenue, and geographic locations. The search engine understands the context of your business and returns relevant results, such as the following:

- A table containing sales data by *region*, with columns for sales amount, revenue, and region name

- A table containing sales data by *store location*, with columns for sales amount, revenue, and store ID

The comments and tags added to assets make the search engine context-aware of business-specific jargon and metrics. By incorporating this metadata into the search algorithm, the engine can understand the nuances of your business and return accurate search results relevant to the user's query. The search functionality in Databricks also supports filtering by specific tags in your data, allowing you to narrow down your search results to show only tables with a particular tag applied. Figure 7-11 shows an example of how search filters show the tables with the tag applied. The search results show only the tables with the `review` tag, making finding the information you need easy.

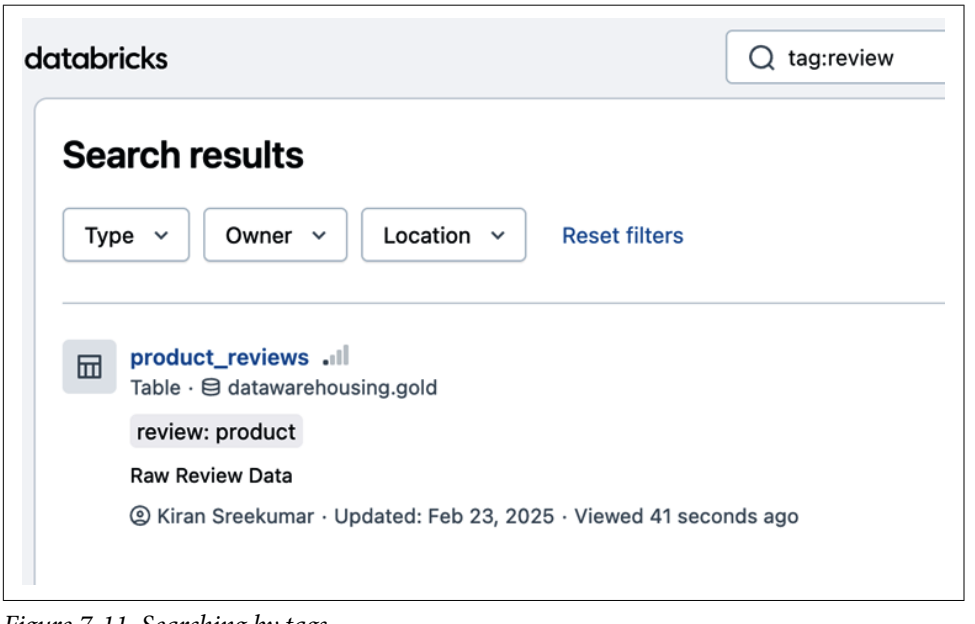


Figure 7-11. Searching by tags

BROWSE Privilege

Adding comments to assets and tagging them appropriately is a great way to enhance the search experience when using Unity Catalog to discover data assets for which you have been granted read or write permissions. An administrator can grant you the `BROWSE` privilege to enable data discovery on assets users don't have read or write access to. This privilege allows users to view an object's metadata, including the table schema and lineage, making it searchable while restricting data access. By granting the `BROWSE` privilege, an administrator can control who can discover and view your assets' metadata while maintaining the security and integrity of your data. This is

particularly useful in scenarios where you want to allow users to discover and explore data assets without giving them full access to the data.

The visibility matrix for a user with `BROWSE` privilege is as follows:

Catalog
: Details about the catalog, like metastore ID, who created the catalog, the storage root location, the owner, the schemas within it, the description, and any catalog-level tags applied.

Schema
: Details about the schema, including its ID, who created it, the owner, the tables within it, the description, and any schema-level tags applied.

Table
: Details about the table, its column names, data types, column-level comments and tags applied, who created the table, the owner, the lineage graph, any table-level tags used, and the table insights described below.

Volumes
: Details about the volume, including its ID, creator, description, and any volume-level tags applied.

Models
: Details about the model, including its ID, creator, description, and any model-level tags applied.

Functions
: Details about the function, including its ID, function metadata, the owner, description, language, and function-level tags applied.

When granted, the `BROWSE` privilege provides enough information for users to identify the data assets they need without exposing any real data or granting read-only access to the asset. The privilege enhances the integration of Unity Catalog with your existing enterprise catalogs for data discoverability.

Insights and Popularity

Another vital question a data discovery solution should answer is how the data is used across different applications and users. Unity Catalog generates insights on your table usage that show metrics like these:

- Usage of the table over the last 30 days
- Top 5 frequent users
- Top 5 frequent dashboards
- Top 5 frequently used notebooks
- Top 5 frequently joined tables
- Top 5 frequent queries

Insights into your data are crucial in helping you answer important questions about the data you're working with. For instance, when exploring an AI/BI dashboard created by the marketing team, you might wonder what data it uses. With data insights, you can quickly discover that the dashboard is pulling data from a specific table, which helps you understand the source of the data and its potential limitations.

When working with a new dataset, you might ask yourself, "Who else uses this data? Is it the correct data that I can trust?" Data insights can provide information about the frequent data users, including the dashboard's owners. This can increase your trust in the data, as you can see that it's being used by others who are likely to be familiar with its accuracy and reliability.

You first use natural language to search for related tables that power the dashboard based on its business context. Once a set of tables is identified, the frequent dashboards data under the insights for a table shows the dashboard names, and the frequent users show the users who frequently query the table. The usage against the table daily is also an indication of how often the data is used for reporting and BI purposes. The data quality indicator gives signals about whether the data is fresh and complete. A *Certified* tag indicates whether it's the most recent version of the data. These capabilities work together to give the best end-user experience while using Unity Catalog for data discoverability.

Unity Catalog captures a table popularity metric that appears in the catalog explorer against a table, which provides valuable insights into the usage and relevance of each table in the catalog. This metric is based on the number of times the table has been queried in the last 30 days relative to other tables in the catalog. By analyzing this metric, you can quickly determine which tables are most frequently used and relied upon by others in your organization. If the object you discovered has a high popularity score, you can be confident that you are looking at a widely used and trusted data asset, as illustrated in Figure 7-12.

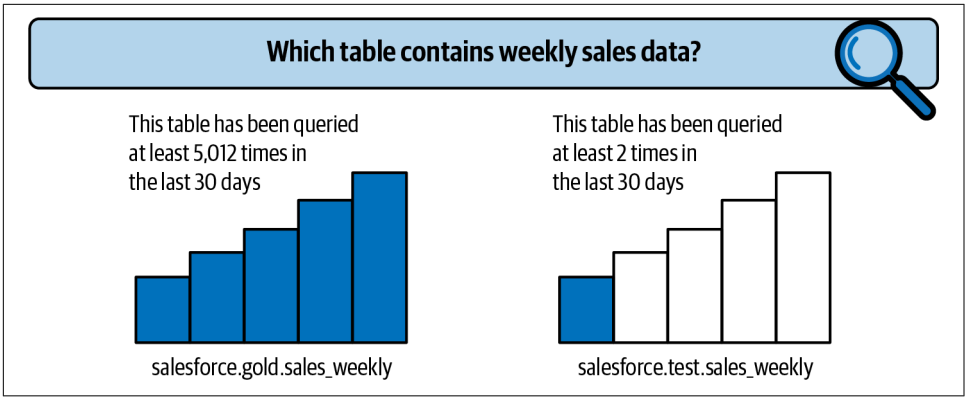


Figure 7-12. Table popularity in UC

Lineage

The Lineage feature in the Databricks workspace UI and system tables provides a powerful tool for understanding a table's column- and table-level lineage discovered in Unity Catalog. Lineage is intelligent enough to capture information about upstream and downstream dependencies with various assets in the Databricks workspace, making it a one-stop place for discovering dependencies and usage. With Lineage, you can easily see how a table is related to other assets in the workspace, including the following:

- Tables
- Notebooks
- Workflows
- Pipelines
- Dashboards
- Paths
- Queries
- Models
- Serving endpoints
- Functions
- Custom—bring your own lineage

Lineage serves as the early warning feature when problems arise with data. The data quality monitoring jobs scan data and detect discrepancies automatically before they spread through your system. Data lineage captures all upstream and downstream dependencies and can highlight the issue's full impact. Consider this scenario where the data quality indicates data freshness issues. You immediately see that your monthly revenue report, customer segmentation model, and marketing automation workflows all depend on this data. It indicates all business-critical workloads and reports that are impacted. Think of lineage as your data dependency map, where every table connects to other tables. Every report pulls data from various sources. You get visibility into the entire data landscape, helping you navigate precisely.

First and Last Mile Lineage

Any operations or integration with tools within Databricks are captured in the lineage automatically. But what if you want to see how your downstream system consumes the data or how the data was ingested from your upstream systems, be it a Kafka queue, database, web API, or a Power BI report?

The bring-your-own lineage feature allows you to specify an upstream source like Kafka and a downstream source like Power BI and their relationships with assets managed by Unity Catalog. You could also update the existing lineage to add any missing connections due to deleted nodes in the relation. You must be a workspace administrator with the appropriate Unity Catalog privileges to add or update the lineage relationships.

Lakehouse Federation

Lakehouse Federation (LF), the query federation platform in Databricks, enables you to connect to external data sources from Databricks with Unity Catalog's governance guarantee. LF is useful when you don't want to migrate the data from your external systems to Databricks but still need access to it during an exploratory phase of a new report or project, or to support infrequent access with lower cost impact. LF pushes down the execution of the query into the source database system, supporting FGACs, lineage, and search in Unity Catalog. It also brings metadata from various database sources, such as the following, into Unity Catalog, providing a centralized data discovery with all the smartness of Unity Catalog:

- MySQL
- PostgreSQL
- Teradata
- Oracle
- Amazon Redshift
- Salesforce Data Cloud
- Snowflake
- Microsoft SQL Server
- Azure Synapse (SQL Data Warehouse)
- Google BigQuery
- Databricks
- Hive metastore federation

Your federated sources behave exactly like native Delta tables. You query them the same way, apply the same transformations, and reference them in your pipelines without custom syntax or workarounds. Lineage tracking captures everything automatically. When you build a report using federated PostgreSQL data joined with a native Delta table, that connection appears in your lineage graph. Every transformation, every join, every aggregation gets tracked regardless of where the data lives physically. Access controls work the same way for federated and native tables. Data

classification tags work with federated sources just like native ones. The zero-copy approach eliminates data duplication and reduces storage costs.

Enterprise Catalogs

Unity Catalog integrates with major enterprise catalogs in the market, allowing you to access Unity Catalog metadata from an external catalog used within your organization. This integration is made possible through system tables and Unity Catalog APIs, which can be accessed by an external tool of your choice. For example, Microsoft Purview can integrate with Unity Catalog to display table metadata, including lineage information. Similar integrations are available with popular catalogs such as Alation, Atlan, Collibra, and Informatica. This enables a seamless metadata exchange between Unity Catalog and these external catalogs, providing a unified view of your organization's data assets.

Certification and Deprecation

Discovering and using the right data drives successful outcomes but using outdated information can destroy months of your work and waste significant resources. Unity Catalog solves this by guiding users to the correct data using system tags to certify and deprecate assets.

Certified tag

Data owners mark approved versions as certified using the system tag `system.certified`. The certified tag signals the latest version is available, meets internal accuracy standards, passes completeness requirements, and remains trusted for production use across all business operations, as shown in Figure 7-13.

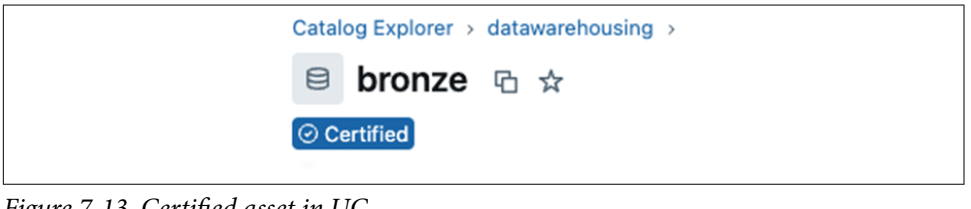


Figure 7-13. Certified asset in UC

Deprecated tag

Data owners can use the system tag `system.deprecated` to identify outdated assets, no longer reliable and unsuitable for new workflows, preventing teams from building solutions on deprecated assets, as shown in Figure 7-14.

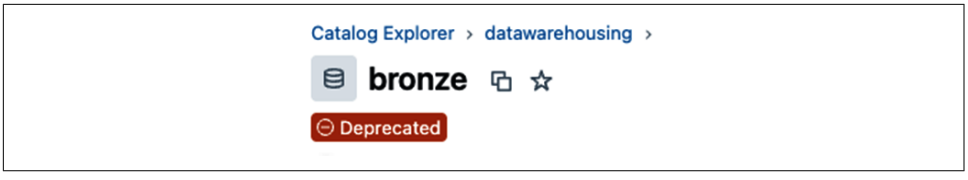


Figure 7-14. Deprecated asset in UC

Request for Access

`BROWSE` privilege enables discovery, but access to data requires permissions to be assigned to the discovered assets. The next logical step is an approval workflow that initiates a request by the requestor, notifies the approver, and grants the requested access.

The *request for access* feature enables the approval lifecycle by sending a notification to the access request destination, which an administrator can set. Destination support includes an email alert, a Slack or Microsoft Teams alert, and a generic webhook option that could trigger the request to any supported destination URL, including common ticketing and workflow tools like Jira and ServiceNow.

The discoverability capabilities in Databricks Unity Catalog are illustrated in Figure 7-15.

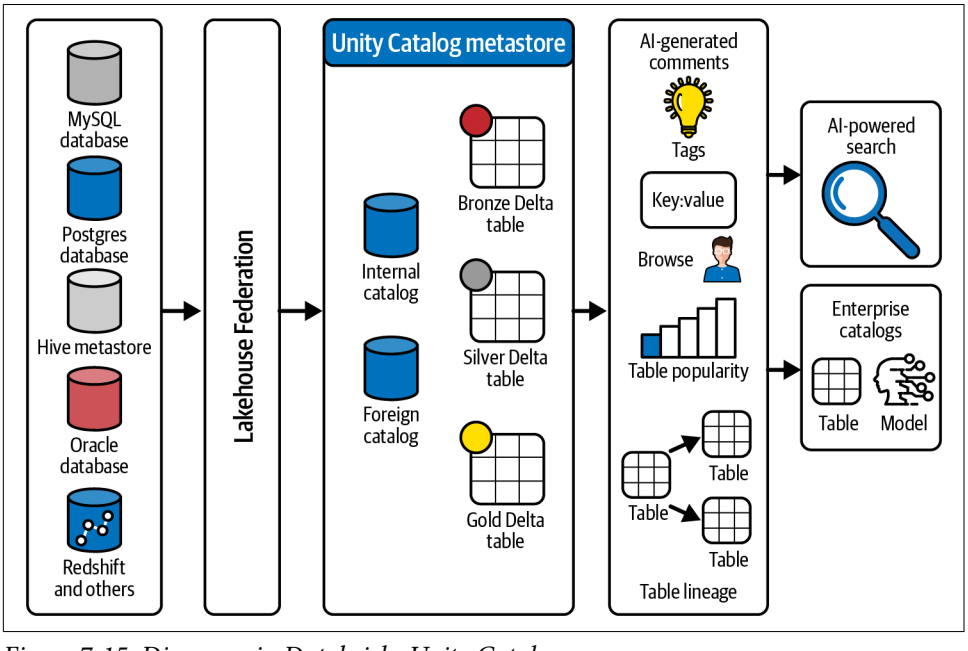


Figure 7-15. Discovery in Databricks Unity Catalog

Summary

This chapter highlighted the importance of observability and discoverability in a data platform and how Databricks excels at them through the functionalities enabled by Unity Catalog. It provided a detailed overview of how system tables improve your platform observability, help reduce spending, and increase team productivity. The chapter also discussed the aspects of data discoverability and how Unity Catalog enables it through AI-powered search engines. Overall, Unity Catalog provides a robust foundation for observability and discoverability in Databricks, and its capabilities will continue to expand and improve over time. Chapter 8 discusses how you can share data and collaborate on the lakehouse, all enabled and governed by Unity Catalog.

CHAPTER 8
Data Sharing and Collaboration

> *Homo sapiens rules the world because it is the only animal that can cooperate flexibly in large numbers.*
>
> —Yuval Noah Harari (*Sapiens: A Brief History of Humankind*)

In his countless conversations with customers, Karthik began to detect a pattern: decision makers across organizations, especially large ones with a presence across different geographies, tended to look for a particular criterion in data platforms: how easy it was to securely share data and collaborate.

This pattern has two main aspects to it: one is about publishing assets targeting one or more consumers. The other is about sharing data and other assets for collaborating on business use cases, which can be as simple as sharing what you own with your teammates, or as complex as collaborating with other organizations across geographies on a use case that involves sensitive data. The data platforms that offer these features are favored by the decision makers. Most organizations are paying the price for not having considered these aspects in the past, which has led to silos resulting in a lack of collaboration and duplication of assets and efforts.

We are indeed happy that organizations are on the right path. However, the road to success with data sharing and collaboration is a hard one, especially in terms of security and governance. Before we discuss the associated challenges, let's first clarify some of the terms that we use.

Similar to the term *data governance,* which we extended to include AI governance, we intend to use the term *data sharing* for sharing data, AI, and other related assets. This includes structured and unstructured data assets such as tables, views, and arbitrary files, as well as AI assets such as models, functions, and notebooks. We use the term *collaboration* to refer to use case-related efforts across OUs or across organizations that leverage data sharing capabilities. To refer to the aspect of sharing data and

AI assets as read-only and with the intention to reach one or more consumers, an approach typically used in *data mesh* and *data marketplace* architectures, we employ the term *data publishing*.

When you imagine the term *data sharing*, what comes to mind? For Karthik, it's downloading a dataset as Excel files and copying it to an SFTP server—at least, that's what he used to imagine. It might be the same for you. SFTP remains one of the most common ways to share data. Nevertheless, there are other options, such as Google Drive, iCloudDrive, OneDrive, Dropbox, WeTransfer, and so on—and for those who are more tech savvy, Amazon S3 and CloudFlare R2 are also feasible options. If this is the case for regular individuals, imagine how limited the options are for organizations that are bound by policies, compliance requirements, and need to evaluate and approve every new technology they use.

The ability to securely share assets has a direct impact on collaboration. If your data team is spending time deciding how best to share a dataset or an ML model with a colleague, that's lost productivity. If you don't figure out a way to securely share data with your partners or suppliers, that's a lost opportunity. Research and development teams and startups lacking access to the right data at the right time means lost innovation. Why is this the case? Why is data sharing so hard?

Data sharing involves at least two parties. One is sharing the data; let's call them *providers*, and the other is consuming the data—*recipients*. Let's now consider a simple example of a private hospital chain that is working with a startup to help them with demand forecasting and capacity planning. For this purpose, the startup needs access to the hospital data, which is stored on a PostgreSQL instance in Microsoft Azure. In this case, the hospital chain is the provider and the startup is the recipient. The startup has enough funding to choose any technology it needs.

How should the hospital share the data with the startup? It cannot give direct access to its database instance to the startup, as it has strict compliance requirements. Moreover, it is allowed to share only anonymized datasets where sensitive information, such as PII, has been masked. Given these requirements, the hospital's dev team working on this project decided to do what anyone else in their position would have done. They wrote a script that reads the required tables, anonymizes the PII columns, exports them into a CSV file, and dumps them on an SFTP server where the startup can connect and download those files. It worked as the dev team expected. But, as you can imagine, neither the hospital's IT security team nor the startup was happy.

A quick look at the issues associated with this example will give us an insight into the major challenges with data sharing:

Data freshness
: The datasets shared by the hospital are static. This means if the startup needs the latest data, then the hospital has to run the script again. They could, of course, set

up periodic automated execution of the script, in which case the data is as fresh as the execution schedule. On the startup side, they have to reconcile the latest data with the existing data that they have. In essence, this setup needs additional effort as well as maintenance overhead.

Scalability
: What if the startup needs more datasets, including all the historical datasets? Depending on how big the hospital chain is and how much historical data it has, the volume of the data could be high. Moreover, the script that fetches the data from Postgres and dumps the CSV should also need to be updated. With the requirement for more data, the effort required to make it available also grows. There are also specific limits with SFTP servers when dealing with large files. Hitting the database servers with queries quite often isn't always the best approach to exporting data, as there is a risk of daily hospital operations being affected. Imagine a developer writing an inefficient SQL query that overloads the database.

Security
: This is a significant issue. Because the hospital is sharing the data as an export, once shared, it has no control over who has access to it. Moreover, if someone can get access to the SFTP servers, they can easily access the data, which can result in a compliance breach. It is hard to enforce any kind of security best practices, such as the principle of least privilege and the zero-trust access model. It is also hard to monitor or track events or activities, as there is a lack of a proper logging setup.

Beyond such a proof-of-concept setup, there are also issues with maintainability, reliability, and costs. You might be wondering, what should the hospital do to mitigate all of this? Is setting up a secure, reliable, and scalable platform just for one use case really worth it? More than that, is it really possible to mitigate these issues, or is the only way forward to live with the trade-offs? The point of this example is to help you comprehend the challenges that come with data sharing and what aspects you need to consider when designing such a solution. It is indeed possible to mitigate these, but whether the hospital should or can invest in it is a question that depends on several nontechnical factors, which we won't discuss here.

In this chapter, we want to show you the data sharing and collaboration features available with the Databricks Platform. We will see how you can deal with data-sharing challenges and how best to publish data and AI assets across geographic regions, cloud providers, and organizations. We'll also look at how you could collaborate on a use case that extends beyond organizational boundaries. And you will learn cross-metastore governance, sharing patterns, and best practices.

Databricks Data Access Patterns

Sharing data can be viewed as granting access to the data that you own or manage. This can be *in-place* sharing, where the recipient gets access directly to the data, or it can be *data-copy,* where the recipient gets access to a copy of the data in the form of tables made available through a platform or an export of the data in the form of files. In the context of Databricks Unity Catalog, we focus only on the in-place access to data because it is more secure, scalable, and maintainable.

There are two main aspects that determine the best way to access the data on Databricks:

- Where does the data live?
- Where are you accessing the data from?

As we saw in "Databricks Constructs" on page 71, you can register the data in the cloud object storage as tables or volumes within Unity Catalog's *standard* catalogs or register the tables living in external data sources under Unity Catalog's *foreign* catalogs.

In this book, we have extensively discussed standard catalogs. You store and access data in cloud object storage using external location and storage credential objects and associate it with standard catalogs. The concept of *foreign* catalogs, on the other hand, is part of the Lakehouse Federation (LF), which we briefly discussed in Chapter 7.

LF enables a uniform interface for users to access the data across different data systems, along with the governance features of Unity Catalog. It essentially allows you to query the data in external systems, without moving the data. However, it's important to keep in mind that it might not be suitable for all kinds of use cases. Karthik has seen a few Databricks customers trying to use it for ingesting data from other systems, which did not work well in terms of query performance and costs. LF is suitable for and intended to be used for the following purposes:

- Doing a proof-of-concept on Databricks using datasets present in external data sources
- Performing exploratory data analysis to determine the right approach for ETL or analytics
- Ad hoc reporting or bridging any gaps during migrations from the source system to Databricks

If you want to ingest data from the source system, in which case you will be moving the data, it's better to use Lakeflow Connect, which offers manageable connectors. These connectors scale to accommodate high data volumes and enable low-latency querying.

On Databricks, typically, the data should live closer to the compute layer. In other words, the region where the cloud object storage is configured and the region where the Databricks workspace exists should be the same. For example, if your data sources are in a particular cloud region, say Azure North Europe, you spin up an Azure Databricks workspace in the same region and create storage accounts in the same region to store and process data. It's rarely the case where you have a workspace in the North Europe region, but your data sits in the Central US region, for example. Technically speaking, it's possible to have that setup—you can create storage credentials and external locations that span cloud regions—but you will incur storage egress costs and experience higher latency. Therefore, it's not recommended unless you have a specific reason to do so. For example, you have a small dataset that is created in a different region, and you don't want to create a Databricks workspace and a metastore just to catalog that small dataset. Figure 8-1 shows the typical setup of workspaces and storage accounts.

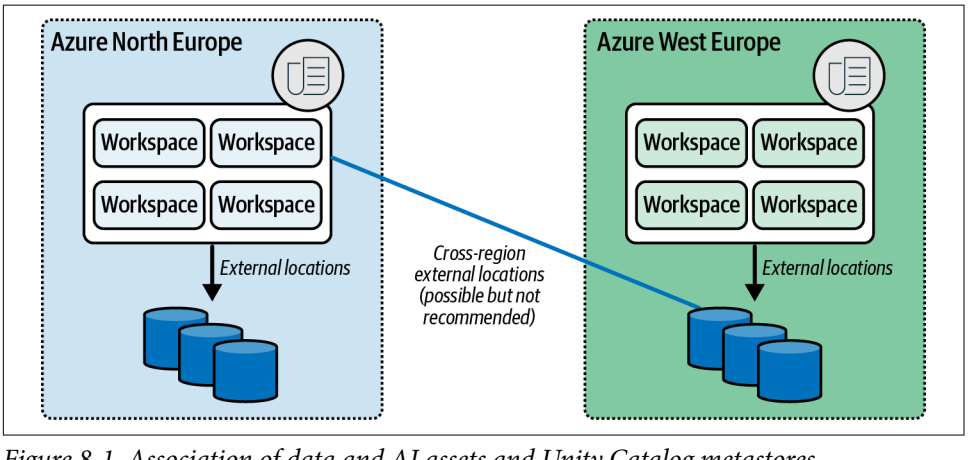


Figure 8-1. Association of data and AI assets and Unity Catalog metastores

You have multiple options for accessing data and AI assets from Databricks. Most of these options are based on the Databricks workspace context, as explained in Chapters 3 and 9. This means the recipient who wants to access the assets should be added to the Databricks account as well as the workspace as a user. Alternatively, you can use an SP, with which the recipient can access the assets. The recipient can be a human user, a compute engine, a BI tool, and so on.

As a Databricks user, accessing data and AI assets cataloged in the corresponding Unity Catalog metastore is simple and straightforward. The complexity arises in the following cases:

- The recipient is not a Databricks user.
- The recipient is a Databricks user but is part of a Databricks workspace in a different region, cloud, or Databricks account.

Delta Sharing, briefly mentioned in Chapter 1, can address both of these. So, let's dive deep into the world of Delta Sharing and see how it helps with cross-region, cross-cloud, and cross-account data sharing and publishing.

Data Sharing and Publishing with Delta Sharing

We have now transitioned from the topic of data access to data sharing and publishing. As mentioned earlier, data *sharing* means granting access to data that you own or manage, but it's usually associated with read-write access. Data sharing and publishing are usually associated with read-only access to data.

Delta Sharing provides a way to share data and AI assets, and it is read-only for the recipients. Its design makes it well-suited for data sharing and publishing use cases. The best thing about it is that it facilitates in-place sharing, even when you want to share data across regions and clouds. In other words, Delta Sharing is zero-copy sharing. Moreover, once shared, you can continue to update the data, and the recipient can retrieve the latest data from the provider without the provider needing to reshare.

On a high level, Delta Sharing works as follows:

1. The provider shares the metadata through a Delta Sharing server.
2. The recipient connects to the Delta Sharing server, fetches the metadata, including a short-lived signed URL or a cloud token for accessing the cloud object storage.
3. The recipient uses the information that it retrieved from the server to fetch the data directly from the cloud object storage.

Figure 8-2 illustrates how Delta Sharing works. This approach of directly fetching data from cloud object storage makes it suitable for handling large volumes of data. You can use it to share arbitrary files and ML models in addition to structured data in the form of Delta tables, making it flexible and scalable in terms of use cases.

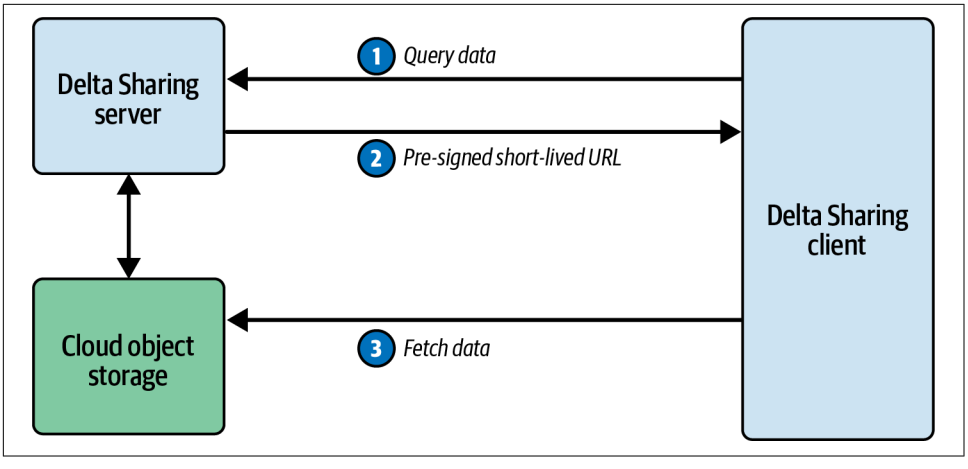


Figure 8-2. Delta Sharing: Under the hood

Although Delta Sharing is available as a managed service within the Databricks Platform, it is based on an open source protocol and comes in four different flavors:

Databricks to Databricks (D2D) sharing
: The provider and the recipient are both on Databricks. This is relatively straightforward to set up, significantly easier to maintain, and usually has more features than other flavors. If you're on Databricks, this should be the first choice for sharing data and AI assets across the regions, clouds, and accounts, provided the recipient is also on Databricks, as both sides can leverage out-of-the-box security and governance features offered by the platform.

Databricks to Open (D2O) sharing
: The provider is on Databricks, and the recipient is a non-Databricks user or group of users. This one is easier to set up from the provider side, but relatively more effort is required to set up and maintain on the recipient side. Authentication got a level-up when OIDC Federation support for the recipient was released. This means you can use the recipient's identity provider to authenticate while accessing the data shared using D2O sharing, and you do not have to rely on long-lived tokens, which are not as secure.

Open to Databricks (O2D) sharing
: The provider is not on Databricks and hosts its own Delta Sharing server, and the recipient consumes the shared data from Databricks. This setup is rare, but we have seen it implemented on a large scale, and it worked seamlessly. This can also be relevant in cases where you're using Databricks as a provider but do not wish to expose your Databricks setup to external recipients.

Open to Open (O2O) sharing

In this setup, neither the provider nor the recipient is on Databricks. Delta Sharing, being open source, enables such a setup that organizations can leverage. This would have been an option for the hypothetical hospital chain use case, discussed at the beginning of this chapter, as the hospital would not need to spend money on adopting a platform such as Databricks, but would nevertheless benefit from the technologies associated with it.

Delta Sharing is one of those things that just works. After experiencing the technology firsthand and having witnessed several Delta Sharing-based use cases go live, we are convinced of its potential and truly believe that it will be adopted across the industry with and without Databricks. Some Databricks customers have also started using Delta Sharing as the foundational component for their custom data-sharing solutions.

We will explore the details of D2D sharing in-depth later in this chapter because it's the most relevant setup for discussing governance in the context of Databricks Unity Catalog.

Data Governance Beyond Metastores

When you have a single metastore, it's relatively simple to define and manage access to data and AI assets and benefit from all the features provided by Databricks Unity Catalog, such as lineage. Chapter 5 went into the details of catalog layout, data sharing, and distribution within a single Unity Catalog metastore, and this is sufficient for most organizations, especially smaller ones and individual business units within large organizations. However, when you start scaling and want to share data with other business units present in another cloud region or who chose to go with another cloud provider, then you might hit some roadblocks. Therefore, we need to go over the details of secure cross-region, cross-cloud, and cross-account data sharing in the context of the Databricks Platform. To use a consolidated terminology, we can call this *cross-metastore data sharing*, and the best way to achieve this is using D2D sharing, depicted in Figure 8-3.

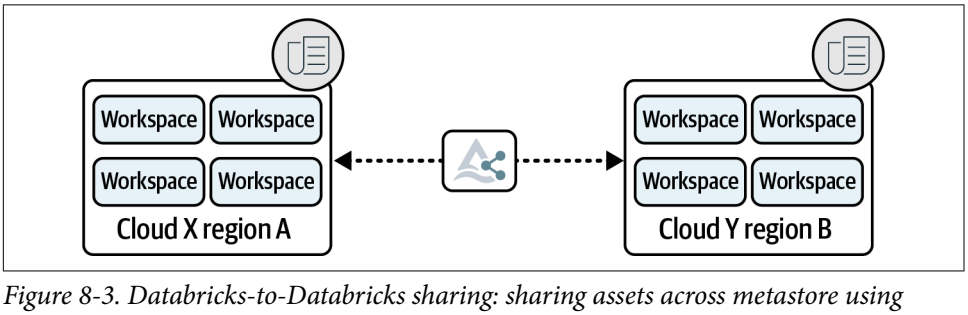


Figure 8-3. Databricks-to-Databricks sharing: sharing assets across metastore using Delta Sharing

To reiterate: Delta Sharing is for sharing assets as read-only. If you want to *collaborate* with the recipient—that is, allow them to write or update the assets—it's not possible with this approach. In other words, D2D sharing is focused on publishing or sharing use cases, such as in data mesh architectures, and not on granting access for writing across metastores.

Why Delta Sharing?

There are several ways to share/consume data across metastores. Apart from Delta Sharing, you can directly connect to a cluster or a SQL warehouse within a Databricks workspace, use Lakehouse Federation as it supports Databricks workspace as a data source, or, more recently, use Unity Catalog REST APIs. Although technically speaking, all these are valid options, let's see why most customers choose Delta Sharing and why it's best suited for cross-metastore data sharing.

Previously, we saw some of the advantages of using Delta Sharing. Now let's see how it fits in the context of an enterprise. When we consider the most important aspects of a data sharing setup in an enterprise, often, three central themes tend to surface: simplicity, scalability, and security.

Simplicity

With Delta Sharing, you have a framework for data sharing that requires the least amount of effort to set up and maintain. It's an approach that works and caters to all kinds of use cases, be it sharing datasets for BI use cases or sharing unstructured data and models for AI use cases. D2D Delta Sharing fulfills these criteria. You can set it up pretty easily and share both data and AI assets without moving mountains. Moreover, you can leverage the data security and governance features offered by Unity Catalog, although separately, for providers and recipients. Moreover, any updates by the provider will be immediately reflected on the recipient's side. This reduces the risk of data inconsistency and ensures everyone is working with the latest information.

Scalability

Delta Sharing is suitable for sharing large volumes of data. Although we might need to consider different sharing and consuming patterns, such as maintaining a managed replica of datasets on the recipient side to reduce costs and latency, the underlying technique remains the same. You can extend your sharing setup to include more recipients, without putting in much effort. If you have the need, you can extend your sharing approach to the recipients that are external to your organization. With Delta Sharing, being open source, a recipient does not necessarily need to use Databricks. In essence, the data-sharing setup based on Delta Sharing can scale in terms of the number of recipients as well as data volume. Nevertheless, there

are some platform-specific resource limits/quotas that you should consider when designing your architecture.

Security

Using the Databricks Delta Sharing means you can leverage the Databricks Platform security for sharing data and AI assets. The security aspects cover identity management, access controls, metadata handling, etc. Moreover, the way D2D Delta Sharing works makes it inherently secure, as you don't need to manage any secrets/tokens. This is not to say that other data sharing/consuming approaches on Databricks are less secure. Irrespective of which approach you choose, you still benefit from the platform's security and governance.

Delta Sharing fulfills all these criteria, and despite some limitations, it's best suited for developing a stable and reliable sharing framework within and across organizations. Alternatively, you can use Lakehouse Federation, which is much simpler in terms of setup and also enjoys the platform-provided security and governance features. However, in our experience, it is best suited for sharing low-volume assets, such as aggregated data.

D2D Sharing Under the Hood

In D2D sharing, as the name implies, both the provider and the recipient should be on Databricks. The *provider* is the one who shares/publishes the data, and the *recipient* is the one who consumes the data. Depending on the use case, a metastore can act as a provider or a recipient. While metastore-to-metastore sharing within a single Databricks account is enabled by default, to share data across metastores in different accounts, the account admin needs to explicitly enable it. To share an asset, which can be a table, view, model, file, notebook, or volume, you first create an object called `SHARE` and add your assets to it. You can also share a schema, which means you're sharing everything under that schema. To identify the recipient, you create a `RECIPIENT` object with the recipient metastore ID. Note that, in D2D sharing, you don't need to explicitly create a `PROVIDER` object on the recipient metastore; it's automatically created when you share assets on the provider side.

With D2D sharing, you are sharing your assets with another metastore and not with an individual or a group. Principals with relevant privileges on the recipient metastore can use the `SHARE` and grant access to others.

Any schema changes to the shared assets—for example, if you add a column to a shared table—are automatically reflected on the recipient side. If you have shared a Unity Catalog schema, then any new asset added to the schema is automatically shared with the recipient.

From the network perspective, D2D sharing works as follows:

1. Once you grant a `SHARE` to a recipient, it is visible on the recipient's side. Databricks automatically propagates and manages the metadata of shared assets.
2. When a user on the recipient side tries to access the asset, the Databricks compute instance on the recipient side has to directly fetch the contents from the cloud storage using the short-lived presigned URLs or temporary tokens. Figure 8-4 depicts the D2D setup.

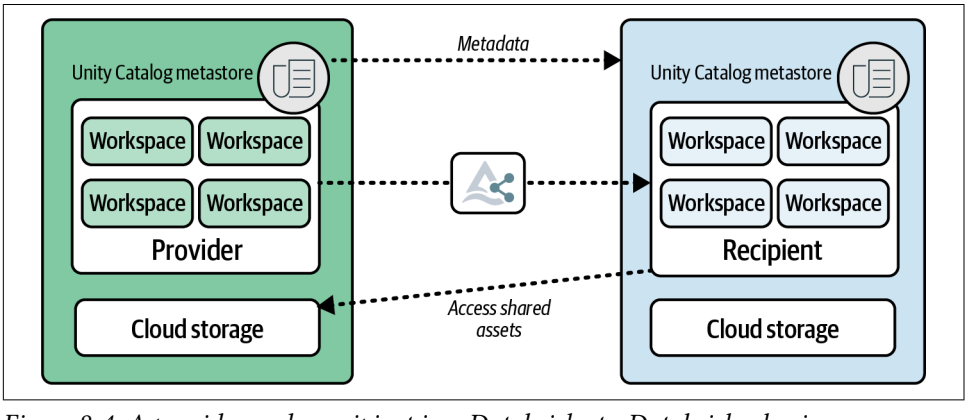


Figure 8-4. A provider and a recipient in a Databricks-to-Databricks sharing

Cross-metastore data sharing requires you to set up network access from the recipient's compute network to the provider's storage. If you have a provider on Azure Databricks and the assets you shared are stored on a firewall-enabled storage account, then you need to configure service endpoints or private endpoints from the recipient's Databricks workspace network. For a cross-cloud setup, you need to establish cross-cloud network connectivity.

Ownership and Privileges

To use a SHARE object on the recipient side, the metastore admin or a principal with CREATE CATALOG and USE PROVIDER privileges should create a catalog from it. This catalog is of a specific type and differs from the regular Unity Catalog catalog in certain ways. On the Catalog Explorer of the Databricks UI, it is listed under Delta Shares Received. As this is a read-only object, you cannot create or update any assets under this catalog. A share can include assets from different catalogs on the provider side. This means, in terms of access controls, you need to treat a SHARE as a different product, a derivative of your assets, to be cataloged and used under a different catalog. Moreover, when you share it with the recipients, the recipient decides who gets access to it. You have no control over which principals get access to the share or the underlying asset on the recipient side, unless, of course, you're also managing the recipient side. It helps if you have a well-defined catalog layout in the provider and the recipient metastores to keep things neat and tidy. For details on how to do that, refer to Chapter 5.

To simplify access management, we recommend using two groups: on the provider side, a group that has the privileges to create and manage SHARE and RECIPIENT objects, say ms_<cloud>_<region>_ds_provider_admins, and on the recipient side, a group that has the privileges to create catalogs and use shares, which we can call ms_<cloud>_<region>_ds_recipient_admins. Note that *cloud* and *region* are placeholders. Table 8-1 shows the group and privilege mapping.

Table 8-1. Proposed Delta Sharing admin groups and corresponding privileges

Group name	Privileges
ms_cloud_region_ds_provider_admins	CREATE SHARE, USE SHARE, SET SHARE PERMISSION, CREATE RECIPIENT
ms_cloud_region_ds_recipient_admins	CREATE CATALOG, USE PROVIDER

One more important thing to keep in mind is that you need to be the *owner* of a SHARE object if you want to add assets to it. Additionally, you have to have read permissions—that is, SELECT on structured assets, READ VOLUME on volumes, EXECUTE on models, and USE CATALOG and USE SCHEMA privileges on the catalogs and schemas that contain them. To grant access to a RECIPIENT on a SHARE, it's necessary to have a USE RECIPIENT or be the owner of it and have USE SHARE + SET SHARE privileges on the SHARE or be the owner of it.

Given this permission model, a good way to set up permissions is as follows:

- Use groups instead of individuals as owners of `SHARE` and `RECIPIENT` objects.
- The provider admin group (`ms_<cloud>_<region>_ds_provider_admins`) creates and retains the ownership of the `RECIPIENT` objects.
- The provider admin group creates `SHARE` objects, grants them to the `RECIPIENT` object(s), and transfers the ownership.
- The OU that owns the assets to be shared should be the owner of the `SHARE`.
- If a `SHARE` consists of assets from more than one OU, the ownership needs to be agreed upon among themselves.
- The OUs have no privileges on the `RECIPIENT` objects.

This kind of ownership/privilege model provides flexibility for the OUs that share their assets with the right amount of restrictions to prevent accidental sharing and misuse. This ownership model is depicted in Figure 8-5.

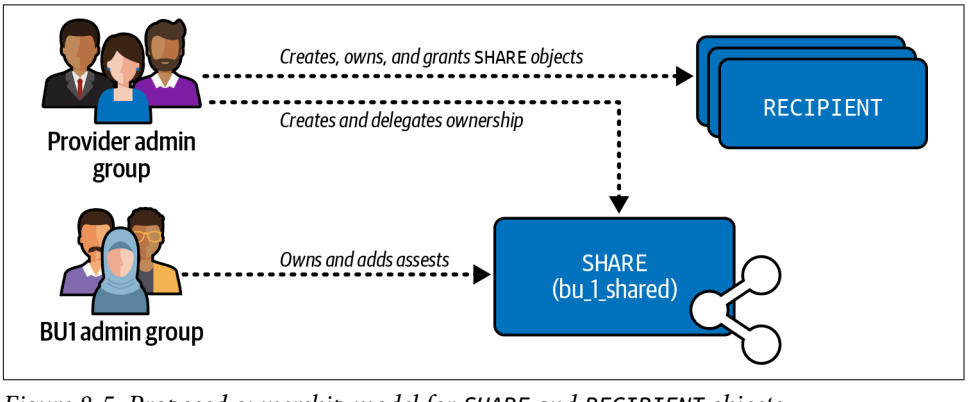


Figure 8-5. Proposed ownership model for `SHARE` and `RECIPIENT` objects

Catalog Layout

In terms of catalog layout, let's say you want to share assets from an OU-specific published catalog. In that case, you create a `SHARE` object and add the assets to that `SHARE`. Then you grant permission to the respective recipients to access that `SHARE`. This is shown in Figure 8-6.

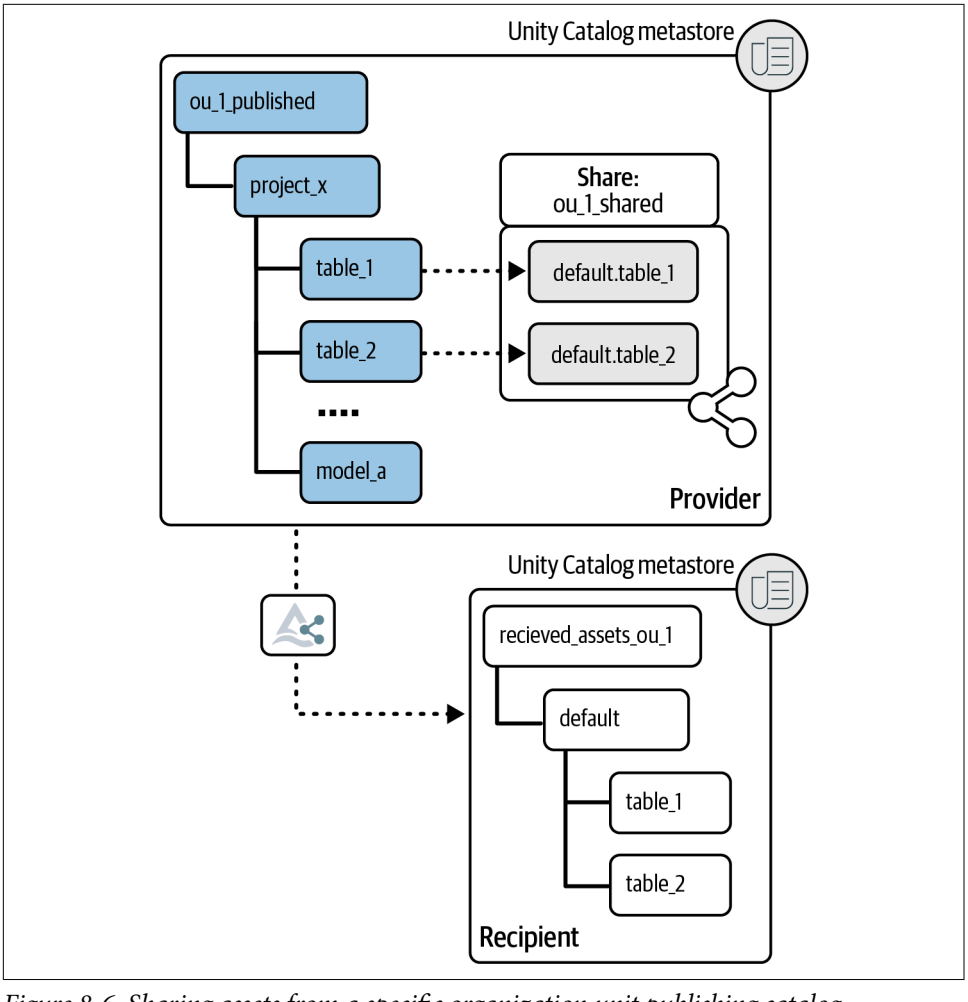


Figure 8-6. Sharing assets from a specific organization unit publishing catalog

Sometimes, you might need to share assets that are in different catalogs. Of course, you can create multiple `SHARE` objects, one per catalog. However, depending on the use case, you can also combine these assets into one `SHARE` object for sharing. For example, you might want to add assets from a specific OU and central publishing catalogs. Figure 8-7 shows this approach.

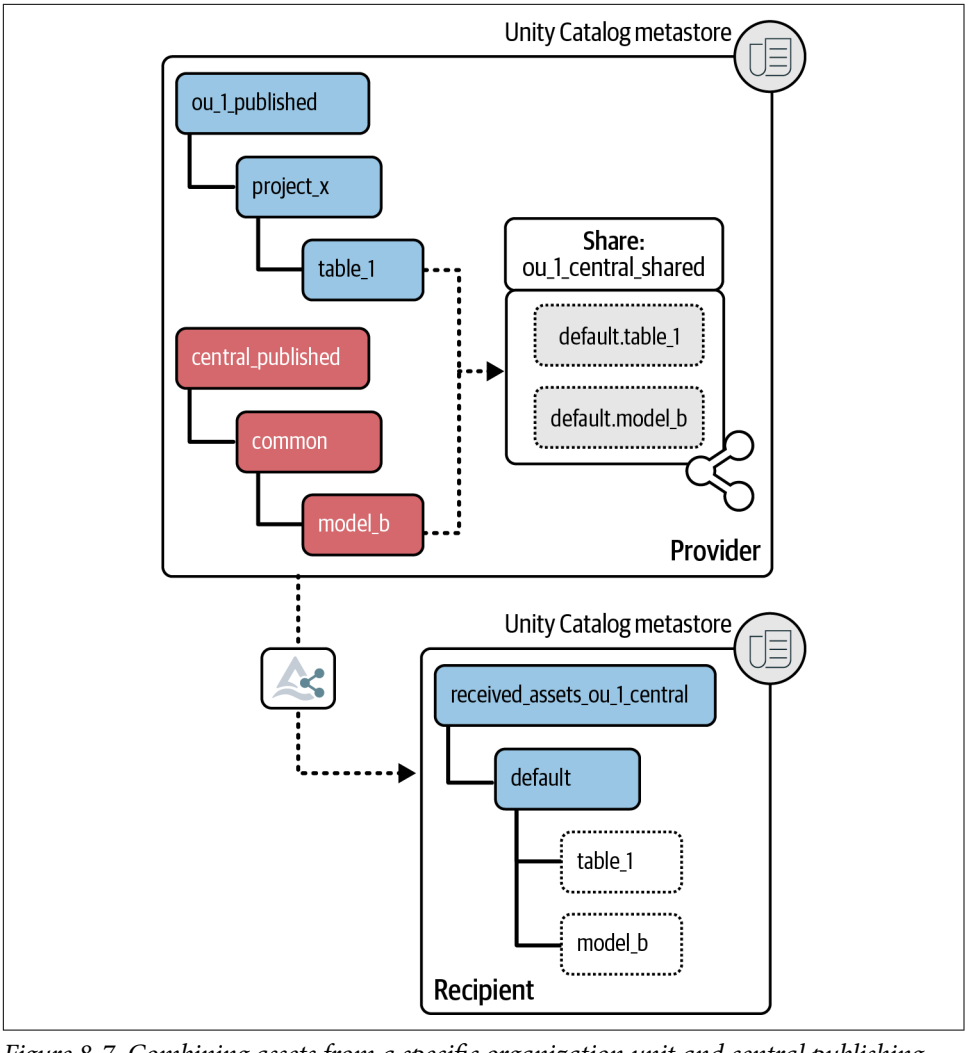


Figure 8-7. Combining assets from a specific organization unit and central publishing catalogs to share

On the recipient side, either the metastore admin or a principal with `CREATE CATALOG` and `USE PROVIDER` privileges can create a catalog for the `SHARE` and grant other principals access to it. You can grant only read permissions on these shared assets. As mentioned previously, use the recipient admin group for managing the `SHARE` objects. Metastore admins should not be doing Delta Sharing-related activities. Figure 8-8 shows this flow.

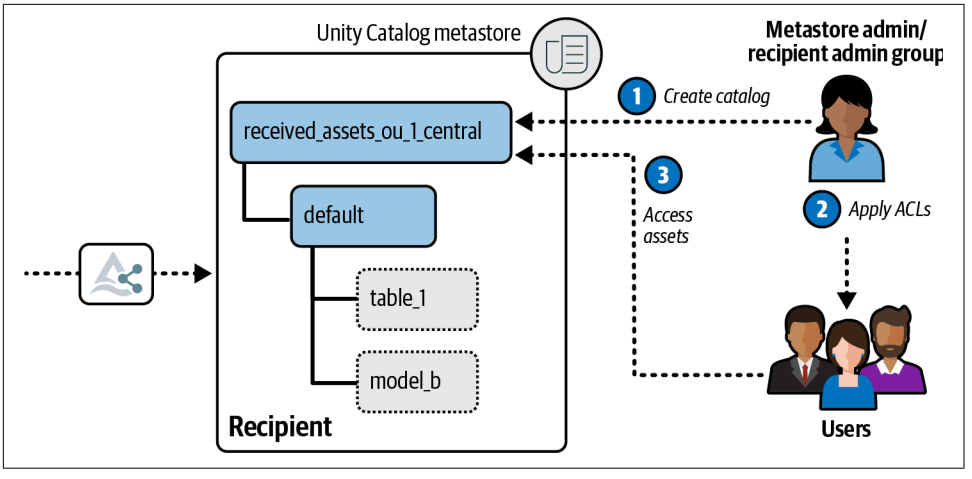


Figure 8-8. Principals with relevant privileges grant access to other users on the shared assets

Challenges

Now let's take a quick look at some important challenges of D2D sharing in the context of governance:

- As pointed out previously, you share the assets with another metastore, and not with specific users or groups. This means the access controls need to be applied independently on the recipient side, even if the recipient belongs to the same organization and uses the same Databricks account.
- Privileges on Delta Sharing securables, namely `SHARE`, `RECIPIENT`, and `PROVIDER`, are at the metastore level. This means you either get a particular privilege on all the objects of a particular type or none. For example, if a user has `USE SHARE` privilege, then that user can view all shares in that metastore.
- FGACs that are based on user identity are not supported with Delta Sharing. You cannot share tables with row filters and column masks, and dynamic views with the `is_member()` or `is_account_group_member()` functions.
- Certain metadata, such as tags and lineage, is not propagated with the `SHARE`. That means this metadata is disconnected and localized to a metastore.

In essence, we are sharing the assets with another entity, and once shared, the governance burden falls on the recipient side. We often hear customers and users ask how a provider can govern the shared assets. The intention is to implement governance with Unity Catalog that goes beyond the metastore boundaries. It is mainly organizations with a multicloud and multiregion Databricks setup that naturally want to have simpler cross-metastore governance, especially in terms of access controls. The main

challenge is that the provider cannot control access to shared assets on a granular level despite using a common identity provider, which is the source of truth for all identities, even across Databricks accounts. Figure 8-9 shows this setup.

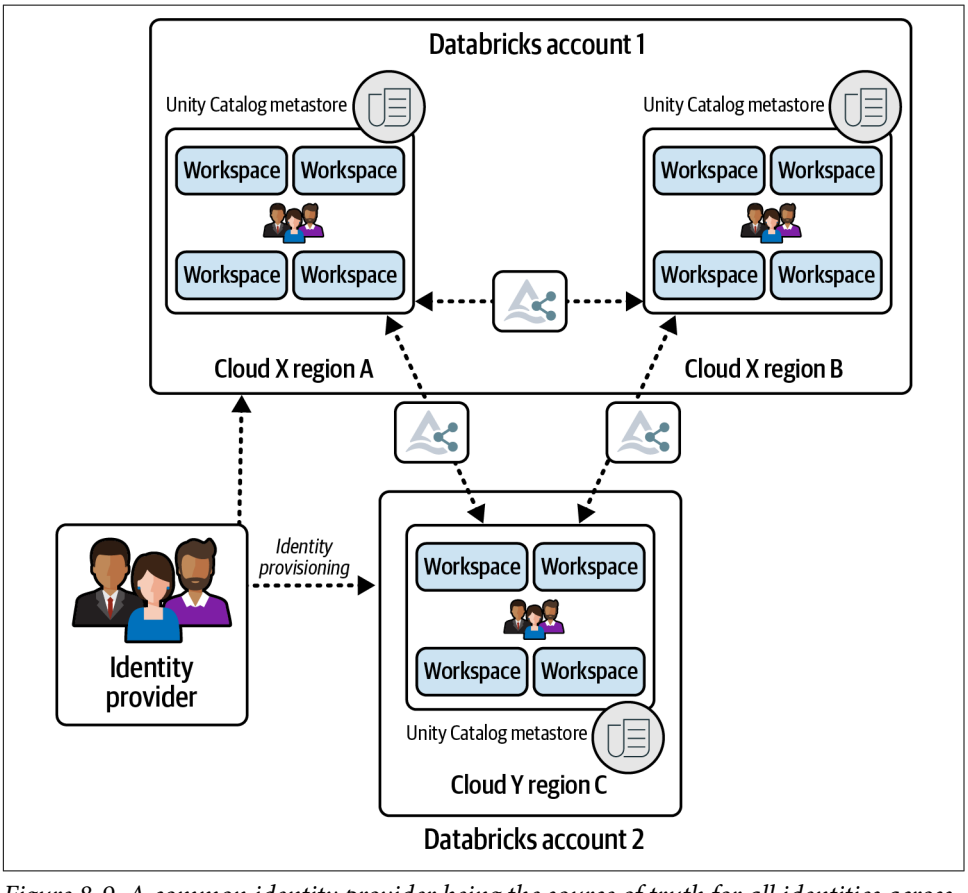


Figure 8-9. A common identity provider being the source of truth for all identities across multiple Databricks accounts

Although Delta Sharing is constantly improving, for the time being, we have to address some of these challenges with custom measures.

Propagating access controls across metastores

One way to address the challenge of cross-metastore access control is to propagate the principal-privilege mapping to the recipient within the `SHARE`. As a provider, if you already know who, as in which principal, should have access to the shared assets on the recipient side, you can store that information in a Delta table or a file within a volume and add it to the `SHARE`, shown in Figure 8-10.

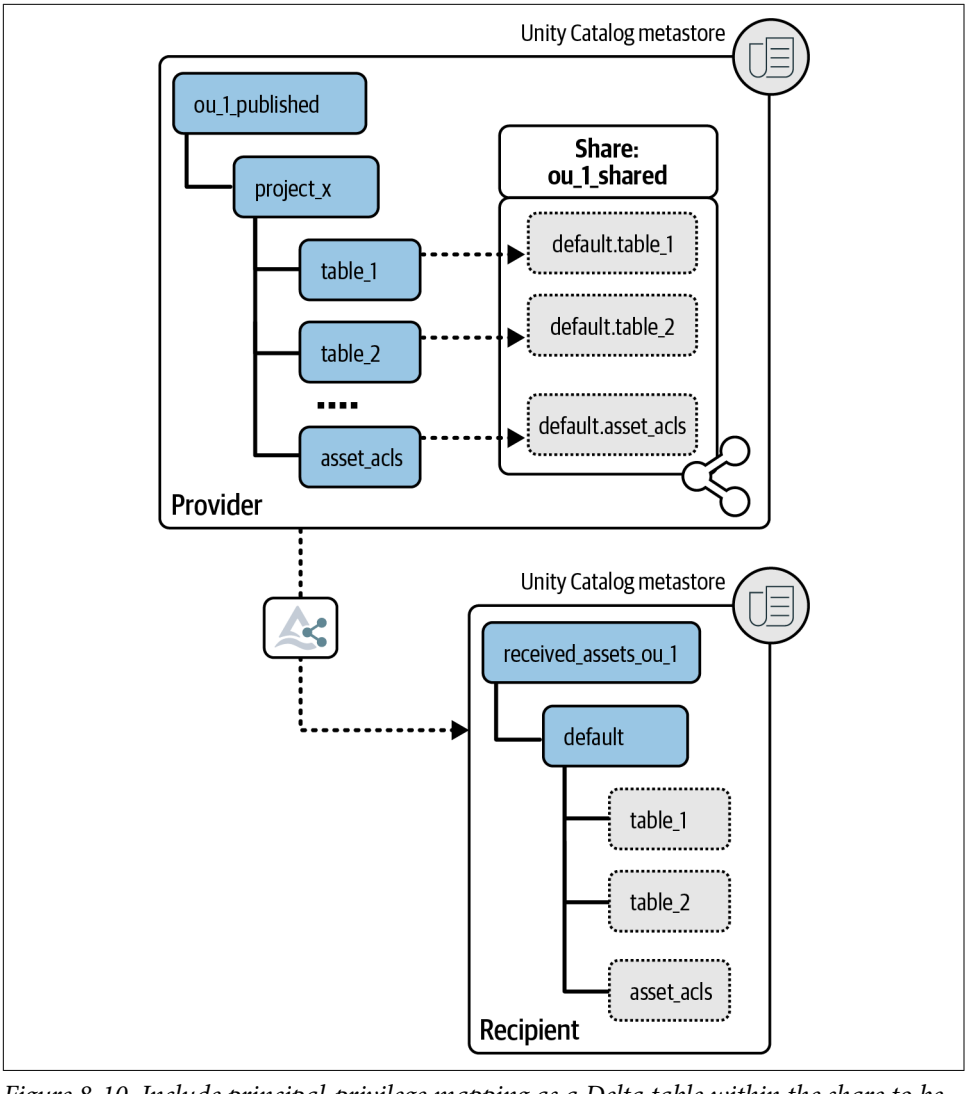


Figure 8-10. Include principal-privilege mapping as a Delta table within the share to be applied at the recipient

On the recipient side, the `ms_cloud_region_ds_recipient_admins` can then use this information to set proper access controls. Of course, this means that you have to trust the admins on the recipient side. A certain level of trust is necessary for an organization to function efficiently. Figure 8-11 illustrates this approach. You also need to consider how to manage updates to the access controls from the provider side. Whenever the `asset_acls` (from the figure) is updated, it needs to be fetched and applied. One way to do this is by running a scheduled job or setting up a

mechanism to detect changes to the table and trigger a job that applies the ACLs. The details of such an approach are out of scope for this book.

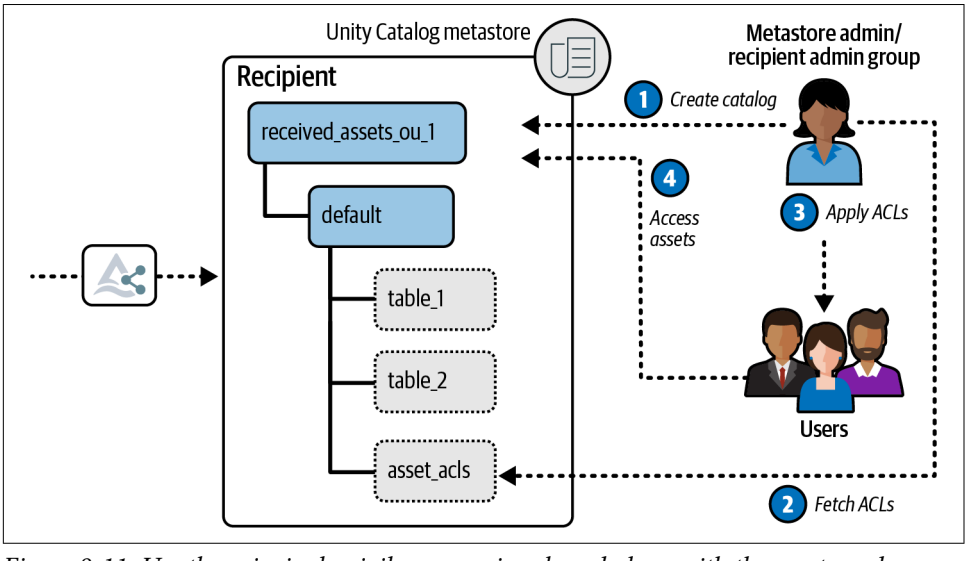


Figure 8-11. Use the principal-privilege mapping shared along with the assets and apply it

Depending on your requirements and organizational structure, there are other approaches that you can consider for managing access to shared assets, such as these:

- An organization-wide admin group–one group to manage all data sharing internally across regions and clouds.
- If you have a data mesh-like decentralized setup with multiple domains and governance is your top priority, then use a centralized hub-and-spoke model, where the hub can at least monitor/audit what gets shared.
- If you want to share assets with a different legal entity (i.e., a different account), it might be difficult to just rely on trust, and moreover, monitoring/auditing might not be possible. In such cases, carefully assess what assets you share.

Cross-metastore data discovery and lineage

You can search and discover assets that are cataloged in a metastore through a workspace associated with it. You cannot look for assets that are cataloged in another metastore. Since discovery tags and lineage are tied to Unity Catalog assets, these are also scoped to a metastore. This presents a challenge when you, as an organization, want an overview of all the assets cataloged across the metastores, or to fetch the end-to-end lineage of shared assets, and so on. This could be addressed with a custom solution. For example, you can build an application that connects one workspace per

metastore via JDBC (Java Database Connectivity)/ODBC (Open Database Connectivity) or REST APIs and collect the Unity Catalog asset metadata. Furthermore, you can build one using Databricks Apps, which provides a simple way to build secure data and AI apps within the Databricks Platform. Figure 8-12 shows this setup. Databricks Apps should be hosted in one of the workspaces. Such an app can help with not only discovering cross-metastore assets and concatenating the lineage, but also in setting up shares and applying tags.

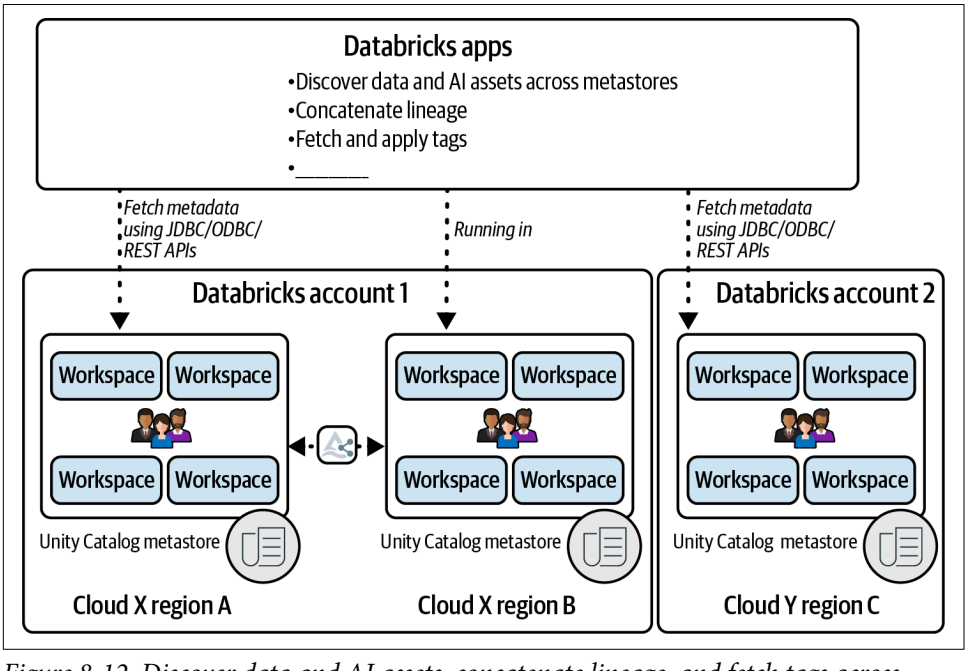


Figure 8-12. Discover data and AI assets, concatenate lineage, and fetch tags across metastores using Databricks Apps

FGACs across metastores

It's not possible to share tables with row- and column-level security and dynamic views that use the `is_member()` or `is_account_group_member()` functions across metastores. However, you can create a dynamic view for filtering rows and columns using the `CURRENT_RECIPIENT()` function to limit recipient access according to properties that you specify in the recipient definition (there's also an option to limit shares to partition-level divisions during creation). But since you can currently register only one recipient per metastore, there is no way to implement row and column security with D2D sharing. One approach to implementing this is to share the source tables and the dynamic view definitions in the form of files with SQL queries in a volume with the recipient. On the recipient side, the recipient admins can use this definition and create views. The

users should be granted access to only the views. Even in this approach (illustrated in Figure 8-13), you need to rely on and trust the recipient admins.

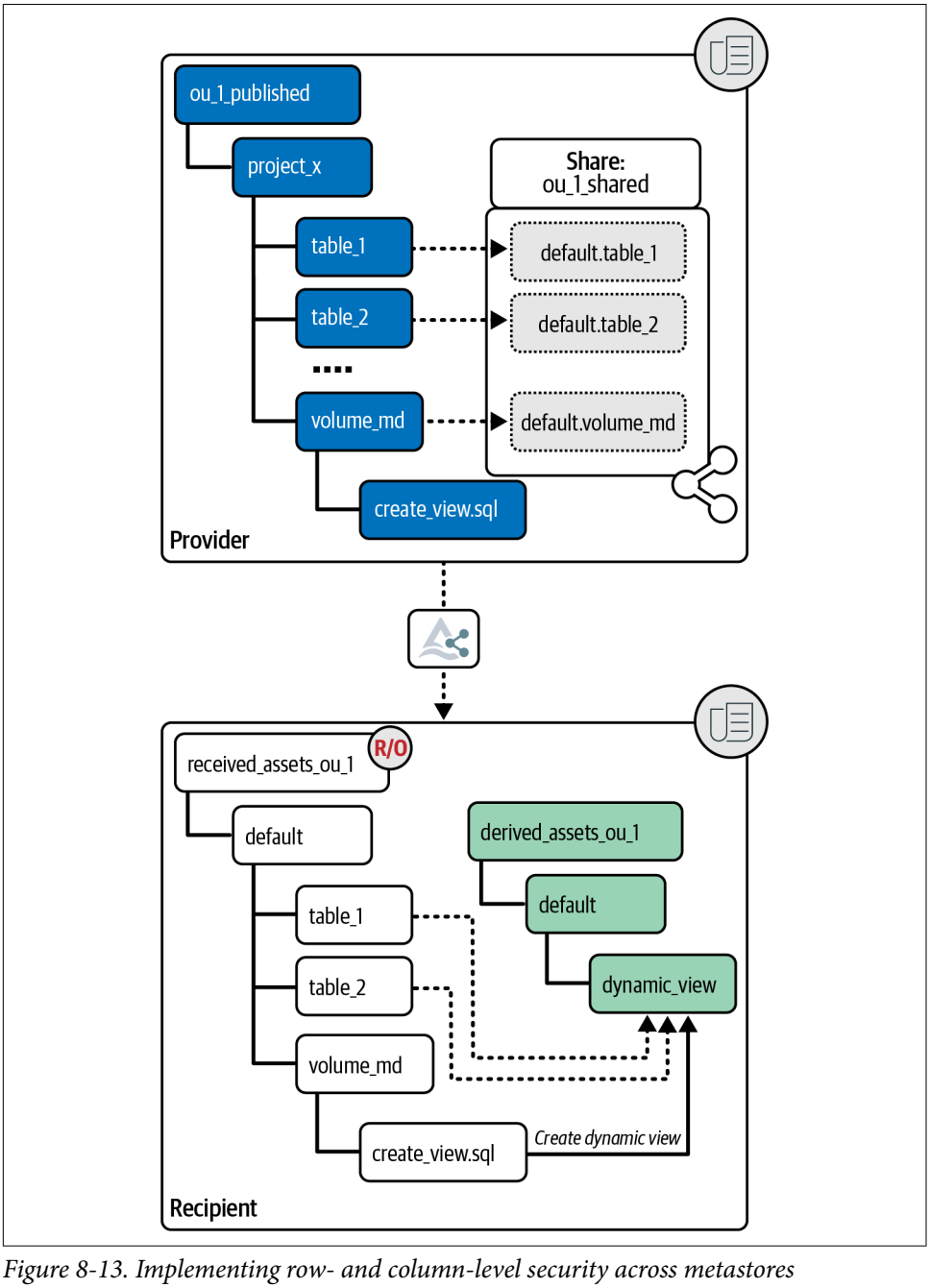


Figure 8-13. Implementing row- and column-level security across metastores

Cost and Performance Considerations

There are no added costs for using managed Delta Sharing on the Databricks Platform. In a D2D sharing setup, most costs are on the recipient side in terms of compute. However, you also need to consider the storage egress costs on the provider side, as we almost always deal with cross-region and cross-cloud sharing. Egress costs increase with the volume of data, hence, special care should be taken when designing the architecture and considering things like what the volume of data is, how it grows over time, how frequently it is accessed, and so on. As a provider, you might need to track the data access costs for cost attribution or chargeback. System tables and storage-specific access logs are helpful in such cases. In Chapter 7, we discussed system tables in detail.

The query performance for cross-metastore data access via D2D sharing depends on several factors such as data volume, optimization options used, and network path. For instance, a longer network path that includes routing the data through firewalls can negatively affect the latency.

You should consider the use cases and understand how the shared assets will be used. If it is sufficient to share aggregated data instead of source tables, that is better in terms of both costs and performance. It's also worth considering the type of assets that you share. For instance, it might be a better option to share materialized views instead of regular views for better query performance, as these are precomputed.

The target data architecture also plays a crucial role in determining costs and performance. On the recipient side, managed replicas of shared assets can reduce egress costs and improve latency.

Internal and External Sharing

In Chapter 5, we saw how Nexa Boutique implemented the harmonized data mesh and how it structured its catalog layout. Nexa was mainly focused on its organizational units that are co-located in a single geography. In other words, sharing data products across domains that are associated with a single Unity Catalog metastore is relatively simple. However, as Nexa grew and expanded its operations across the globe, naturally, the data estate was also spread across and no longer confined to a single geographic region.

Moreover, Nexa had acquired Tamarind Shoes, which remained a different legal entity even after the acquisition. From a technology perspective, Nexa was a heavy user of AWS and had almost all its Databricks workloads on AWS. Tamarind Shoes, on the other hand, was an Azure-centric company and was also a Databricks customer using Azure Databricks and had no intention of migrating to another cloud. Nexa was also embracing the idea of multicloud to avoid too much dependency on one cloud provider, similar to its peers and competitors.

Data Mesh with Delta Sharing

Apart from Nexa's deliberate decision to go multicloud, its growth and acquisition further fueled the transition to a multiregion and multicloud data platform, which was more complex to manage and govern. Naturally, the teams across the globe wanted to collaborate and leverage data and AI assets owned by others, and since Nexa had fully adopted Databricks and Unity Catalog, they were in the best position to leverage Delta Sharing to extend the data mesh concept across regions and clouds.

Obviously, there were a few challenges that the teams had to overcome, more or less the same challenges that we discussed earlier:

- Centralized data discovery was a challenge. Each domain made data available in the domain-specific published catalogs, as described in Chapter 5. However, these catalogs were visible only to users who were part of the same metastore, and users from other regions or clouds had no way to discover data and AI assets.
- The lineage was localized to a metastore. Nexa was quite happy with the Unity Catalog lineage feature; however, in use cases spanning regions or clouds, there was no easy way to stitch the lineage across.
- Some domains wanted to share tables with row filtering or column masking, but Delta Sharing did not support this.

Nexa solved the first two challenges by developing a Databricks Apps app that would provide a simple interface where users could log in and discover available data and AI assets and request access to them. There were two main access patterns:

- A user accessing the asset through a workspace within the same metastore context: upon approval, the user's principal would be simply added to the account group that had read access to that asset.
- Users accessing the asset through a workspace in the context of a different metastore: upon approval, the asset would be shared using D2D sharing.

In essence, the app was only for discovery and overview. Access to the data was always through a workspace. Developing an app on top not only allowed Nexa to address the discoverability and end-to-end lineage challenges, but it also paved the way to implement additional governance features such as a Business Glossary, Master Data Management, and so on that were not available with Unity Catalog.

Regarding the challenge of sharing secure assets such as tables and dynamic views with row filtering and column masking, Nexa decided not to share those assets across metastores. If a user needs to access those secure assets, then they have to connect to the workspace where it is available or use Lakehouse Federation.

External Sharing

Nexa, being an online retailer, was also sharing data with its suppliers and partners, mainly to forecast demand and optimize the supply chain. However, like many other retailers, they had a not-so-nice data-sharing setup, involving CSV exports, SFTP servers, and even emails. However, while they were rolling out D2D sharing for sharing data internally across domains, they realized that they could use the same approach with their partners and suppliers. But only a few of them had adopted Databricks, so D2D sharing could only be used with them.

Therefore, Nexa decided to go with D2O sharing for sharing data externally for all their suppliers and partners, just to be consistent in their setup and approach. For those who were reluctant to use this new approach, Nexa provided additional guidance and training and made it happen. Although this was an additional investment in terms of time and effort, it paid off really well as they now had a stable, reliable, and secure platform for sharing data. Figure 8-14 shows how Delta Sharing enables sharing across regions, clouds, and organizational boundaries.

Using D2O sharing provided two additional benefits:

- Nexa could offer OIDC Federation for secure authentication for those partners and suppliers who have their own identity provider.
- Nexa could filter out records based on the recipient. This meant Nexa could share the same table with multiple recipients, but each one would see only the relevant records.

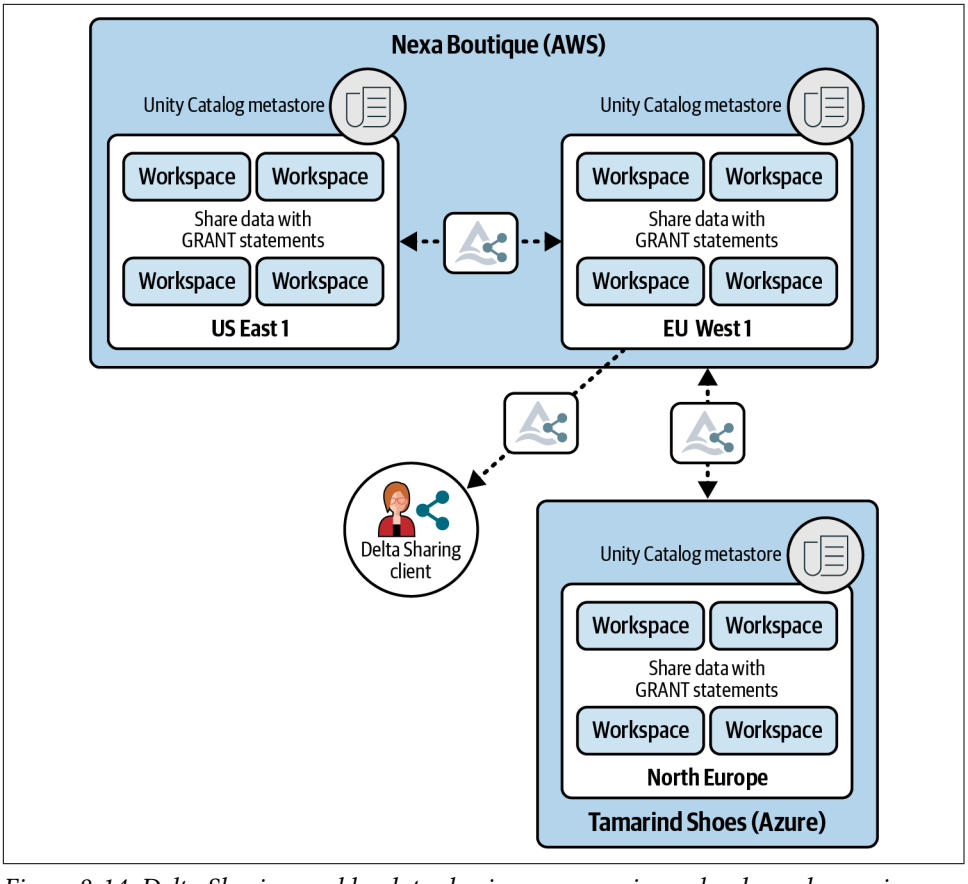


Figure 8-14. Delta Sharing enables data sharing across regions, clouds, and organizational boundaries

Databricks Marketplace and Clean Rooms

Databricks has leveraged Delta Sharing to provide additional product offerings, namely Databricks Marketplace and Clean Rooms. Databricks Marketplace allows you to publish your data and AI assets, which means your reach is broad. It's mainly useful for you as a provider if you wish to monetize or donate your assets. As a consumer, you'll find many useful datasets in the Marketplace offered either for free or for a fee. Access to the data and AI assets is through Delta Sharing. The data discoverability and usability of the Databricks Marketplace UI are business user-friendly and more convenient than the Catalog Explorer, which is the main UI for exploring and managing data and AI assets in Databricks.

You could also choose to share your assets with only a few recipients instead of publishing and making it available for everyone. You do this by creating a Private Exchange, where you can specify the list of recipients in the form of Unity Catalog metastore IDs. The advantage of using the Databricks Marketplace instead of configuring Delta Sharing separately for each recipient is that you can obviously minimize the effort in setting it up and maintaining it but also leverage the business user-friendly experience for the recipients.

If your goal is to collaborate with the external parties who are also on Databricks, rather than just share the assets, consider using Databricks Clean Rooms. This option is suitable for the collaborative use case where you need to combine sensitive data to derive business insights. The concept of a clean room is to provide a zero-trust environment for multiple collaborators to bring their sensitive data and combine it with the data from other collaborators. This is achieved using an ephemeral Databricks environment where collaborators share their data, run code using Notebooks, which must be approved by all parties, and fetch results. The collaborators see only the metadata, such as column names and types, but do not see data shared by others. As you might have already guessed, the sharing of data and Notebooks is through Delta Sharing.

Summary

In this chapter, we discussed the challenges of data sharing and mainly focused on how Delta Sharing acts as a scalable, reliable, and secure framework for organizations to share data and AI assets, both internally and externally. We elaborated on the governance aspects of data sharing based on D2D sharing, which enables data mesh-like architecture within organizations, even with a multiregion and multicloud setup. Finally, we also touched upon the topics of Databricks Marketplace and Clean Rooms. In Chapter 9, we explore how to access data stored in Databricks from external engines and platforms, focusing mainly on the interoperability aspects of the Databricks Platform.

CHAPTER 9
Open Access

Given enough eyeballs, all bugs are shallow.
—Linus's Law, Eric S. Raymond

The software world divides into two main camps: open source and proprietary. You see this divide in action with GNU/Linux (Linux), a family of open and collaborative operating systems, and Windows, another operating system, controlled by Microsoft. This distinction shapes how you develop, use, and experience technology, affecting its flexibility, security, and innovation. Most Linux distributions come without a license cost and are highly customizable, with accessible source code. Windows, on the other hand, is licensed, and customizations are minimal. By understanding this difference, you can make informed decisions about your software. When Kiran joined Tata Consultancy Services as a Java developer in 2012, he was handed a book, *Hadoop: The Definitive Guide* (O'Reilly), by Tom White. This was when *big data* and *Hadoop* were among the trendiest words in the industry. He never imagined this introduction would lead to a long-term career in data and open source technologies.

Kiran's early experience with open source technologies, which he discovered through Hadoop, showed him how they benefit end users. Then came Apache Spark, initially built by Matei Zaharia, which changed how enterprises use and adopt open source technologies. Years later, as an architect working on data governance, specifically with Unity Catalog, Kiran encountered a common question from users: isn't Unity Catalog closed and proprietary to Databricks? Users also often ask: how do I connect my external engines to read data managed by Unity Catalog?

Databricks is recognized in the industry for developing and making its products open source to benefit a wider community of users. Among its notable contributions:

Apache Spark
: The de facto data processing engine for data engineering, ML, and data science workloads.

MLflow
: For managing the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

Delta Lake
: Enables building a lakehouse architecture on top of storage systems such as AWS S3, ADLS, GCS, and HDFS.

Delta Sharing
: The industry's first open protocol for secure data sharing; makes sharing data with other organizations simple.

Unity Catalog
: One of its most recent contributions.

In 2024, Databricks announced that Unity Catalog would be open source at the Data and AI Summit to provide an open foundation for governing data and AI assets. With Unity Catalog being open source, the open REST API enables support for Apache Iceberg Catalog API and the Unity REST API. The acquisition of Tabular, a data management company built by the creators of Apache Iceberg, laid a strong foundation to expand the Unity Catalog openness. This led to native support for the Iceberg format being developed in Unity Catalog tables and unified with the Delta format in two key ways:

- You can now read and write managed Iceberg tables using:
  - –Any Databricks compute engine
  - –External Iceberg engines via the Unity Catalog Iceberg REST catalog API
- Via the LF, you can access and govern Iceberg tables managed by external catalogs, including:
  - –AWS Glue
  - –HMS
  - –Snowflake Horizon Catalog

This expansion enables a broader ecosystem of tools and engines to connect to and consume data irrespective of file format, while maintaining data governance.

In this chapter, we discuss how you can benefit from Unity Catalog being open source and better understand how to use Unity Catalog to govern your data and AI assets. Before exploring open source and open access to data, it's essential to understand the need. Unity Catalog supports three types of tables: foreign, managed, and external.

Each has distinct features that set it apart. However, it fell short in certain aspects of interoperability. The open source Unity Catalog addresses these limitations. You likely want to understand the trade-offs between managed and external tables for your use cases, as discussed in Chapter 5. You can deploy the open source version of Unity Catalog on your infrastructure, unlike the Databricks Unity Catalog, a fully hosted service in the Databricks control plane. This version complements your existing tools, including Delta Lake tables, UniForm files in volumes, functions, and models. We discuss the reasons for making Unity Catalog open source and its impact on you as a Databricks user. Unity Catalogs Open REST API specification and credential vending feature allow external engines to access Unity Catalog assets. Unity Catalog's support for the Iceberg REST catalog API through the UniForm data compatibility also enables seamless integration with external engines that read Iceberg file formats.

Managed Versus External Table

You might find the term *managed table* confusing. Many Databricks users question its implications on data control. The concern stems from a misconception that Databricks copies and manages data in its cloud storage, limiting external access. Nexa's data architects initially shared this concern about managed tables when adopting Unity Catalog. They transitioned from HMS, where external tables were the norm and recommendation. As a result, they applied this existing knowledge when transitioning to Unity Catalog. Regarding interoperability, Nexa's data architects were right to rely on external tables because Databricks recommended external tables in the initial days of Unity Catalog for scenarios where multiple systems read and write the same data.

Contrary to popular belief, with managed tables in Unity Catalog, your data remains in your cloud storage account, which you own and control. With external tables, you specify the storage location path where the data gets written, while for managed tables, the path is determined by Unity Catalog. Unity Catalog follows a specific hierarchy when choosing the data storage location for a managed table. Here's how it works:

- The data is stored at the schema level if a storage location is defined.
- If not, the system uses the catalog-level storage location.
- If neither is defined, the system uses the metastore level storage (no longer recommended).

Refer to Chapter 5 for more details on managed tables.

Why Use External Tables?

Nexa used external tables to enable multiple tool integrations. For example, the warehouse operations team created a Delta table in an S3 storage location using Databricks on AWS. The demand planning team consumed this data using an OSS Apache Spark application. Figure 9-1 illustrates the external table path-based access at Nexa.

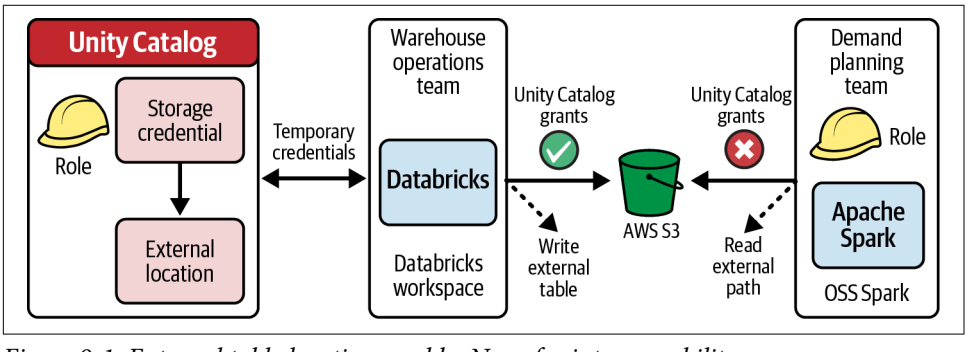


Figure 9-1. External table location used by Nexa for interoperability

You may also have similar patterns used in your organization. For example:

- Multiple teams accessing the same data from a shared path.
- Various tools and data processing engines are involved.

Using external tables can facilitate this setup because external tables are merely an abstraction layer on the physical file that is manipulated by any data processing engine. Databricks becomes just another compute layer that can facilitate further processing of the files registered as a table in Unity Catalog. However, it can also compromise data governance. External tables bypass Unity Catalog access controls because they could allow direct data access via the cloud storage path using a credential not managed by Unity Catalog. Direct path-based access bypasses Unity Catalog access controls, requiring you to manage permissions elsewhere and creating a more complex governance setup. When you access data via a path instead of a Unity Catalog-managed table, the Unity Catalog commit service is unaware of these writes and reads, potentially causing collisions with concurrent writes to the same data. Accessing data via paths bypasses Unity Catalog's FGACs, like RLS and column masking, that Databricks' compute engine enforces, so you risk compromising your data governance.

Accessing data directly from storage using external engines, outside the control of Unity Catalog, has several drawbacks:

- You lose any control and security that Unity Catalog provides and must then implement controls at the storage layer.
- Any lineage and audit logs are unavailable for external engine access.
- You must also reconfigure access controls on the external engine.
- The external engine may not support advanced security features like FGACs.
- Renaming tables breaks clients because the path changes. (The Hadoop-led shortsighted approach of handling datasets as paths in a filesystem where the table name is part of the path is flawed at best.)
- Unreliable `UNDROP` operations, as a new table can be created in the same location.

These drawbacks can result in inconsistent security across tools and engines. Managing multiple security frameworks becomes complex and time-consuming. You should consider the potential impact on your organization's data security and compliance.

External tables are suitable for use with legacy data from Hadoop and on-premises systems. They support various file formats, such as Avro, ORC, JSON, CSV, Text, Delta, and Parquet.

When migrating from legacy systems, consider converting data in an open-file format like Delta or Iceberg. Delta is the default and currently best-performing format for Databricks tables. You should evaluate your file formats and consider converting to an open lakehouse format for better performance and ACID compliance.

Data Independence

In 1970, Edgar Frank Codd, a British computer scientist working for IBM, published a seminal paper titled *A Relational Model of Data for Large Shared Data Banks*. This paper discusses the relational model for organizing large amounts of data in a shared database. Codd's paper has significantly impacted the development of database systems that have revolutionized how we store, manage, and query large amounts of data today. One key idea that Codd proposed was about data independence: "Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed." By this, Codd meant that how you access data should not be dictated by how the data is stored and organized. You want your applications to remain unaffected by changes in data formats or how your data is organized.

Further, Codd mentions three kinds of data dependencies that can occur:

Ordering dependence
: Data storage and retrieval are tied to a specific ordering, making it difficult to change the ordering without affecting applications.

Indexing dependence
: Applications rely on specific indexes or access structures to access data, making changes to or removing indexes problematic.

Access path dependence
: Data retrieval is tied to a specific sequence of steps or access paths, making changes to the storage structure or data organization challenging.

Accessing an external table via its path makes your application logic dependent on that specific path, creating a fragile setup prone to errors when the path changes. Using advanced Delta features like deletion vectors requires upgrading the table protocol specification (that is, setting `minReaderVersion` to 3 and `minWriterVersion` to 7), which can cause compatibility issues with older Delta clients that don't support the new protocol versions. Delta Lake's *table features* allow different clients to work with the Delta protocol even if they don't implement every feature. However, many clients still rely on older protocols, making them potentially incompatible. Using a fixed path to serve all clients can be limiting due to potential compatibility issues with different clients and Delta protocol versions, making them data-dependent. It's better to avoid this limitation and keep your applications flexible.

Definition: Table Features

Delta Lake table features are a feature-centric approach that replaces traditional version-based management, making it easier to work with Delta Lake tables and clients. It enables Delta clients, users, and developers to focus on the required features rather than building support for all features in a specific Delta protocol version.

Managed Tables for the Win

You achieve data independence by eliminating path-based access and switching to table name-based access. Managed tables in Unity Catalog accomplish this by removing predefined paths and relying on the catalog to deliver data to clients, thus achieving data independence. The data stored in open formats like Delta or Iceberg provides interoperability using UniForm. You interact with the data using the table name and its open REST API, giving Unity Catalog the visibility it needs to manage and organize the data for you, for both writes and reads. With Unity Catalog, Databricks introduced features that make managed tables a more attractive option.

One key advantage is that Unity Catalog is aware of all transactions on managed tables, including reads and writes. Since Unity Catalog manages the storage location, it can also optimize the data layout automatically using predictive optimization.

Definition: Predictive Optimization

Predictive optimization is a feature that analyzes your data and query patterns to optimize the data layout. It uses AI models to predict the most effective optimizations, evaluating potential benefits and costs to determine the best approach. Essentially, predictive optimization is a data layout optimization technique that automates the process of improving data performance and reducing costs.

Along with predictive optimization, managed tables in Unity Catalog benefit from automatic statistics collection. Unity Catalog managed tables automatically collect statistics such as minimum and maximum values for columns and cardinality of columns. This information helps optimize queries by skipping irrelevant columns and determining the best join strategies. Predictive optimization collects these statistics automatically and predicts which columns to prioritize. If you're coming from a Hive background, you're likely familiar with Hive-style data partitioning, which can be limiting. Partitions are data-dependent, so changes to the folder structure can cause queries to fail or return inaccurate results. Moreover, partitions affect your query patterns, reducing flexibility in organizing data and making it difficult to change if your query patterns shift.

Liquid clustering is a feature that enables more agile file organization by clustering data based on frequently used keys. It's available for external and managed tables, but the configuration differs. You need to specify the clustering keys manually for external tables, but changing the keys is flexible based on changing query patterns. In contrast, managed tables in Unity Catalog offer a more streamlined approach, where you can simply designate the clustering as *auto*. This is where Unity Catalog's visibility to your query patterns becomes helpful. Unity Catalog uses heuristic-based rules and models to analyze your query patterns and determine the best way to reorganize your data, which is done automatically via predictive optimization. Using managed tables with automatic liquid clustering allows you to enjoy improved performance and flexibility, without being tied to a specific path.

External tables need to specify the column name for the liquid clustering key using the `CLUSTER BY` command, as in this example:

```
CREATE TABLE CATALOG.SCHEMA.table_name (
 col0 INT
 ,col1 string
 ) CLUSTER BY (col0) LOCATION 'table_location';
```

Managed tables can specify `AUTO` rather than specific columns, like this:

```
CREATE TABLE CATALOG.SCHEMA.table_name (
 col0 INT
 ,col1 string
 ) CLUSTER BY AUTO;
```

Unity Catalog's awareness of table transactions enables it to cache Delta logs in memory, resulting in faster query performance. When you query a table, the Delta log is read first from cloud storage to determine which files to read to satisfy the query. The latency of cloud storage APIs slows this process down when users frequently query multiple tables. With managed tables, Unity Catalog is aware of changes to the Delta log and can cache it in memory, significantly speeding up query performance. In contrast, accessing data via a path, as with external tables, prevents Unity Catalog from detecting modifications, making it impossible to cache the Delta log.

As a data platform team, after investing countless hours in optimizing tables for performance and cost, you can now reap the benefits of automated data layout optimizations, file management, and concurrency control. Managed tables in Databricks provide features like automated liquid clustering, column mapping, and deletion vectors, freeing you from manual optimization tasks. By leveraging Databricks' AI-enhanced capabilities, you can enjoy high-performing tables right from the start, without the need for constant monitoring and tuning.

Initially, managed tables had their own set of limitations, posing a problem for users who wanted to access data using external engines that don't integrate with Unity Catalog. You could locate the path for the table and access it via an external engine. There are multiple ways to retrieve the table location:

- Use the `DESCRIBE DETAIL catalog.schema.table_name` command
- Invoke the Unity Catalog REST API: `/api/2.1/unity-catalog/tables/{table_name}`
- Query the system tables in Unity Catalog:

```
SELECT table_catalog
 ,table_schema
 ,table_name
 ,storage_path
FROM system.information_schema.tables
WHERE table_type = 'MANAGED'
 AND table_schema = 'schema'
 AND table_name = 'table_name';
```

Dropping, re-creating, or renaming a managed table changes its storage path, which contains a unique GUID (global unique identifier) style folder. This breaks path-based access to the data from external engines.

A sample path of a storage location registered at the catalog level would look like this:

```
s3://storage/<uniqueid>/tables/<uniqueid>
```

You might wonder why managed tables use a unique identifier in their path. Like an external table, a predictable path would be more convenient for pointing external engines to the data. Let us try to clarify this. When a user drops a managed table, Unity Catalog also deletes the underlying data. This raises concerns about accidental deletes in production environments. To mitigate this risk, Databricks provides an `UNDROP` command. This command can recover a dropped table within a fixed retention period.

Here's how it works: dropping a managed table doesn't immediately delete its data, instead, it remains stored under its unique GUID-based location. This design prevents data loss and enables recovery through the `UNDROP` command, safeguarding against accidental deletions. The unique GUID also boosts security by preventing unauthorized path-based access.

Open access to managed tables remained a challenge in the initial days of Unity Catalog until the open API and open source Unity Catalog were released.

You can now convert a Unity Catalog external table to a managed table from DBR Version17.0 and above with a simple SQL. Expect some minimal downtime during the conversion, which varies between approximately 1 to 5 minutes based on the volume being migrated:

```
ALTER TABLE catalog.schema.uc_external_table SET MANAGED;
```

Open Source Unity Catalog

Previous discussions have focused on the workings and architecture of Databricks Unity Catalog as a fully managed service. Databricks Unity Catalog provides comprehensive data governance for the assets it manages. Additionally, it enforces access controls when data is accessed through Databricks compute engines, ensuring secure data consumption. This approach suited many users who relied entirely on Databricks compute for their lakehouse architecture. However, a large group of users and the architects at Nexa were concerned about the limited interoperability of Unity Catalog with data processing engines outside of its ecosystem. As a large organization with diverse teams utilizing various tools and applications to process and consume data, Nexa required an open and governed approach to data access. It needed to balance the need for unrestricted data access with the robust governance controls that

Unity Catalog provides. This limitation made users perceive that Unity Catalog was a closed system that was inaccessible to tools and applications outside of the Databricks ecosystem.

Nexa's CDP team had invested heavily in an on-premises Hadoop platform for their enterprise data lake. Regulatory requirements, compliance policies, and legacy applications prevented a complete migration to the cloud. As a result, the team continued to rely on various other tools, including OSS Spark across multiple applications, and maintained a trivial portion of their data assets on premises. Nexa's technology strategy emphasizes the adoption of open source technologies and requires any tools with a heavy footprint used within its platforms to be OSS-compatible or have open APIs. Having an OSS version ensures you can maintain business continuity if a software vendor fails to support their software. At Nexa, a large ecosystem of tools and applications relies on data interoperability, which means cross-system accessibility is just as crucial as good data governance.

Databricks making Unity Catalog open source created two new opportunities:

- You can now deploy and host your OSS Unity Catalog instance on your infrastructure. The current implementation allows data access via credential vending through the *TemporaryCredentialsApi*. Future releases will introduce RBAC and ABAC. This lets you govern data and AI assets from your open source tools and integrate them with your ecosystem of data processing engines. For example, you can use OSS Spark or Trino engines for data processing and integrate with your OSS Unity Catalog instance to manage data access. This setup allows you to register models with OSS MLflow and nontabular datasets using volumes. OSS Unity Catalog integrates with AI and LLM tools like OpenAI, Anthropic, LlamaIndex, LangChain, and so on, enabling you to manage and govern the data used in your AI applications.
- You get OpenAPI spec-based access to Databricks-managed Unity Catalog and the Iceberg REST catalog support. With credential vending, the open API lets you integrate with many leading compute engines in the data and AI ecosystem to read data cataloged by Unity Catalog. This allows open access to data managed by Databricks Unity Catalog from any data processing engine of your choice, irrespective of the data format.

With OSS Unity Catalog, Nexa can access and manage data using various compute engines outside of Databricks, eliminating concerns about data being locked into a specific platform.

The OSS Unity Catalog comes with an OSS server application available on GitHub that you can use to host your Unity Catalog server. It also provides the OpenAPI Spec for managed tables, external tables, volumes, functions, and models. Credential vending is the backbone of its open API that helps external engines interact with

Unity Catalog tables securely without compromising data governance. We discuss that in detail in upcoming "External Access". Support for Delta, Iceberg, CSV, JSON, and Parquet file formats, and both the Unity REST APIs and Iceberg REST APIs, allows you to migrate to any other catalog supporting these file formats, alleviating vendor lock-in concerns.

You can deploy a self-hosted data governance platform using OSS Unity Catalog, as illustrated in Figure 9-2.

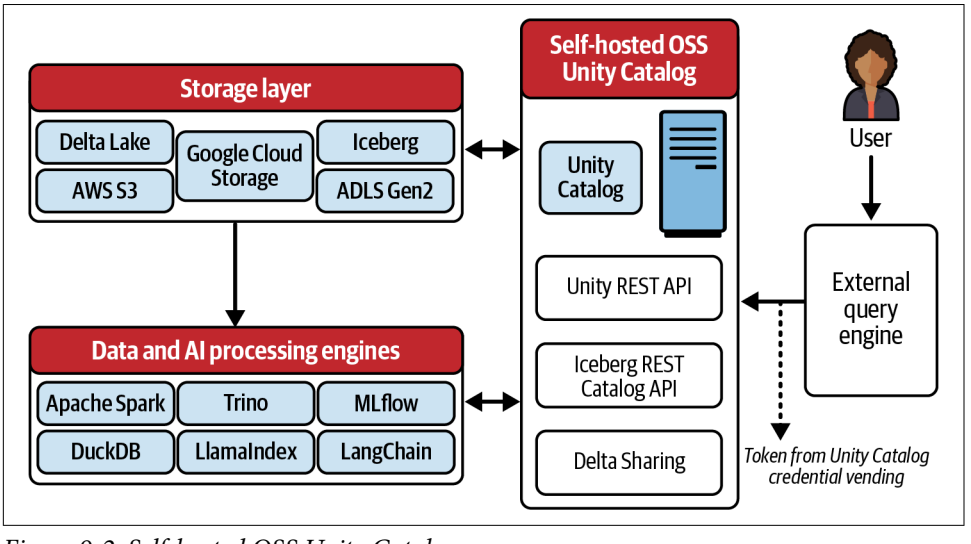


Figure 9-2. Self-hosted OSS Unity Catalog

External Access

Organizations rely on open source technologies because they help them reduce dependence on a software vendor. However, not every OSS project comes as a fully integrated solution that requires low maintenance. A pure OSS implementation often requires a large infrastructure team that supports the platform, but not everyone has the time and resources to keep their infrastructure up and running. When you deploy an application from a software vendor, you must also ensure the data is in an open format and accessible by any application, including the OSS counterparts if available. The same is applicable for Databricks and Unity Catalog. Until the OSS Unity Catalog was made available, accessing Databricks data using an external data processing engine was difficult. However, the introduction of Unity REST API and Iceberg REST catalog API support has addressed these issues, allowing users to access managed and external tables from external engines like Apache Trino or Apache Spark, making the ecosystem more flexible and open.

Open access also enables access to data, whether in Delta or Iceberg format. Chapter 2 explored UniForm, a feature that simplifies the decision-making process around file formats by generating metadata for multiple formats. The debate between Iceberg and Delta is common, with users often having a strong preference for one over the other. UniForm eliminates this dilemma by allowing Unity Catalog to present tables in both formats. Iceberg and Delta store data in Parquet files but differ in metadata management. Iceberg relies on a catalog to track its tables, whereas Delta can operate without one. UniForm bridges this gap by generating Iceberg metadata, and Unity Catalog serves as the catalog, implementing the Iceberg Rest Catalog. This allows clients to read tables in either format, promoting interoperability across various systems and tools.

Unity and Iceberg REST Catalog

Before Databricks enabled the open API specification, achieving interoperability between Unity Catalog tables and external engines required external tables with path-based access. However, this approach had significant drawbacks, as it bypassed Unity Catalog's access controls and forced users to implement access controls at the filesystem level instead. The open API changed this by allowing external Delta and Iceberg readers to access Unity Catalog-managed data directly from cloud storage using temporary credentials issued by Unity Catalog. The open REST API elevated the value proposition and openness of Unity Catalog and Databricks by enabling integration with multiple engines.

While the Unity REST API enabled Delta readers, multiple engines needed Iceberg compatibility. Unity Catalog also supports the Iceberg REST catalog specification to serve applications that read only Iceberg tables. With the release of Iceberg Version 3, key features like deletion vectors, the variant data type, row lineage, and geospatial data types were made available, which share identical implementations in Delta Lake.

Let's revisit the interoperability requirement at Nexa. An OSS Spark engine used by the demand planning team reads the Delta tables written by the warehouse operations team in Databricks. This time, we'll look at enabling this external engine to access a managed table and read the same table as an Iceberg table and a Delta table.

First, the warehouse operations team uses an `ALTER TABLE` statement to enable Iceberg read compatibility for the Delta table with the UniForm compatibility:

```
ALTER TABLE
 catalog.schema.table_name
SET
 TBLPROPERTIES (
   'delta.columnMapping.mode' = 'name',
   'delta.enableIcebergCompatV2' = 'true',
```

```
    'delta.universalFormat.enabledFormats' = 'iceberg'
)
```

Once enabled, the Iceberg metadata is generated alongside the Delta log asynchronously after the Delta commit. The warehouse operations team's data owner grants the user read access to the table in Databricks Unity Catalog along with use schema and use catalog privileges. You must meet certain prerequisites to enable an external engine to read data from Unity Catalog using the open API:

- A metastore admin must enable *external data access* for the Unity Catalog metastore.
- The principal accessing the table must be granted `EXTERNAL USE SCHEMA` privilege on the table's schema.

An OSS Spark engine that the demand planning team uses can connect to and read the Delta table by passing the additional Unity REST API configurations:

`"spark.sql.extensions": "io.delta.sql.DeltaSparkSessionExtension"` # Registers Delta Lake-specific SQL extensions with Spark.

`"spark.sql.catalog.spark_catalog": "io.unitycatalog.spark.UCSingleCatalog"` # Sets the implementation class for the default Spark catalog (`spark_catalog`) to Unity Catalog's single-catalog connector.

`"spark.sql.catalog.<uc-catalog-name>": "io.unitycatalog.spark.UCSingleCatalog"` # Registers a named catalog (replace `<uc-catalog-name>` with your actual catalog name) using Unity Catalog's connector.

`"spark.sql.catalog.<uc-catalog-name>.uri": "<workspace-url>/api/2.1/unity-catalog"` # Specifies the REST endpoint for the Unity Catalog service.

`"spark.sql.catalog.<uc-catalog-name>.token":"<token>"` # Provides an authentication token for accessing Unity Catalog.

`"spark.sql.defaultCatalog":"<uc-catalog-name>"` # Sets the default catalog for Spark SQL operations.

The Delta kernel fetches data and metadata using credentials from Unity Catalog. You don't need to provide cloud storage credentials to the Spark engine because Unity Catalog handles credential vending and acts as a commit coordinator, ensuring atomic operations for Delta tables.

The Delta kernel works with Unity Catalog to:

- Manage table discovery
- Handle data access
- Coordinate commits

To use Apache Spark and Delta Lake with Unity Catalog, you need:

- Apache Spark Version 3.5.3 or later
- Delta Lake Version 3.2.1 or later

Similarly, the OSS Spark engine can read the same Delta table as an Iceberg table by passing additional Iceberg REST API configurations and installing the dependent libraries:

`"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"` # Registers Iceberg-specific SQL extensions with Spark.

`"spark.sql.catalog.<spark-catalog-name>": "org.apache.iceberg.spark.SparkCatalog"` # Sets up a named catalog in Spark using the Iceberg Spark Catalog implementation.

`"spark.sql.catalog.<spark-catalog-name>.type": "rest"` # Specifies the backend type for the Iceberg catalog.

`"spark.sql.catalog.<spark-catalog-name>.uri": "<workspace-url>/api/2.1/unity-catalog/iceberg"` # Provides the REST endpoint for the Iceberg catalog.

`"spark.sql.catalog.<spark-catalog-name>.token":"<token>"` # Supplies an authentication token for accessing the Iceberg catalog.

`"spark.sql.catalog.<spark-catalog-name>.warehouse":"<uc-catalog-name>"` # Sets the default warehouse or namespace for the catalog

Here you can notice the difference in the `spark.sql.catalog.<uc-catalog-name>.uri` that passes the open REST API endpoint to the engine. The Delta version uses the `/api/2.1/unity-catalog` corresponding to the Delta format REST API. In contrast, the Iceberg reader uses the `/api/2.1/unity-catalog/iceberg` endpoint, which corresponds to the Iceberg format REST API. The connectivity via open APIs is illustrated in Figure 9-3.

You no longer have to make multiple copies of your data and generate different file formats so other teams can read it using their preferred data processing engine.

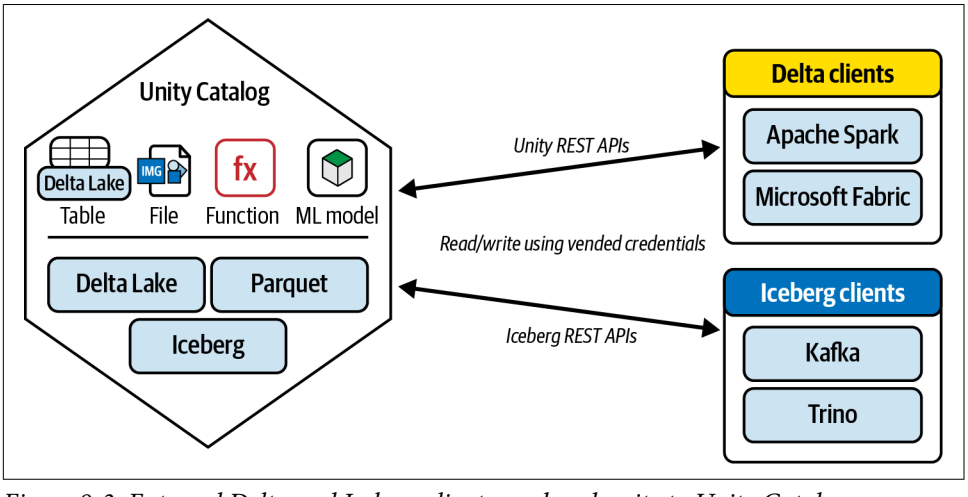


Figure 9-3. External Delta and Iceberg clients read and write to Unity Catalog

Credential Vending

Imagine being charged twice for compute, for the same data once for Databricks and again for the external engine you use to query it. It's a frustrating experience, similar to being taxed twice on your income. Consider the UK tax system, where you're taxed on your earnings through income tax, and then again on your spending through value-added tax (VAT) on the already-taxed income. Credential vending in Unity Catalog's external access via open APIs eliminates this double charging by generating a downscoped short-lived credential with a TTL (time to live) that allows direct access to the storage location, making it a more cost-effective experience. The generated token contains both the credential and the storage location, allowing direct access to the data in your cloud storage location. The credential vending process is illustrated in Figure 9-4.

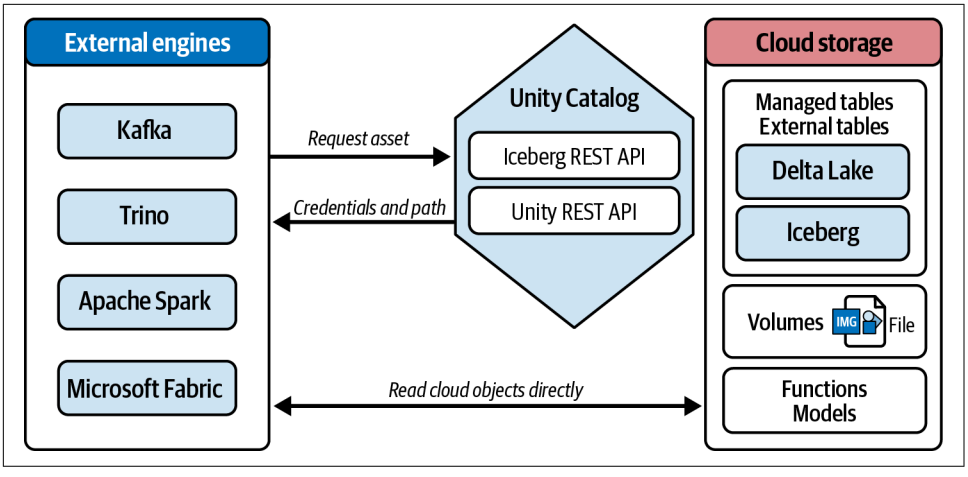


Figure 9-4. Credential vending in Unity Catalog for external access

Credential vending offers several advantages:

Respects data access controls
: The principal accessing the data must have the necessary read or write access granted by the owner, ensuring that data access is controlled and secure.

Does not bypass security measures
: Credential vending does not bypass firewalls or IP access lists, providing an additional layer of security. To enable external clients to access data, you must configure network access to the Unity Catalog environment and the cloud storage layer.

Time-bound access
: The generated token has an expiration time, defining its lifetime. After this time has elapsed, access is revoked, ensuring that data access is temporary and secure.

Audit logging
: The *audit* system table records all API invocations, specifically capturing events with the `action_name 'generateTemporaryTableCredential'`. This allows for tracking and auditing every API call, including the principal making the request and the timestamp of the invocation. This provides a clear audit trail, enabling monitoring and analysis of API usage and access to data.

With the credential vending API, any client—whether a Delta or Iceberg reader or writer—can integrate with Unity Catalog and access data stored in Unity Catalog. This open access and Delta Sharing make Unity Catalog a highly open and interoperable catalog for managing and governing data and AI assets. This enables seamless integration with various tools and engines, providing a unified and governed data management experience.

Catalog Interoperability

The final piece of the open access puzzle is catalog interoperability. Imagine being part of a large enterprise with multiple teams using different catalogs and systems. It's like having multiple islands of data, each with its own unique ecosystem. Unity Catalog bridges these islands as an open catalog, enabling seamless integration with other major catalogs in the data and AI space. For instance, via JDBC-based query federation, LF allows you to access and govern data from external systems, including catalogs like HMS, AWS Glue, and Snowflake Horizon. The open API also enables external catalogs to integrate with Unity Catalog. For example, the mirroring capability in Microsoft Fabric connects to Unity Catalog, allowing access to the data in Microsoft Fabric. The catalog integration in Snowflake uses the Unity Catalog Iceberg REST catalog API to read Iceberg tables in Unity Catalog.

Interoperability enables you to break down data silos and create a unified data landscape, where data can be accessed and governed across multiple catalogs and systems. No more data fragmentation, no more duplication of effort. Just a seamless, open, and interoperable data ecosystem.

Summary

Open access to data is one key consideration when choosing a data platform and a data catalog. We are far from an era of closed data formats and vendor-locked proprietary tools that make migrations harder when adopting the latest technology. Although Unity Catalog is available in an OSS flavor, it will take time to reach the maturity of the Databricks version. You can bridge the gaps by leveraging the open APIs and data sharing ecosystem.

The long-standing debate over file formats has largely been resolved through file format interoperability. The next decade is for catalogs. Only time will tell who wins, but ultimately, the end users benefit from the innovations in data governance. Chapter 10 discusses compliance with regulatory standards like GDPR while using Unity Catalog as your data governance tool.

CHAPTER 10
Being Compliant with Regulatory Standards

The art of progress is to preserve order amid change and to preserve change amid order.
—Alfred North Whitehead

Nexa Boutique, being a business-to-consumer (B2C) company, collects and stores users' data. When Nexa expanded to Europe, it was in for a surprise. It's well-known that data privacy laws in Europe and the UK are relatively strict; however, what organizations outside of Europe usually fail to understand is that the general public in Europe really cares about their privacy. They are very cautious about sharing their information with others, especially digitally. Karthik has been living in Germany for more than a decade, and he has seen the stark difference between Germany's approach to data privacy in comparison to other countries such as India and the US. The reasons are rooted in historical experiences, cultural values emphasizing human dignity, and a robust rights-based legal framework.

We have observed that engineers, scientists, entrepreneurs, and anyone owning or running a business do not tend to like regulatory standards and compliance requirements. These standards and requirements often make it harder for businesses to operate and cost them significant time and effort, and noncompliance is not an option. This indeed hinders innovation and blocks companies from progressing faster. Data scientists or ML engineers having unrestricted access to all sorts of datasets are able to build well-performing models in comparison to those who get only filtered and limited datasets. The more data you feed the ML models, the better they perform. The scientists and engineers who are driven by innovation and the entrepreneurs who are focused on problem-solving, at least the majority of them, do not have any insidious reasons for collecting and using sensitive data. They just want to optimize what they're doing. However, this approach of unrestricted

access to sensitive data, especially data related to individuals, can lead to unintended consequences.

As a thought exercise, suppose your car collects your driving data and shares it with your insurance company. Your insurance company determines that you are an irresponsible driver based on some criteria defined by them. Your insurance premium will go up, even if you do not make any mistakes. A more serious case would be if your smart watch collects your health data and sells it to your health insurance provider, which decides to increase your health insurance premium purely based on this data. Would that be fair to you? You can probably think of more such examples. Engineers want to optimize performance, and companies want to optimize profits. So, with unrestricted access to data, everyone wins—except the individuals whose data is being collected and misused. And these individuals, including us, are the majority.

Those were just thought exercises, but such practices are not uncommon. For example, employees of the popular American ride-sharing company Uber allegedly had access to a tool called God View, which allowed them to track riders' real-time location without their consent. This constitutes unauthorized surveillance. Cambridge Analytica, a British political consulting company, reportedly misused the data of up to 87 million Facebook users to interfere in 2016 US elections. It acquired the Facebook users' data from an external researcher who had informed Facebook that the data was collected for academic purposes.

Of course, the consequences of unrestricted access to individuals' data can be more severe than just higher insurance premiums. Europe has firsthand experience of its dangers, as evidenced by what happened during the eras of National Socialism and Communism. It's no surprise that the Charter of Fundamental Rights of the European Union establishes data protection as a fundamental right. Nevertheless, Europe's privacy-first approach to data and AI is not merely a reaction to technological change or historical abuses but is grounded in centuries-old philosophical commitments to dignity, autonomy, and the right of individuals to control their own lives and information. These values have been codified in law and continue to inform Europe's robust privacy protections today.

Regulations in principle are not a bad thing; in fact, they are necessary for individuals' rights and interests. The problem comes from improper planning and execution by uninformed politicians and bureaucrats. For regulations to have the right and appropriate impact, all the stakeholders—such as organizations, data and AI experts, entrepreneurs, politicians, bureaucrats, and so on—need to work together and come up with a feasible and practical solution that is fair for individuals as well as organizations. As with anything else in life, the best approach is to strike a balance and not go toward extreme ends of the spectrum, either toward the left or the right.

If you're in a position to make data platform decisions, you should take regulations seriously and work toward designing platforms that make it easier to be compliant. You have to learn to respect individuals' rights and their digital identities. It's important to keep in mind that what you do with your machines and scripts can have a significant effect on individuals' lives, not to mention your company's legal and financial well-being. Keep telling yourself that data security and compliance are topics that are not just materialized by technological implementations but require a shift in the cultural values of an organization and the mindset of its employees. Being compliant is not just for the sake of compliance and avoiding a huge fine, but also for the sake of society and ourselves.

Nexa had high standards for data protection and security; however, when the company entered Europe, it realized that having secure platforms and practices is not enough; it also needed to prove it if and whenever needed. Nexa also had to comply with EU- and UK-specific compliance requirements, namely the General Data Protection Regulation (GDPR). They could not afford to make mistakes or take things lightly since they were collecting and storing personal data of EU and UK residents. Moreover, this was just the beginning. More regulations, such as the EU AI Act, were yet to come. This trend of increased regulation has started to spread like wildfire across the globe. For example, India enacted the Digital Personal Data Protection Act in 2023. By 2025, 20 US states will have enacted comprehensive privacy laws.

Compliance is one of the key aspects of data governance, and like Nexa, more and more organizations have realized that they need to deal with increasing and evolving regulations related to data and AI. Many have started working toward establishing teams, processes, and frameworks to be able to handle this. However, many of them are facing various challenges, and based on our experiences, three of the main ones are as follows:

Understanding what being compliant means
: Compliance requirements are often described in legal terms, and it's nontrivial to translate them into technical requirements. Most of these regulations come from politicians, who do not have an understanding of how the technologies that they are regulating actually work. How can you be compliant when you don't know what being compliant means?

Significant effort and cost
: The effort and cost involved in making sure that you are compliant can sometimes exceed the benefits of implementing the use case itself. For smaller organizations, it can significantly affect their business model and even threaten their existence. In extreme cases, some organizations might simply decide to quit. For example, in 2018, a couple of ad-tech firms, Drawbridge and Verve, decided to shut their operations in the EU, citing GDPR compliance requirements as one of the key reasons.

Paying off technical debt

If you're designing a data platform from scratch and are aware of the compliance requirements, it's relatively easier to take those requirements into account. However, if your data landscape has already evolved like an unplanned city over time, without much control or oversight, then it's harder to be compliant. This challenge sounds like a curse, but in our view, it's really a boon in disguise. As an organization, you are, in fact, forced to get your affairs in order and design, or let's say, *redesign* your platforms. The only way out of the technical debt that you've accrued over the years is by unifying the tech stack, consolidating the platforms, and breaking down silos.

The substantial majority of data and AI regulations are related to personal data protection. Therefore, in this chapter, we focus mainly on handling PII data on the Databricks Platform. We will also discuss how best to design the data and platform architecture to simplify compliance. We will pick GDPR as an example to discuss the best practices, as it's regarded as one of the best and most influential data protection regulations globally. Although we focus on GDPR, the same approach applies to other data protection regulations as well.

The goal of this chapter is not to provide you with concrete steps to be compliant with GDPR. It's to show you how to deal with such compliance requirements and what things you need to be aware of.

GDPR Compliance

GDPR is a law. As an engineer or an architect, reading, interpreting, and applying it in the form of architectural decisions, processes, and practices is a complex endeavor. Nevertheless, GDPR provides seven principles based on which the law is defined. These principles make it much easier for you to interpret, understand, and implement the steps that will lead you toward compliance. These seven principles that apply to the storage and processing of personal data are as follows:

1. Lawfulness, fairness, and transparency
2. Purpose limitation
3. Data minimization
4. Accuracy
5. Storage limitation
6. Integrity and confidentiality
7. Accountability

If you want to dive deeper into the details of the GDPR principles, head to the GDPR website.

When you look at GDPR through its principles, it doesn't feel overwhelming compared to when you look at it as a law. The terms in these principles are familiar, and they also make sense even to those who aren't familiar with legal terminology. Now, if you try to map these principles to data and AI platform features, processes, and practices, on a very high level, you will get something like the following. These primarily apply to the data and an AI asset that contains PII:

- They should be ingested into the platform only for specific, explicit, and legitimate purposes.
- The quality of these data and AI assets should be maintained even after processing or transforming them.
- Every event associated with these assets should be audit logged.
- These assets should be stored and processed in the most secure way.
- Prevent unnecessary duplication of these assets.
- They should be deleted/purged after the intended use.
- These assets should be accessible to only the intended users.
- They should be contained and dealt with within the dedicated zone(s).
- These assets and the use cases that they are used in should have clear ownership.
- It should be possible to track the lineage of all these assets.

This is not an exhaustive list, and these are not descriptive enough. The idea of listing this is to show you how you can map the GDPR principles to data and AI platform activities. You won't be automatically compliant just by implementing these in your platform. However, these would make it much easier for you to be compliant. For example, if you have proper access controls, auditing, and lineage in place, it will be easier for you to enforce GDPR's "right to be forgotten" rule by being able to delete or purge the records corresponding to an individual who requests it.

Enforcing GDPR's "right to be forgotten" rule, especially within the data platforms, is nontrivial because of the following challenges:

Identifying what to delete
: When a user requests that their personal data be purged, you must first figure out what to delete. It's not easy, as the data may not be attributable to the individual who is requesting it unless the data design allows for it. Therefore, it's crucial to

design the data layout and architecture keeping such requirements in mind and choose the platform that allows for it.

Actually deleting the data
: This can be difficult if you store data in formats such as Parquet, which do not allow specific deletions. You need to create completely separate files, capturing everything but the ones you need to delete. However, it is indeed simpler if you are using transaction log-based table formats such as Delta Lake, Apache Iceberg, or Hudi.

In essence, complying with GDPR is not different from following data security best practices and principles, such as the principle of least privilege and zero-trust access model. You should follow a security-first approach while designing your data and platform architecture and have a proper incident response and recovery strategy in place. You should take every measure possible to safeguard the data and AI assets that consist of PII. If you don't do that or cannot prove that you have done that when audited, then you have to face the consequences.

The Platform Decision

With enough time and effort, you can make sure you are compliant with GDPR and similar compliance requirements, irrespective of which platform you choose. However, you also need to keep in mind that you won't have unlimited time and budget to secure your platform and assets. Therefore, decision makers often end up in a dilemma of whether to onboard use cases with PII and invest significant time and capital into them, or to just stay away from them and lose the potential benefits.

If you have a legitimate use case and are convinced that ingesting data with PII into your data and AI platform makes sense, then it would be wise to make sure that your platform is GDPR ready. A platform being GDPR ready means two things:

- The vendor who provides the platform is GDPR compliant.
- The platform provides all the necessary features that make it possible for you to be GDPR compliant.

Databricks is one such platform, although not the only one. Databricks did two things right that have helped it evolve into an enterprise-ready solution that enables organizations to easily implement data and AI security and governance. The first is offering the lakehouse paradigm with support for open table formats such as Delta Lake and Apache Iceberg, and the second is introducing Unity Catalog. Without a proper governance solution, any platform will struggle to offer a unified and secure way to implement data and AI use cases. Databricks seems to have realized this sooner than most others.

Choosing the right platform will solve several of your compliance-related challenges. Moreover, having a single and unified platform that can govern all kinds of data and AI assets and enable all kinds of use cases significantly simplifies the compliance journey. If you're starting from scratch, it's worth spending time on the platform decision, as this will directly address the two challenges that we talked about earlier by significantly reducing the costs and effort involved in being compliant and preventing the buildup of technical debt.

Simplifying the Compliance Journey on Databricks

The decision to go with the right platform is just the first step. Your responsibility regarding compliance does not end there. In the next step, you need to focus on the architecture, processes, and practices.

Treating Data and AI Assets as Products

One way to increase accountability and boost the quality of your data and AI assets is to treat them as products. The concept of *product thinking* for data is not new and has been around for a couple of decades. The formalization of this approach was accelerated by Zhamak Dehgani's introduction of the principle of data as a product in her 2019 book *Data Mesh* (O'Reilly). But why does this shift toward treating data as a product help with compliance requirements?

Let's go through some of the key aspects of software products, a concept that is familiar:

Purpose-built
: Software products are built based on specific requirements and for particular uses. Typically, you don't build a software product and then look for who needs it; rather, you start with something that someone needs and build a software product that fulfills their requirements. You don't stop there. You iteratively improve the product based on user feedback.

Lifecycle management
: Software products follow a lifecycle that includes planning, development, deployment, maintenance, and eventual retirement. They're versioned, updated, and enhanced to meet evolving user requirements.

Discoverable and accessible
: These products are easily discoverable and accessible by the intended users, facilitated by clear documentation, standardized interfaces, and mechanisms that allow users to find and use them internally within an organization or externally.

Interfaces
: Typically, these products use defined API interfaces to facilitate integration with other systems and enable automation and interoperability. These stable interfaces provide users with certainty regarding what to expect.

Quality and reliability
: Quality and reliability are important for functions to work as intended and for continuously delivering value. They're achieved through monitoring, rigorous quality assurance, testing, and validation.

Security and access controls
: Proper security measures and access controls are established to allow only intended users access to the product. Appropriate authorization allows for selective access to the product features.

Ownership
: Software products typically have product owners or managers who are accountable for the overall product. They make strategic decisions and ensure value delivery.

You can apply these same aspects to data and AI assets, which include tables, views, AI models, and so on. However, some aspects of software products cannot be *exactly* mapped to data products. For data and AI assets to be considered a product, in the context of a lakehouse paradigm, they should fulfill the following criteria:

Valuable and trustworthy
: A data or AI asset or a group of assets should be high quality, reliable, trustworthy, and independently valuable.

Self-descriptive
: A data product should provide a clear perspective of what it is and what it offers. This should include a human-understandable title and description as well as machine-readable metadata, tags, and attributes.

Interoperable
: In the context of a lakehouse, it should be possible to easily combine data products meant for use together. This can be achieved by using a consistent storage format or having a common interface to access the data in different formats. It's also important to ensure semantic consistency across data products through shared definitions of terminologies and common governance rules.

Discoverable and accessible
: A user should be able to discover the available data products and determine if they are suitable for their use case. If they're able to locate a relevant data product, there has to be a way to access it, preferably using a platform or tool familiar to the user.

Secure
: The data products should be secured at the platform level and have proper access controls. They must abide by organizational policies and regulatory requirements. Data products containing PII data should be handled appropriately by anonymizing, masking, or encrypting the assets if needed, granting access to only relevant/certified users, limiting storage to specific environments, following data residency requirements, and so on.

In addition, a data product:

- Is built for specific users
- Provides details using a data contract
- Has an owner
- Follows a well-defined lifecycle
- Is published using a well-defined governance framework

Data Contracts

A data *contract* is a formal agreement between data producers and data consumers that typically specifies the structure, format, semantics, and terms of use of the data and AI asset being provided. The main goal is to ensure the reliability, quality, and consistency of these assets. Data contracts bring clarity and certainty for consumers, enabling them to not only understand what they're getting but also be clear on aspects such as data freshness and availability.

On the other hand, having a contract means additional effort for the data producers, as they have to ensure that the data products adhere to the corresponding data contracts. Therefore, producers typically dislike data contracts and consumers love them. This is one reason it's hard to implement data product concepts in organizations.

If you want to get started with data contracts, refer to the *open data contract standard* provided by Bitol, an open source Linux Foundation AI & Data Incubation project focused on establishing open standards for data contracts and data products in modern data engineering.

These data product criteria are based not just on theoretical understanding of the concepts or creative thinking, but on our experience working with various customers across the industry. We have had hundreds of endless discussions on what makes sense and what is, in fact, feasible. We have seen and experienced firsthand the implementation of data product concepts, which surprisingly are very different from one another. That's why we do not prescribe these as must-have criteria; rather, we refer to these as *principles*, and there is always room for customization. The real

world is significantly different from the prescribed theoretical concepts, and it's more so for concepts such as data products and data mesh that are cross-functional and concerned with the entire organization.

In this book, we won't discuss the details of building data products on Databricks. If you're interested in learning more, see this blog post and this video.

In essence, the criteria for developing data products bring strict discipline to the processes of creating, publishing, and maintaining the assets. Having an owner for each data product brings accountability. Data contracts bring clarity and certainty. The need to maintain quality, reliability, and consistency means the correctness of PII, if any, is maintained. Finally, the security and governance practices enforced by this approach reduce the probability of security incidents. Therefore, treating data and AI assets as products inherently makes it easier to be compliant.

Detecting and Securing Sensitive Data

Different categories of data exist in organizations, ranging from publicly shareable to confidential and restricted ones. You need to carefully consider what categories of data you choose to ingest or onboard onto your data and AI platform. Karthik's first reaction to someone who wants to talk about handling PII or GDPR compliance is: *what is the use case?* and *Do you really need it?* It's really important to make sure that the use case is truly worth it and will result in a significant return on investment before getting started with it.

Many organizations think that they need to use all their data to get value out of it. They don't realize what they're getting into when they decide to ingest data with PII into their data platform. This is also precisely why GDPR exists in the first place: to prevent organizations from using individuals' personal data unnecessarily or without their consent. So, before you ingest or onboard PII data into your data platform, it's crucial to make sure that you have a solid use case and that you're allowed to use the corresponding PII data for it.

When it comes to ingesting and processing PII on your data platform, you'll come across three main scenarios:

Scenario 1

You know where the PII data exists in your operational data sources and choose not to ingest it into your data platform. This is the safest scenario, and you don't need to do anything else.

Scenario 2

You know where the PII data exists in your operational data sources and choose to ingest it into your data platform. In this case, you have additional steps to take to secure the PII once it's ingested or while it's being ingested, but you are in control.

Scenario 3

You don't know where the PII data exists in your operational data sources, as it might be too expensive or infeasible to detect it directly in the source systems, and you ingest data into your data platform. In this case, you have two steps: detect where PII is during or after ingesting the data and secure it. Hence, this is a complex scenario.

Basically, you have two main problems at hand: to *detect* and to *protect* PII data. There are multiple ways to detect PII data on the Databricks Platform, which can be categorized into built-in solutions that are offered out of the box by the platform and built-on solutions that are custom solutions that you deploy within the platform. The built-in solution includes the data classification feature, which you could enable at the catalog level. Once enabled, the data classification engine, powered by a compound AI system, automatically classifies and tags the tables and corresponding columns with predefined system tags. The best thing with this approach is that it automatically determines when to run and classify newly ingested data without any manual intervention. The results of the classification can be reviewed through a dashboard within the platform.

If the built-in solution is not sufficient for your use cases, you can build custom solutions based on open source tools and even leverage GenAI models. There are multiple open source tools capable of identifying sensitive data in an automated fashion, with different degrees of precision. One of the most popular tools is Presidio from Microsoft. It has preconfigured profilers and can use deep-learning libraries to identify sensitive information. You can combine it with other open source tools, such as the Databricks Labs project DiscoverX, to scale the PII detection and tagging.

A more advanced do-it-yourself approach would be to use GenAI models dedicated to sensitive data identification, such as the Roberta Large NER model and Patronus AI EnterprisePII. However, you might need to customize the LLMs by training/fine-tuning them against your data to account for your organizational context and semantic meaning to achieve precise results.

Apart from the technical solutions, you should also consider the human element involved in securing PII. Enabling the data teams within the organization to identify and handle PII data through training and extensive documentation will empower them to act as human firewalls. Moreover, combining these efforts with the automated tooling discussed before will significantly boost the organization's security stature.

The result of detecting PII within the Databricks Platform is persisted in the form of tags in Unity Catalog. The tags are typically applied to tables and columns. Once you have this information, you can then secure these tables and columns tagged as PII. Depending on the use case and regulatory requirements, you can choose the appropriate next steps, which are as follows:

1. Do nothing and continue to use the data as is. This is certainly a valid option if you have relevant and approved use cases for using the data and have proper access controls and other security measures in place that permit authorized access to only relevant individuals.
2. Drop the table or column. If you don't have a relevant use case or if you're not permitted to use the data, the best next step is to delete or purge it.
3. Anonymize the data. Depending on the use case, you can choose to anonymize or pseudoanonymize the data.

Figure 10-1 shows the steps in detecting and protecting the PII data for data ingested into the data platform.

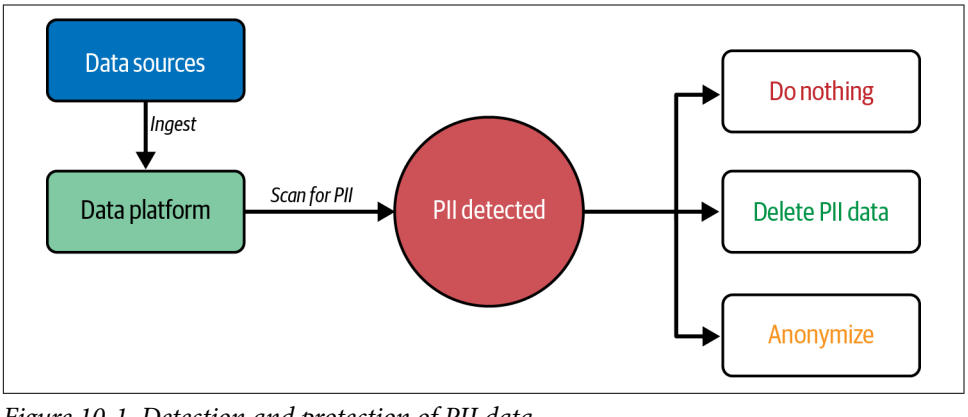


Figure 10-1. Detection and protection of PII data

There are different anonymization techniques available that you can apply to data in the Databricks Platform. Hashing is a technique that is irreversible, with data consistency maintained. Tokenization can be used to encrypt sensitive values with or without format preservation. Tokenization comes in two flavors:

Vaulted tokenization
: Replacing sensitive data with tokens and storing the mapping between the original values and tokens in a secure store referred to as a *token vault*.

Vaultless tokenization
: Replacing sensitive data with encrypted values, eliminating the need for a token vault. *Format-preserving encryption (FPE)* is a commonly used encryption method.

In either case, securing the key or token mapping is crucial to protect sensitive data from unauthorized access because these approaches are reversible. You can also use built-in Databricks functions such as encryption, hash, and mask for a streamlined approach to protecting sensitive data.

You can also leverage FGACs, namely row filters and column masks, offered by Unity Catalog. This mechanism allows you to apply complex logic to the tables during runtime to mask certain columns or filter out a subset of records from the output for the given user without the need to copy the filtered results into a new table or view. For details, refer to the Fine-grained access controls section in Chapter 5.

Architecture Best Practices for Handling Sensitive Data

The data and platform architecture can play an important role in improving security, reducing the effort, and optimizing the cost of processing sensitive data. The Databricks Platform and Unity Catalog features provide a way to isolate the processing of sensitive data. Think of a dedicated and isolated zone where you ingest and process sensitive data. It can also act as the space where you determine if the ingested data contains PII. This is the place where you also perform the steps to protect the PII data by either dropping or anonymizing it.

You can create such a zone in Databricks by creating dedicated workspaces with maximum security configurations, such as frontend and backend private links. You can also create dedicated catalogs with managed locations pointing to dedicated cloud storage units. Bind these catalogs to the corresponding workspaces and allow only certified users to access the workspaces and the catalogs. We term this dedicated and isolated PII data processing zone as the *de-identification zone*. This zone could be used as an additional operational tier or environment in addition to the regular development, staging, and production ones.

The datasets that definitely contain PII and those that might contain PII should be ingested in the de-identification zone, where the detection and tagging of the ingested data are performed. Then, the steps to protect the PII data are executed. Depending on the use case, you would then move the data, either anonymized or otherwise, into the staging or production environment. The development environment usually consists of only mock data. Figure 10-2 depicts a de-identification zone along with other environments.

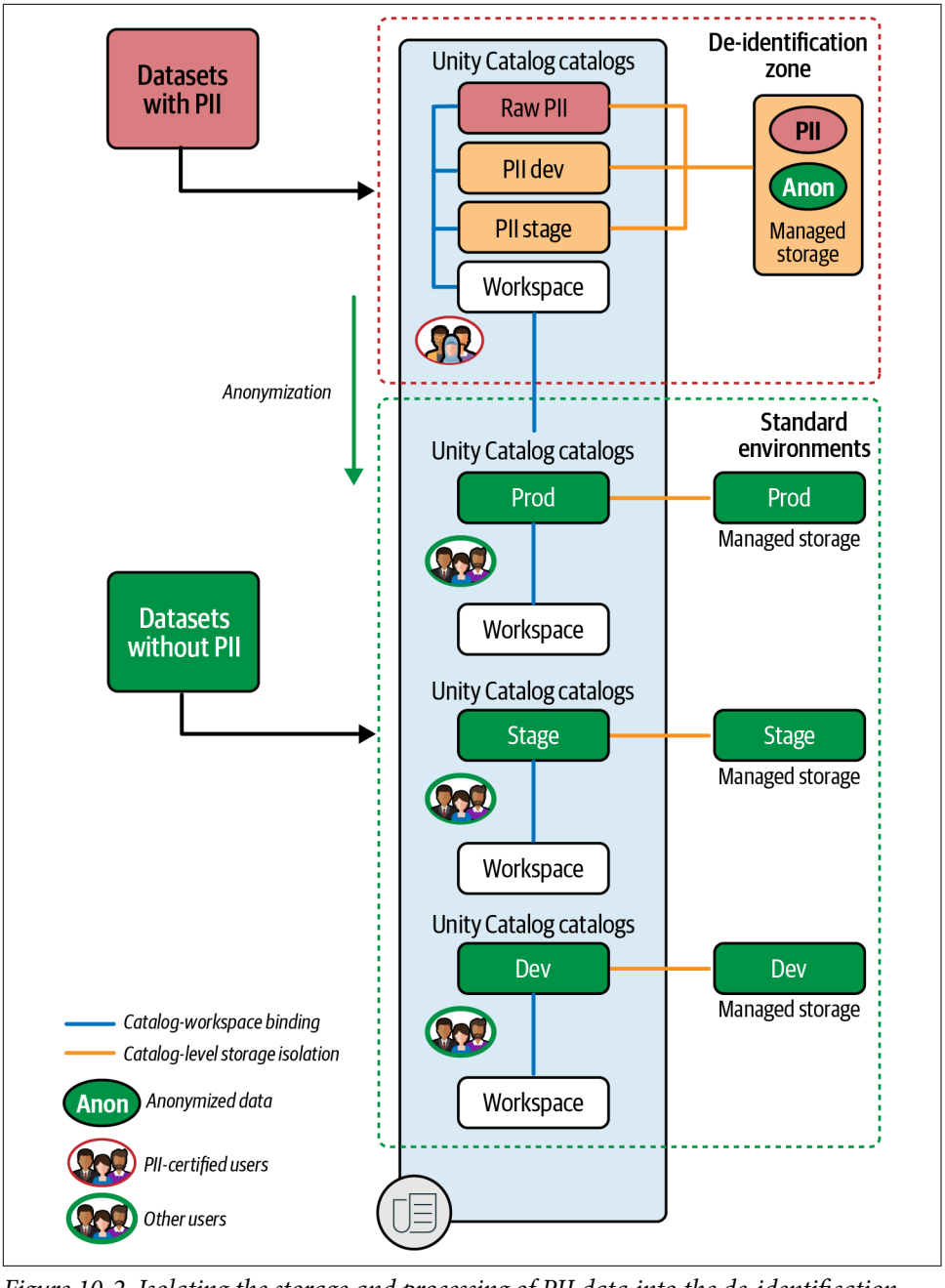


Figure 10-2. Isolating the storage and processing of PII data into the de-identification zone

Databricks workspaces in the de-identification zone should be configured with the highest level of network security to protect against both external and internal threats. This can be achieved by leveraging the private connectivity options (AWS, Azure, GCP) and setting up a data exfiltration protection architecture (AWS, Azure, GCP).

Because both Databricks workspaces and Unity Catalog catalogs are isolated, managing access becomes straightforward. You should establish dedicated groups within the identity provider, ensuring that members are ideally trained and certified to handle PII data. These groups can then be provisioned to the Databricks account and assigned to the appropriate workspaces. Access to catalogs containing PII data should be limited exclusively to these certified user groups and to dedicated SPs used for automation. If there is a need to use PII data outside the de-identification zone, FGACs can be implemented to grant selective access to only certified users and SPs.

Nexa Boutique had the right platform in place for efficient handling of PII data. After expanding to Europe, they made sure to leverage all the Unity Catalog features to the maximum extent. They already had plans to implement the data mesh concept, so they made sure to treat all their data and AI assets as data products. It was difficult in the beginning, as some of the data teams were reluctant and pushed back on the idea of data products and data contracts. But after establishing the Databricks center of excellence (CoE) team, which did a great job evangelizing the skeptical data teams to adopt the data product concepts, things started to fall into place.

Nexa had or undertook the following:

- The right platform, processes, and practices that followed the data product principles.
- Made sure to follow the best practices suggested by Databricks for handling PII data.
- Set up several de-identification zones, one in each cloud region, to ingest and anonymize PII data.
- Perfected the art of securing the PII data based on the use cases by using the right anonymization techniques that balanced security, performance, and usefulness of the data.
- Regularly organized training to educate and empower their entire workforce on handling sensitive data.
- Set up a proper incident response playbook for their teams to be able to handle security breaches.

Of course, they had to take a ton of additional measures that are not directly related to the data platform. But now they can confidently say that they are compliant with GDPR.

Summary

In this chapter, we explored concepts related to regulatory requirements, especially around personal data protection. We saw why it is important, what the challenges are in being compliant with those requirements from the data platform perspective, and how to address them. Starting with choosing the right platform, embracing the data product principles, and finally, focusing on the right data and platform architecture simplifies the compliance journey for both large and small organizations. In Chapter 11, we will explore different ways to accelerate Unity Catalog adoption within your organization.

CHAPTER 11
Accelerating Unity Catalog Adoption

Unity Catalog transforms how organizations manage data and AI governance within Databricks. The system represents a fundamental shift in data governance strategy, introducing a transformative approach to metadata management that empowers users with remarkable control and visibility of their data and AI assets. Organizations can now implement enterprise-grade governance without sacrificing agility or performance.

For early adopters of the Databricks platform, the journey involves migrating existing HMS-based workspaces to the new Unity Catalog-based architecture. While requiring careful planning, this transition unlocks significant governance capabilities and enhanced security features that weren't previously available.

In contrast, organizations building new solutions on Databricks enjoy a more streamlined experience. These users can implement Unity Catalog from the ground up, free from legacy configurations or technical debt that might complicate adoption. With Databricks now setting Unity Catalog as the default catalog for all new workspaces, these users benefit from best-in-class governance tools from day one.

The inevitable phasing out of HMS in new workspaces signals Databricks' commitment to Unity Catalog as its strategic governance solution. This evolution has created three distinct adoption patterns:

Early adopters
: Organizations that have already completed their transition to Unity Catalog and are leveraging its full capabilities

Migration users
: Teams currently navigating the journey from HMS to Unity Catalog, balancing existing workloads with governance enhancements

Unity Catalog-native users
: New Databricks customers starting fresh with Unity Catalog as their default catalog system

Each group faces unique challenges and opportunities in their governance journey. Understanding these different perspectives is essential for organizations seeking to maximize the value of Unity Catalog while minimizing disruption to their data workflows.

This chapter will cover some key topics related to migration and adoption to help you effectively leverage Unity Catalog based on your specific stage of Databricks adoption. Databricks now provisions all new workspaces with Unity Catalog as the default catalog, transforming how you interact with your data environment right away. We'll examine the powerful characteristics of this architecture and reveal how it differs from traditional HMS-default environments. You'll gain insights into the significant implications for your data management strategies, security posture, and governance capabilities.

Most Databricks users face the critical task of transitioning their HMS-based workloads to the new Unity Catalog architecture. We'll guide you through the essential considerations and proven best practices that ensure a smooth migration journey. Discover how UCX, a powerful Unity Catalog migration tool built by Databricks Labs, dramatically accelerates this transition, helping you unlock the full benefits of Unity Catalog quickly while minimizing disruption to your existing workflows. Adopting Unity Catalog for seasoned HMS users requires a migration of assets to Unity Catalog, which can sometimes hinder adoption. HMS federation enables you to federate external HMS, AWS Glue, and Databricks internal HMS to Unity Catalog. By using HMS federation, you can get started with experiencing the powerful features that Unity Catalog offers without any data migration. We'll explore why this is important and when you should use it to accelerate your Unity Catalog adoption.

Automatic Enablement of Unity Catalog

For over a decade, HMS has been the default catalog in Databricks. However, with the release of Unity Catalog, the platform underwent a significant transformation. Early adopters of Unity Catalog had to navigate a migration process during which Unity Catalog and HMS coexisted within the platform. To ensure a seamless transition, Unity Catalog compute was made compatible with HMS, allowing users to migrate to Unity Catalog without disrupting existing workloads.

This compatibility enabled users to query and transform their data without interruptions, even as it resided in both HMS and Unity Catalog. However, new users can start using Unity Catalog immediately without the need for any migration.

Starting November 9, 2023, new Databricks workspaces created in eligible cloud regions are automatically enabled with Unity Catalog as the default catalog, provided

all other eligibility criteria are met. This automated process eliminates a significant portion of the administrative effort, simplifying the Unity Catalog journey. In contrast, manual enablement of Unity Catalog requires creating and attaching metastores to a workspace, defining storage locations and credentials, creating catalogs, and binding them to the workspace, as discussed in Chapters 2 and 5. With automated enablement, Unity Catalog is ready to use instantly, allowing users to dive right in. In this chapter, we'll explore the details of getting started with Unity Catalog by default and what happens behind the scenes.

Databricks Express Setup

The Databricks express setup allows you to start exploring its capabilities for free. Figure 11-1 shows the current sign-up page for the express setup. The express setup provides a seamless onboarding experience by spinning up a serverless workspace. This workspace is fully managed by Databricks, with 100% of the platform running on serverless compute infrastructure. Additionally, the serverless workspace comes with managed storage, eliminating the need for you to create a metastore or storage location to get started with Unity Catalog. Sign up with your business email ID, and you'll receive free credits to try out the platform. Express setup allows you to experience the full range of Databricks features and capabilities without incurring costs.

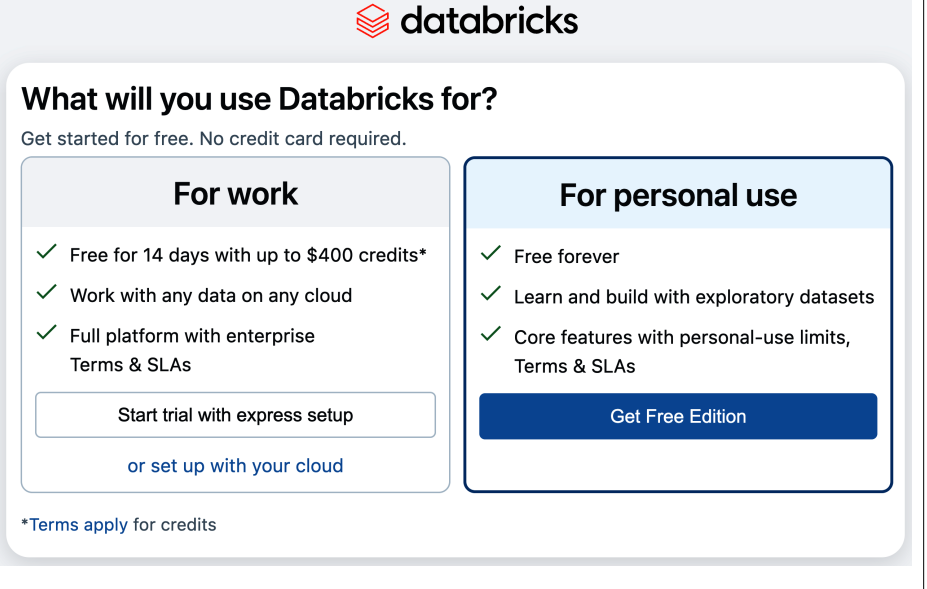


Figure 11-1. Express setup in Databricks for a serverless workspace

As you prepare to enable Unity Catalog as your default catalog, it's essential to consider your current position in the Databricks journey. Two distinct scenarios emerge, each with its own set of circumstances:

Scenario 1

New to Databricks. You're just starting with Databricks; your environment is clean. Specifically:

–No Unity Catalog metastores already exist.

–No workspaces currently exist.

In this scenario, enabling Unity Catalog in new workspaces is a straightforward process.

Scenario 2

Existing Databricks user with assets in HMS and Unity Catalog. You're a seasoned Databricks user who has begun migrating assets to Unity Catalog. Your environment is characterized by the following:

–Existing metastores in your cloud regions.

–Multiple workspaces exist.

–HMS remains in use for at least some workloads.

In this scenario, enabling Unity Catalog in new workspaces requires the manual step to attach the workspace to an existing metastore in the region by an account administrator.

If you're a new Databricks user who just signed up to build your data and AI use cases entirely governed by Unity Catalog, you're at the right place. When a new workspace is deployed to your cloud region of choice, a few things happen automatically in the background to enable Unity Catalog. We will consider Azure as your cloud provider for the rest of the discussion. The process will be similar across other cloud providers as well.

You can verify whether a newly created workspace is Unity Catalog-enabled automatically by running a simple SQL query:

```
SELECT CURRENT_METASTORE()
```

If this returns a metastore ID, your workspace is Unity Catalog-enabled; otherwise, you will find an error message "[`OPERATION_REQUIRES_UNITY_CATALOG`] Operation `CURRENT_METASTORE` requires Unity Catalog enabled."

Default Metastore

As an integral part of Unity Catalog, a metastore is required to enable Unity Catalog for your Databricks workspace. In the Azure portal, a user with the *Contributor* role to the Azure *subscription* can deploy a Databricks workspace within the subscription. The workspace is always deployed in a specified cloud region. Let's assume the workspace is deployed to the Azure North Europe region, which is in Ireland. As Databricks restricts you to creating only one metastore per region, the Unity Catalog metastore deployed automatically will be in the same region, i.e., North Europe, and will become the default metastore in the region. Any new workspace you deploy in the North Europe region will automatically attach to the default metastore deployed because you created your first workspace in the region.

By default, the metastore won't have a dedicated metastore administrator. Instead, the workspace administrator will have the necessary privileges to manage the metastore. This means the workspace administrator will be able to perform key tasks, such as creating new catalogs, external locations, and managing storage credentials. This design allows the workspace administrator to fully control the metastore and create custom catalogs tailored to their specific needs, using their storage locations and credentials.

Centralized Versus Decentralized Administration

Many users have been looking to eliminate the need for a centralized metastore administrator, which can create a bottleneck and hinder progress. In a distributed business where teams require autonomy, having a single point of control can be counterproductive. By delegating metastore administrator privileges to the workspace administrator role, you can empower individual teams to manage their catalogs and schemas in Unity Catalog without needing to navigate multiple levels of approvals. Delegated privileges also align with the data mesh architecture, where each node in the mesh will function on its own with autonomy.

This approach aligns perfectly with business needs, allowing teams to work more efficiently and independently while still maintaining control and governance over their data assets. However, for organizations that typically rely on centralized governance, having a single metastore administrator with complete control can be a suitable approach, allowing tight control over how teams work with Unity Catalog. This also helps enforce organization-wide data governance policies, ensuring consistent access controls, privilege assignments, and compliance standards across all workspaces attached to the metastore.

During our discussions around Unity Catalog, most business units demand control over their data, seeking to isolate it from other units within their organization. To achieve this, they require their data to be stored in a dedicated Azure subscription. However, using the default metastore storage compromises this goal, as data is not separated by subscription. This issue is particularly pertinent when working with managed tables in Unity Catalog, which are stored at the metastore level by default if no alternative storage location is specified at the schema or catalog level. The default metastore will not include a storage location to address this concern. Instead, business units must provide their storage layer at the catalog or schema level under their Azure subscription. This approach ensures that each unit's data remains separate and secure, meeting their data isolation requirements.

Default Catalog

When a metastore is created, a new catalog is automatically deployed and bound to the workspace. The catalog is named after the workspace by default, and its ownership is assigned to the workspace administrators.

A workspace admin can rename the catalog or share it with other workspaces later if required. This will enable the administrators to manage the catalog permissions and create Unity Catalog-managed assets within the catalog. The workspace administrator can also transfer ownership to a different user or identity.

The default workspace catalog is accessible by all the workspace users as they are assigned the `use catalog` privilege by default.

Default Schema

Inside the workspace's default catalog, a new schema named `default` is also automatically provisioned. All workspace users are automatically granted the `use schema, create table, create volume, create model, create function, and create materialized view` permissions on the schema.

In essence, the new workspace you just deployed comes packaged with a Unity Catalog metastore, an assigned catalog accessible only from that workspace, and a default schema with predefined permissions for the users. This will enable your end users to get started quickly on the platform, removing any admin dependency. The deployment process is illustrated in Figure 11-2.

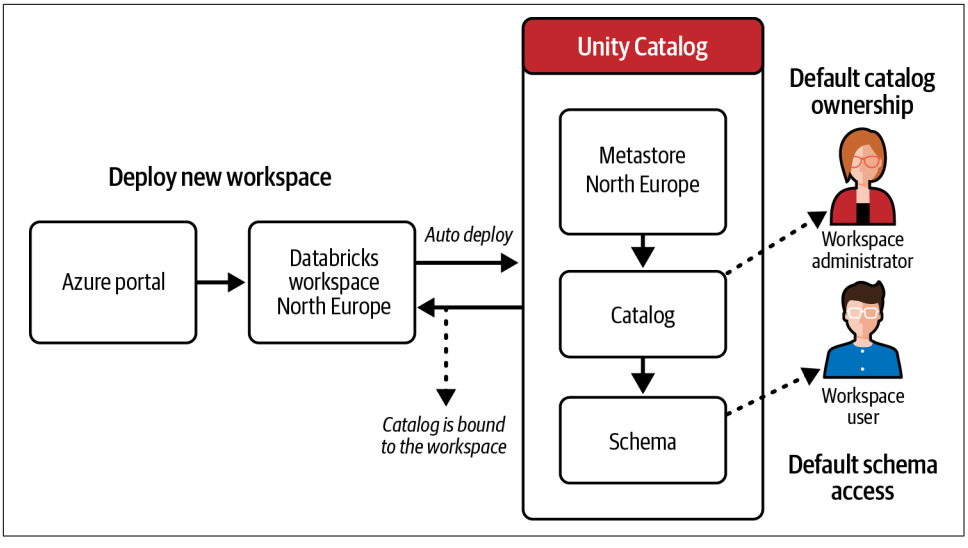


Figure 11-2. New workspace with Unity Catalog attached by default

Creating a new data asset without providing an entire namespace in an automatic Unity Catalog-enabled workspace has a new default behavior: it will now be stored in Unity Catalog rather than HMS. The same applies when you register an MLflow model—it will be registered to Unity Catalog by default instead of the workspace model registry. What's controlling this behavior? The workspace admin can set a new workspace setting called the `default catalog for the workspace`. Setting a default catalog gives you more flexibility and control over where your data and models are stored. For a visual reference, you can find the default catalog setting in the workspace settings, as illustrated in Figure 11-3.

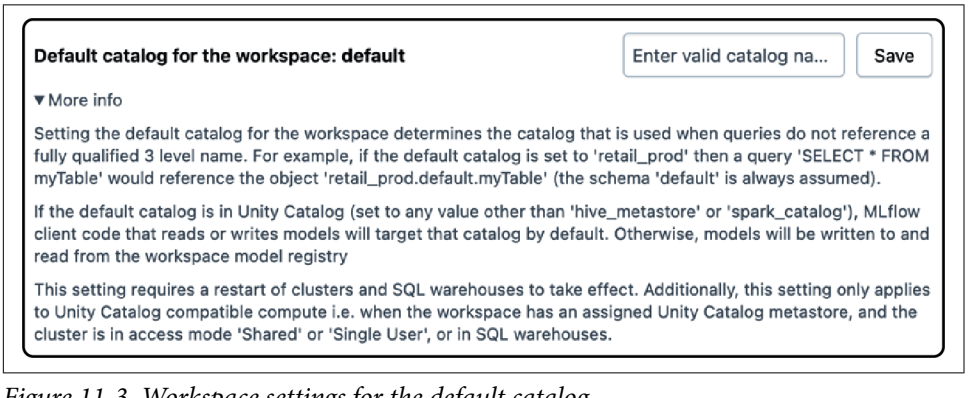


Figure 11-3. Workspace settings for the default catalog

When you deploy your first workspace in a cloud region, the system automatically creates a default Unity Catalog metastore. The platform then links any additional workspaces deployed in the same region to this existing metastore. Although these workspaces share the metastore, each creates and binds its unique catalog during deployment. You can also use HMS and the workspace model registry to register ML models with MLflow in these workspaces. Looking ahead, Databricks will deprecate HMS and workspace model registry in new workspaces, making Unity Catalog the sole catalog option.

If starting from scratch, focus exclusively on Unity Catalog and disregard HMS. This approach helps you avoid potential challenges when switching to Unity Catalog, which requires a migration effort. By starting with Unity Catalog, you set yourself up for success from the outset. For existing HMS users, don't worry—we cover the migration process to Unity Catalog next, enabling you to plan a smooth transition.

Migrating from HMS to Unity Catalog

If you're among the second category of users who have built their Databricks platform around HMS, migrating those assets to Unity Catalog can be complex. However, tools like the upgrade wizard and Databricks Labs UCX can simplify migration.

Upgrade Wizard

The complexity of your migration depends on several factors, including the format and volume of data, your storage configuration, and whether you're using the DBFS root or mounted cloud storage. The migration process might be as straightforward as a single UI click or as involved as writing custom code. The upgrade wizard allows you to select the tables you want to upgrade using a simple interface, and Databricks will handle the migration behind the scenes. You can upgrade an entire schema or select specific tables. The upgrade wizard supports migrating managed and external Hive tables to Unity Catalog.

The resulting tables in Unity Catalog after the migration will be external tables. The upgrade wizard is suitable for small workspaces with fewer schemas and tables, if you prefer to use a visual interface for the upgrade.

To notify users that a Hive table has been upgraded to Unity Catalog, you need to add a comment to the Hive table in a specific format. If you use this format, a warning message will be displayed with the comment whenever a user tries to query the table from a notebook or query editor, alerting them to use the upgraded table in Unity Catalog:

"This table is deprecated. Please use catalog.default.table instead of hive_metastore.schema.table."

UCX

The Databricks Field Engineering team runs Databricks Labs, which develops applications that empower users to build and deploy their use cases more efficiently. By providing these applications, Databricks Labs accelerates the adoption of new features, enabling users to transition their projects to production faster and more seamlessly. UCX is a purpose-built tool for Databricks users to migrate their data assets and workloads in HMS to Unity Catalog. Along with UCX, Databricks also provides inbuilt capabilities that help you migrate your HMS tables to Unity Catalog.

Your starting point would be enabling Unity Catalog for your Databricks account, which an account admin can initiate from the account console. An account admin creates a metastore for the regions you operate from and where your corresponding workspace cloud regions are. Unity Catalog supports one metastore per cloud region. Each workspace from a cloud region can be attached only to the corresponding metastore created in its cloud region. Once a Databricks workspace is connected to a Unity Catalog metastore, it becomes *Unity Catalog-enabled*.

Enabling Unity Catalog also enables identity federation, which means your identities in the workspaces should now be available in the account console to be supported by Unity Catalog. The system automatically syncs user identities into your account console but does not sync groups created by your admins locally in the workspace, as illustrated in Figure 11-4. Databricks flags groups created within the platform as local to the workspace and incompatible with Unity Catalog. To continue working with the same groups you created in your workspace, you should sync them back to your account console.

Based on the source, Databricks maintains four different types of groups:

Account groups
: The groups are available in the Databricks account console and work with the Unity Catalog permission model.

Workspace-local groups
: These are the workspace-level groups that exist only in the workspace in which they were created and are not accessible in other workspaces. It does not work with the Unity Catalog permission model.

External groups
: These are the groups that were synced from your IdP using SCIM to Databricks. External groups are also account-level groups.

System groups
: Databricks creates and maintains some default groups in every account. These are *account users*, *users,* and *admins.*

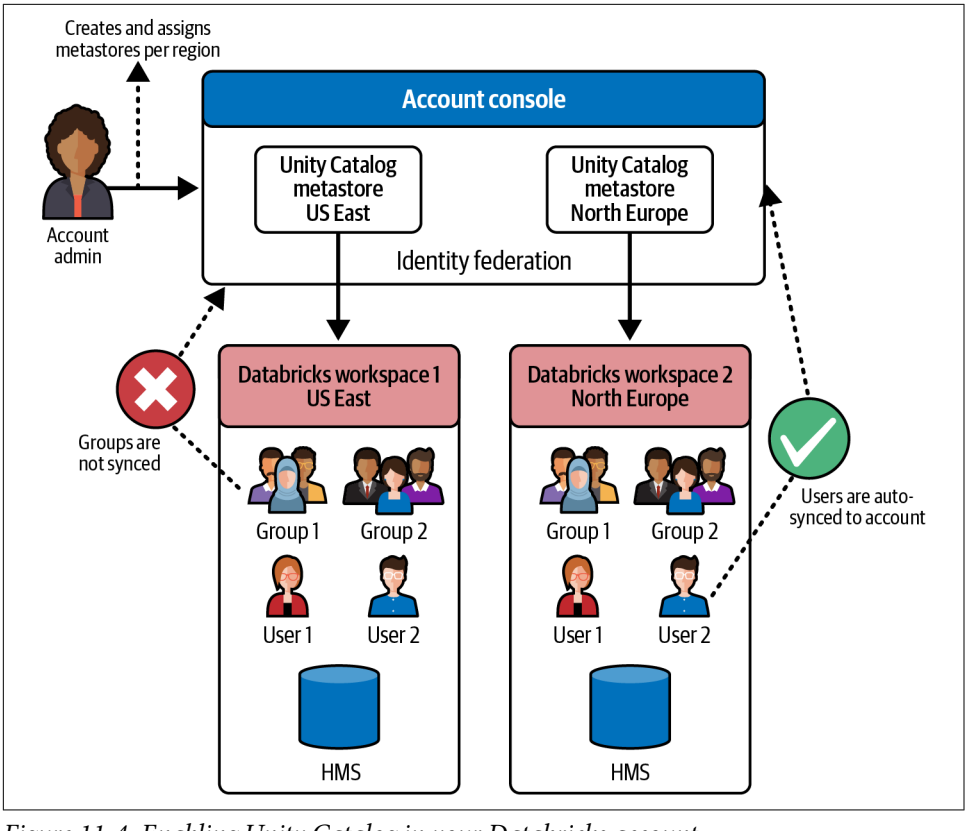


Figure 11-4. Enabling Unity Catalog in your Databricks account

Workspace group migration

You can have Databricks-managed workspace-local groups in your workspace, which grant access and permissions to assets within that workspace. These groups are isolated to the workspace where they were created and cannot be shared with other workspaces. If you currently use workspace-local groups in your Databricks workspace, you must migrate them to account-level groups to use Unity Catalog. Unity Catalog supports only account-level groups defined at the Databricks account and propagated to workspaces. This will enable you to centralize management and ensure consistent access control across your organization. UCX provides functionality to automate the group migration process, which is called the `group migration workflow` module. Over the years, you could have created multiple workspace-level groups to manage access to assets in your workspaces, including:

- Legacy table ACLs and entitlements
- AWS instance profiles
- Clusters, instance pools, and policies
- Databricks SQL warehouses
- Lakeflow Declarative Pipelines
- Jobs
- MLflow experiments and registry
- SQL dashboards, alerts, and queries
- Token and password usage permissions
- Secret scopes
- Notebooks, directories, repos, and files

When enabling Unity Catalog, the workspace administrator must reapply the permissions granted to the groups after re-creating them at the account console and federating them back to the workspace via identity federation. However, doing this manually can be cumbersome and error-prone, which is why the UCX tool is handy for automating the group migration process. UCX is installed from the Databricks CLI after you authenticate to a Databricks workspace from your local development machine:

```
databricks auth login --host WORKSPACE_HOST
```

Once authenticated, UCX can be installed via the Databricks CLI:

```
databricks labs install ucx
```

Once installed, UCX deploys several notebooks, code, workflows, and configuration files into the authenticated workspace. Following is the list of workflows created as part of the UCX deployment:

- `[UCX] assessment`
- `[UCX] failing`
- `[UCX] migrate-data-reconciliation`
- `[UCX] migrate-external-hiveserde-tables-in-place-experimental`
- `[UCX] migrate-external-tables-ctas`
- `[UCX] migrate-groups`
- `[UCX] migrate-tables`
- `[UCX] migrate-tables-in-mounts-experimental`
- `[UCX] migration-progress-experimental`

- `[UCX] remove-workspace-local-backup-groups`
- `[UCX] scan-tables-in-mounts-experimental`
- `[UCX] validate-groups-permissions`

UCX deploys multiple dashboards to interpret your Unity Catalog migration metadata:

- `[UCX] Assessment (Interactive)`
- `[UCX] Migration (Main)`
- `[UCX] Progress (Main)`
- `[UCX] Migration (Groups)`
- `[UCX] Assessment (Azure)`
- `[UCX] Assessment (Main)`
- `[UCX] Assessment (Estimates)`

UCX then runs an assessment workflow, `[UCX] assessment`, which captures all workspace-level asset metadata and securable objects in HMS. The assessment workflow persists all the crawled information in an inventory database in HMS. Next, UCX uses the captured data to hydrate the deployed dashboards to help you analyze the UC migration complexity and compatibility for the specified workspace, as illustrated in Figure 11-5.

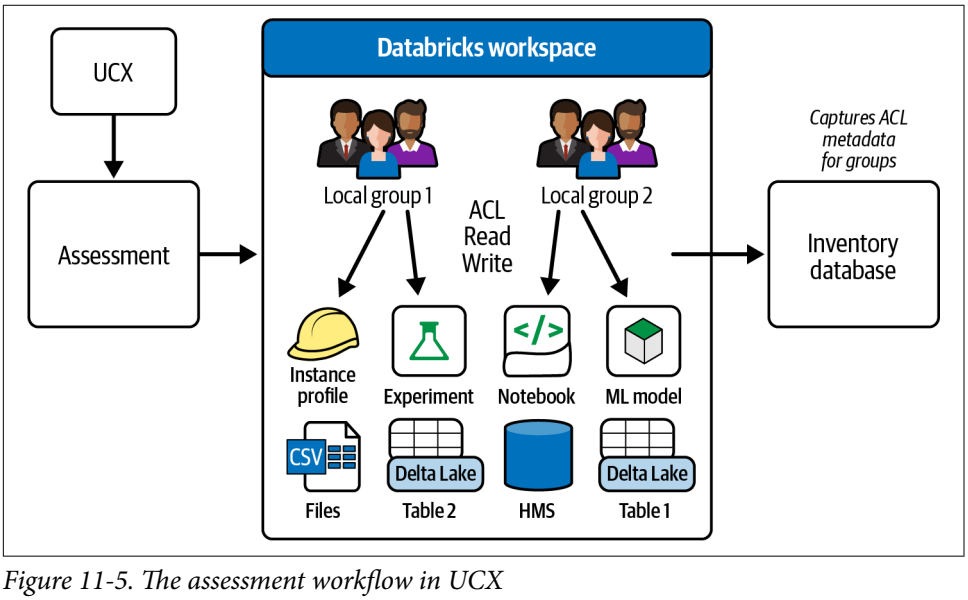


Figure 11-5. The assessment workflow in UCX

Once the assessment workflow is completed, you can trigger the group migration workflow. This process involves several tasks that will help you migrate your workspace-local groups to account-level ones. Your admin first ensures that any workspace local groups are created at the account level, and then the workflow `[UCX] migrate-groups` is initiated. The workflow has the following tasks that get executed:

1. `verify_metastore_attached`
   The first task verifies the existence of a Unity Catalog metastore for the region of your workspace and is attached to it.

2. `rename_workspace_local_groups`
   The second task then kicks in, backing up existing groups in the workspace and renaming them with a prefix.

3. `reflect_account_groups_on_workspace`
   The third task adds the corresponding account-level groups to the workspace, ensuring a smooth transition.

4. `apply_permissions`
   The fourth task applies the existing permissions for the groups to the new account-level groups, ensuring that access and control remain intact.

After completing the workflow, run the `[UCX] validate-groups-permissions` workflow to validate the permissions. If you're satisfied with the group's current permissions after validation, the `[UCX] remove-workspace-local-backup-groups` workflow can be initiated to remove the workspace backup groups, completing the group migration process.

By moving your workspace-level groups to account-level ones, you're taking a significant step toward Unity Catalog adoption and setting the stage for successfully migrating your workloads to Unity Catalog.

Table migration

Due to the significant differences in storage, cataloging, and governance models between the two systems, most of your time during a Unity Catalog migration will be dedicated to migrating tables from HMS to Unity Catalog. The migration pattern from HMS to Unity Catalog varies depending on the type of table storage used in HMS, such as external tables, managed tables, tables using the DBFS storage, mount points, Hive SerDe tables, and tables with various file formats, including CSV, Parquet, Delta, ORC, and so on.

Definition: Hive SerDe

SerDe is a short form representing serializer/deserializer. Hive uses a SerDe to read and write table rows in a specific file format. Besides the built-in SerDes for Avro, ORC, RegEx, Thrift, Parquet, CSV, and JsonSerDe, Hive also supports custom SerDes, which you can write to work with custom file formats.

Migrating external tables in Delta or Parquet file formats is straightforward because they seamlessly integrate with Unity Catalog due to their compatibility. In contrast, Hive SerDe tables are not supported in Unity Catalog and typically require conversion to Delta format. One important thing to note is that mount points that are shortcuts to cloud storage are not supported. This is because anyone in the workspace can access the data stored there. To use these storage locations safely, you need to register them as external locations first. This ensures that your data remains secure and is accessible to only authorized users.

The table migration process involves five key steps:

1. Register storage credentials. Register a credential that grants read and write access to the storage layer of the table being migrated. This is essential for migrating external tables from HMS to Unity Catalog as external tables.
2. Link external locations. Register the external locations of HMS tables in Unity Catalog using the storage credential with read and write access.
3. Create catalogs. Set up catalogs in Unity Catalog based on the desired data organization structure.
4. Define schemas. Create schemas within catalogs to ensure data isolation and grouping according to specific requirements.
5. Migrate tables and data. To complete the migration process, transfer tables and view definitions, and data for managed tables.

UCX has workflows that help you migrate HMS tables and views to Unity Catalog. The documentation available on UCX GitHub covers more information.

To properly migrate your assets from HMS to Unity Catalog, you must understand how they are set up. UCX distributes these assets under the following tags:

`DBFS_ROOT_DELTA`
: These are tables created at the root location of the Databricks workspace filesystem in the Delta file format. Unity Catalog cannot work with tables using DBFS as the storage layer. The best way to migrate tables onto Unity Catalog is to migrate them to another storage layer as a managed table. This is best achieved using the *deep clone* command in Delta. Deep clone is a robust feature that enables users to create an exact copy of a Delta table, including its data and

metadata, excluding the table history and schema. This powerful replication tool allows users to duplicate tables, making it ideal for testing, development, and data backup purposes.

`DBFS_ROOT_NON_DELTA`
: Deep clone is not supported for tables stored in the DBFS root location with non-Delta file formats. As a result, deep clone is not a viable option for migrating these tables. To migrate these tables, you can use the following command, which is essentially a `CREATE TABLE AS SELECT` (CTAS) statement that creates a new table based on a `SELECT` query on another table:

```
CREATE TABLE uc_table_name AS
SELECT
 *
FROM
 hms_table_name
```

This command creates a new table in Unity Catalog with the same data as the original table in HMS. The resulting Unity Catalog table will be stored in the Delta file format, ensuring optimal performance and reliability. While it might complicate the process, this also presents an opportunity to reformat the layout of some tables.

`MANAGED`
: Managed tables in HMS can be migrated to Unity Catalog using multiple approaches. The most straightforward method involves creating a new table in Unity Catalog by cloning the HMS table using the `CTAS` command.

This command creates a new table in Unity Catalog with the same data as the original HMS table. For managed HMS tables stored outside of the DBFS root location, the SYNC command can synchronize the table with an external Unity Catalog table. The `SYNC` command is an incremental data synchronization tool that re-creates tables in HMS and puts them onto Unity Catalog. It is incremental and can create new tables in Unity Catalog or update them based on changes in the source table in HMS. The `DRY RUN` option tests compatibility before actual migration. It checks if your HMS tables will work in Unity Catalog without moving data. This prevents migration failures and saves time during HMS to Unity Catalog transitions. The following command syncs an existing managed HMS table with a Unity Catalog external table:

```
SYNC TABLE main.default.uc_table AS EXTERNAL
FROM
 hive_metastore.default.hms_table
```

When using the SYNC command, be aware that if any user unintentionally drops the managed HMS table, the underlying data will also be deleted, affecting the synchronized Unity Catalog table. To prevent data loss in Unity Catalog when a managed HMS table is dropped, you can convert the HMS table to an external table in place and then use SYNC to synchronize between Unity Catalog and HMS. This approach ensures that the data remains intact in Unity Catalog even if the managed HMS table is dropped.

`EXTERNAL_SYNC`
: The SYNC command supports synchronizing external tables in Unity Catalog with various file formats, including:

  Delta
  Parquet
  CSV
  JSON
  ORC
  Text
  Avro

  These formats are compatible with the SYNC command, allowing seamless synchronization between Unity Catalog and HMS.

`EXTERNAL_NO_SYNC`
: Use the CTAS command in Unity Catalog to re-create file formats unsupported by the SYNC command, including those requiring conversion or those incompatible with Unity Catalog. This ensures that the data is appropriately formatted and accessible in Unity Catalog.

`EXTERNAL_HIVESERDE`
: The SYNC command does not support external tables in HMS that use Hive SerDe formats. To migrate these tables to Unity Catalog, you must re-create them using the CTAS command, which allows for proper data conversion and formatting.

  The `[UCX] migrate-tables` workflow automates the migration of tables from HMS to Unity Catalog, making it fast and easy. Using this workflow, you can streamline your migration process and quickly get your tables up and running in Unity Catalog. Leverage this powerful tool to save time and effort and take the hassle out of table migration.

Reconciliation

It's essential to validate your migration process, especially when undertaking complex enterprise-level Unity Catalog migrations. UCX deploys the `[UCX] migrate-data-reconciliation` workflow that exactly does that. The process validates the integrity of your migrated tables in Unity Catalog and generates the results in a table `recon_results`. It compares the following:

- Schema (`schema_matches`)
- Columns (`column_comparison`)
- Row counts (`data_matches`, within 5% threshold)
- Rows (hash comparison, if `compare_rows` flag is set)

Results are displayed in the migration dashboard via the `reconciliation_results` view.

Code migration

The introduction of catalogs in Unity Catalog requires updates to your existing code, specifically where tables are referenced in code using the old two-level namespace. While setting a default catalog can help, it's not a foolproof solution, especially when working across catalogs. Additionally, outdated usage patterns like DBFS-based access or mount points must be updated in every reference. Manually fixing these changes can be tedious, but UCX offers an automated solution to make your code Unity Catalog-compliant.

UCX breaks down the migration process into four parts:

`lint-local-code`
: Get a quick snapshot of migration requirements with this command, which detects dependencies and highlights necessary changes:

```
databricks labs ucx lint-local-code
```

`migrate-local-code`
: Convert your SQL and Python files to be Unity Catalog-compliant with this command (note: this is experimental, so verify and test changes manually):

```
databricks labs ucx migrate-local-code
```

`migrate-dbsql-dashboards`
: Easily migrate your dashboards to be Unity Catalog-compliant:

```
databricks labs ucx migrate-dbsql-dashboards
 [--dashboard-id <dashboard-id>]
```

`revert-dbsql-dashboards`
: Revert dashboard changes if you encounter any issues:

```
databricks labs ucx revert-dbsql-dashboards
 [--dashboard-id <dashboard-id>]
```

While Unity Catalog brought exciting new features, existing users still needed to migrate to unlock their full potential. UCX stepped in to ease migration concerns, helping you adopt Unity Catalog quickly and accelerate your data governance journey. With UCX, you can harness the power of Unity Catalog and take your data governance to the next level.

HMS Federation

Many enterprises have built their data platforms around multiple tools that rely on HMS as their catalog. Migrating these to Unity Catalog can be daunting, but HMS federation offers a clever solution. By federating external HMS instances, you can quickly tap into Unity Catalog's governance capabilities without disrupting your existing setup. This enables a flexible, hybrid approach, where you can leverage Unity Catalog's benefits for some workloads while keeping others in HMS—all accessible from your Unity Catalog-enabled workspaces.

HMS federation shines in scenarios where you must bridge the gap between HMS and Unity Catalog:

Interoperability between HMS and Unity Catalog
: If you have invested in multiple tools that rely on HMS, federating those tools to Unity Catalog enables effortless interoperability. For example, suppose you are using AWS Glue metastore or an external HMS in your Databricks workspace. In that case, HMS federation allows you to access and govern AWS Glue tables in Unity Catalog while leveraging external processes like Elastic MapReduce (EMR)-based ETL to hydrate your tables external to Unity Catalog. This hybrid approach unlocks a unified governance experience across your data platform.

Getting started with Unity Catalog quickly
: Need to get started with Unity Catalog quickly? HMS federation is the answer. If you're building new solutions that rely on HMS-based data, you can rapidly make them available in Unity Catalog and build out new solutions entirely in Unity Catalog. With no immediate need to migrate existing HMS assets, you can incrementally plan your migration, unblocking progress and allowing you to experience Unity Catalog's features on top of your HMS data immediately.

Internal Databricks HMS to Unity Catalog
: The best part? HMS federation can instantly connect your existing HMS in your workspace to Unity Catalog; no migration is required! Your HMS instance is now directly accessible from Unity Catalog as a catalog, allowing seamless access to your data without extra effort.

Assist in migration
: HMS federation can also help you migrate your data from HMS to Unity Catalog. By federating your HMS instance, you can run your new workloads against Unity Catalog, while existing workloads can be pointed to run against the federated catalog. Meanwhile, you can incrementally re-create the tables in Unity Catalog using UCX. Once the migration is complete, you can effortlessly redirect end users to the new native tables in Unity Catalog, ensuring a smooth transition.

Supported HMS Variants

The HMS federation capability can be used across different HMS variants, including the internal Databricks workspace HMS, an external Databricks HMS managed by the users, and an AWS Glue-based HMS. These variants differ in what capabilities are supported.

Internal Databricks HMS

As of this writing, every new Databricks workspace has an internal HMS and Unity Catalog. By federating the bundled HMS to Unity Catalog, you can access your data via Unity Catalog-enabled compute, unlocking powerful features like lineage, access controls, and more. This enables you to leverage Unity Catalog's capabilities while incrementally planning your migration, making it easier to transition to Unity Catalog at your own pace. With federated internal HMS, you get read access and write capability, allowing Unity Catalog compute to update data in your federated HMS tables. The federation ensures that updates go both ways. Even if a workload creates a new table in Unity Catalog, it is automatically reflected to the federated HMS, keeping everything in perfect sync.

DDL operations (`CREATE`, `ALTER`, `DROP`) and DML operations (`INSERT`, `UPDATE`, `DELETE`) are instantly reflected in HMS, ensuring your data remains consistent and up to date across Unity Catalog and HMS. This seamless bidirectional sync enables downstream systems to rely on HMS to access the latest data, maintaining an uninterrupted workflow.

External HMS

Databricks allows you to federate external HMS, including user-managed HMS and AWS Glue metastores, to Unity Catalog. While these external HMS work similarly to internal HMS, note that write capability is not supported as of this writing. However, any permissions you apply to these federated external HMS are enforced across Unity Catalog compute, including SQL warehouses and interactive clusters. To federate external HMS, ensure your external metastore is compatible with Apache Hive Versions 0.13, 2.3, or 3.1.

Note that Hive SerDe tables are supported via federation, even though they're not a natively supported format in Unity Catalog. Additionally, shallow cloned tables, managed tables, external tables, and views are supported. However, some features are unsupported, including the following:

- Hive functions
- UDFs
- JDBC-backed tables
- Delta Sharing of foreign catalogs
- Lakehouse monitoring
- Vector search
- Online tables

How to Federate HMS

Federating an external HMS to Unity Catalog is typically managed by a metastore admin. However, workspace administrators can also perform this task if they've been delegated the necessary privileges, including the following:

- Create catalog
- Create connection
- Create an external location
- Create a storage credential

The federation process for HMS is similar for internal, external, and AWS Glue; the main difference is the authentication mechanism used. Glue catalogs use an IAM role for authentication and are supported only in Databricks on AWS, while an external HMS requires credential-based authentication.

Create connection

HMS federation is initiated by registering a connection to the external metastore, which a metastore administrator can do through various methods, including these:

- UI
- Databricks CLI
- REST API
- SQL command

This connection sets the stage for federating the external metastore with Unity Catalog, enabling seamless access and management of data across both platforms:

```
CREATE CONNECTION < connection - name > TYPE hive_metastore OPTIONS (
 host '<hostname>',
 port '<port>',
 user secret (
```

```
    '<secret-scope>', '<secret-key-user>'
  ),
  password secret (
    '<secret-scope>', '<secret-key-password>'
  ),
  database '<database-name>',
  db_type 'MYSQL',
  version '2.3'
);
```

Definition: Secret

Databricks secrets are sensitive information, such as passwords, API keys, or other confidential data, that are stored securely within a Databricks workspace. They can be stored directly within Databricks or referenced from external secret management systems like Azure Key Vault, allowing users to manage and rotate sensitive information in a centralized and secure manner. By using secrets, users can reference sensitive information in their code and notebooks without directly exposing sensitive data.

Register locations

After connecting to HMS, register the storage locations of HMS tables as external locations in Unity Catalog using a storage credential with read and write permissions. Since Unity Catalog manages only registered locations, all paths used by HMS tables must be registered in Unity Catalog to ensure access and governance. These registered paths must cover all tables you want to access through the HMS federation.

Create a foreign catalog

After registering the storage locations, the metastore administrator can create a foreign catalog using the HMS connection, specifying the registered locations as authorized paths. This provides an additional layer of security, allowing catalog owners to control user access to federated data and prevent unauthorized changes. By setting authorized paths, you can maintain data access and security, even when HMS allows users to update metadata and alter table locations. The create catalog screen in the UI, as shown in Figure 11-6, requires you to specify the necessary attributes, including the authorized path.

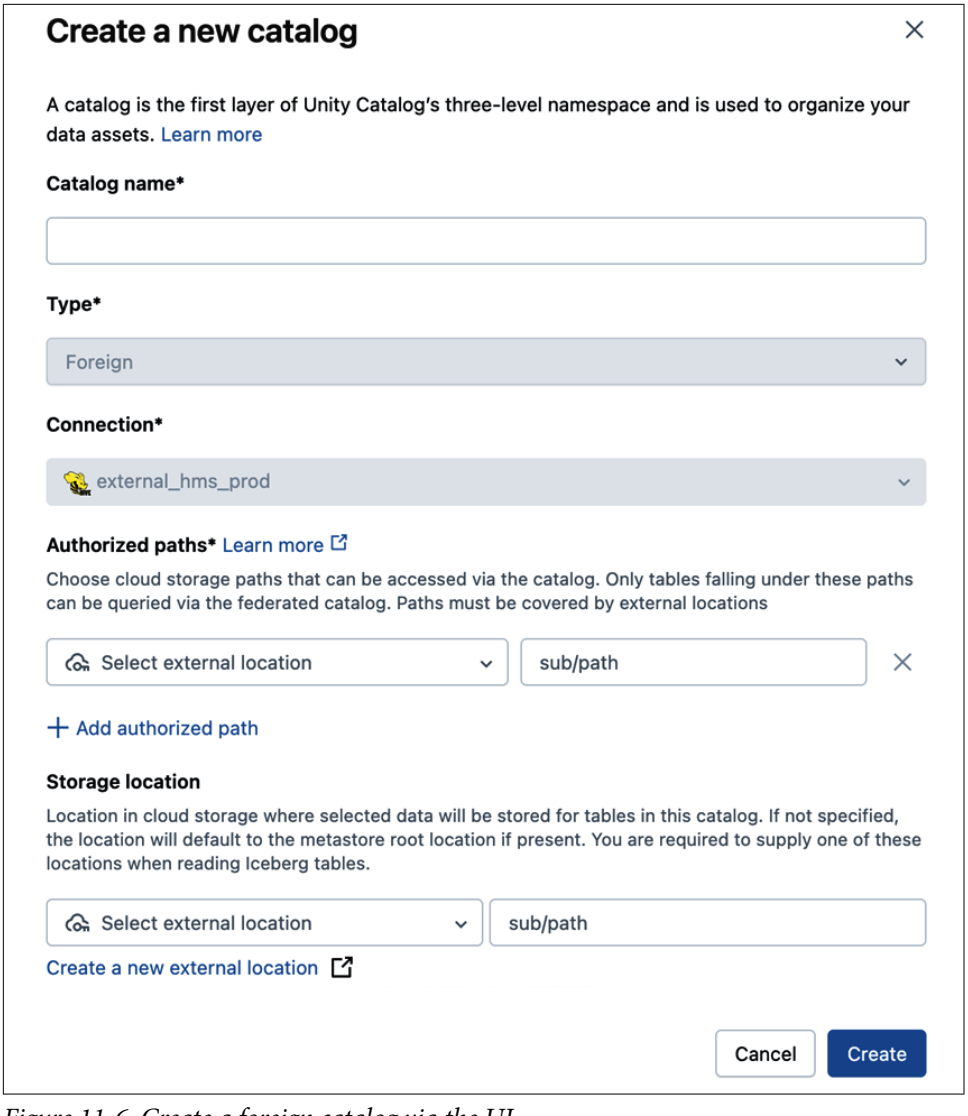


Figure 11-6. Create a foreign catalog via the UI

After registering a foreign catalog in Unity Catalog, metadata is synchronized between HMS and Unity Catalog, ensuring the data remains up to date. As processes access tables in the foreign catalog, metadata is continuously updated to reflect the latest changes, providing a unified and accurate view of your data across both platforms, as illustrated in Figure 11-7.

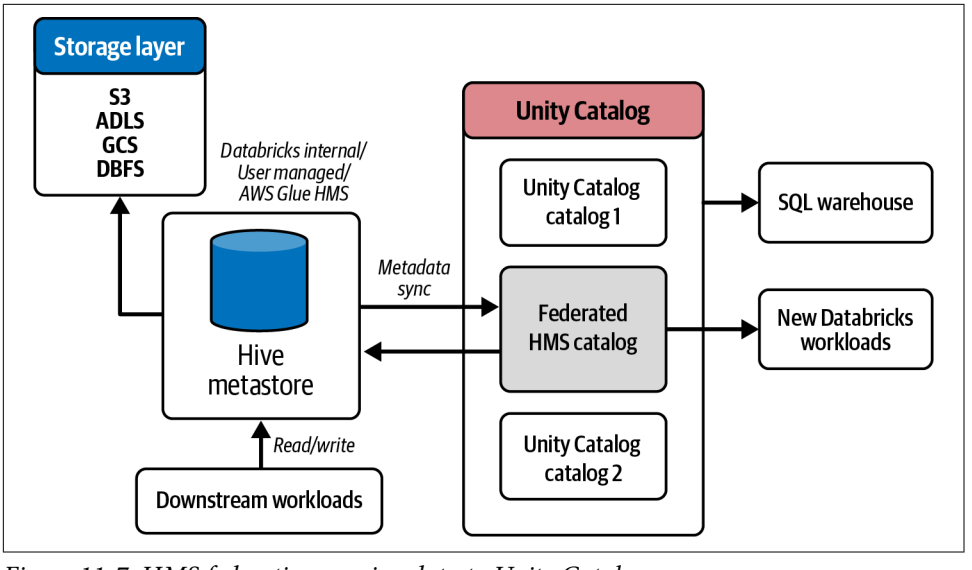


Figure 11-7. HMS federation syncing data to Unity Catalog

Fallback mode

HMS federation, particularly internal Databricks HMS to Unity Catalog, facilitates a gradual migration to Unity Catalog. However, Unity Catalog access controls can challenge existing workloads against the federated foreign catalog. When federating HMS to Unity Catalog, you define authorized paths that Unity Catalog manages, and users must obtain administrative permission to access these paths. However, this can disrupt workloads that rely on HMS data via the federated foreign catalog. To address this, Fallback mode can be enabled for external locations, allowing temporary access to files using credentials not managed by Unity Catalog, such as instance profiles or SP keys to access data by path. Use Fallback mode as a temporary solution during the transition to Unity Catalog and turn it off once all workloads have switched to using Unity Catalog.

Nexa adopted a phased approach to migrate from HMS to Unity Catalog. The first step was to prioritize the components to be migrated. The CDP team used the UCX assessment to evaluate the complexity of the migration and internally assessed the business criticality of each element to determine the migration order. They started with less critical solutions and, after gaining confidence, gradually moved on to more critical ones. As an early Unity Catalog adopter, Nexa had to develop custom migration scripts due to the unavailability of automated tools like UCX.

To ensure a smooth migration, they created a detailed plan with checklists that outlined the necessary steps for a successful migration. This plan was reviewed at each stage to ensure that all requirements were met. The company's successful migration

was also facilitated by rigorous testing across all teams, which helped identify and resolve potential issues early on. The early involvement of all teams and stakeholders was also crucial, as it ensured that everyone was informed and aligned with the changes. Nexa identified limitations in Unity Catalog clusters and developed workarounds for unsupported features or deferred migrations when necessary. With the introduction of UCX, subsequent migrations became smoother, and the knowledge gained from previous migrations enabled the company to migrate more efficiently. After experiencing the capabilities of Unity Catalog, Nexa found it to be a significant improvement, making it an indispensable tool.

Summary

Consider your current stage in the Databricks data governance journey. New users are all set to start with Unity Catalog as their default catalog, while existing users can migrate to Unity Catalog using the UCX tool or use HMS federation to access Unity Catalog features quickly without a full migration. Choose the path that suits you best and embark on a more governed and secure data journey with Unity Catalog. The final chapter, Chapter 12, explores what lies ahead for Unity Catalog. We examine newly announced features that will reshape how you manage data governance.

CHAPTER 12
The Future of Unity Catalog

The future cannot be predicted, but futures can be invented.
—Dennis Gabor

Jamie Dimon, Chairman and CEO of JPMorgan Chase, was a guest at the Data + AI Summit 2025 held in San Francisco. JPMorgan Chase is the largest bank in the US. It was really interesting to listen to Jamie's views on AI and cybersecurity. He mentioned that JPMorgan Chase is spending around one billion dollars a year on cybersecurity and two billion dollars a year on AI, and these numbers are expected to keep growing. This speaks volumes about how much importance is being given to these topics.

Similarly, more and more companies seem to be willing to invest in the secure use of AI. Moreover, this investment is not just for engineers and scientists, but for everyone within the organization. Data democratization applies to all, not just the ones who are technically savvy. This means more users with a nontechnical background should be given access to the tools so that they can also leverage data and AI. This requires organizations to develop user-friendly interfaces to such tools, ensuring that access and usage are secure. As the number of users increases, security and governance become more complex.

One way to achieve the democratization of data and AI is to onboard nontechnical users, especially business users, to the data and AI platform. Instead of bringing the data and AI to the users, bring the users to the data and AI. This avoids organizations having to develop a new set of tools and techniques. But the platform should be capable of catering to these business users, including data-visualization capabilities, AI tools to interact with the data, a simple interface, easier and limited access to only the relevant platform features, and so on. In addition, everything should be secure and governed. Is Databricks with Unity Catalog such a platform?

Since its inception, Unity Catalog has evolved into a trusted, enterprise-ready solution that you can rely upon for operational governance. However, most of the features and interfaces cater to technical users. Recently, Databricks introduced features such as AI/BI dashboards and Genie spaces, which seemed like a good way for business users to interact with the data, but the platform itself is still too complex. A user who wants to use an AI/BI dashboard or a Genie space needs to be added to the Databricks account and workspace. Until recently, users needed to be given at least the "workspace access" entitlement, which includes the data engineering and ML sections of the platform, which, for a business user, is often irrelevant. Essentially, the business user journey to access a dashboard or an AI chat interface on Databricks is/was too complex.

Nevertheless, the latest and upcoming features add a new dimension to Unity Catalog and Databricks in general. The future of Unity Catalog seems to be headed toward being more open, interoperable, and inclusive of nontechnical users. This final chapter briefly discusses some of these latest and upcoming features and explains how they will help organizations to democratize access to data and AI. The chapter also touches upon some of the advanced data governance capabilities and interoperability aspects.

Advanced Data Governance

In Chapter 5, we covered access controls in great detail and discussed some of the recently introduced advanced access controls, such as governed tags and ABAC. We believe ABAC is one of the features that will be further enhanced and enriched with new types of policies, such as `GRANT` and `DENY`, and attributes, such as workspace details, location, and time. (For more details, see this video.) Apart from ABAC, governed tags will play a crucial role in some of the upcoming governance-related features. One of them is *data classification*, which automatically detects and tags sensitive data across Unity Catalog. Another one is *certification and deprecation* tags, which can be leveraged to signal quality, status, trustworthiness, and business relevance across datasets, metrics, and dashboards.

Another upcoming advanced access control feature that will be extremely useful is RBAC. We briefly mentioned the RBAC concept in Chapter 5 and how you can simulate RBAC behavior with role-specific groups. As required by certain industries, especially highly regulated ones, it's great to see that Databricks is working on providing RBAC as part of the platform. It is being implemented to function as follows:

- Define *exclusive groups,* a new type of group created directly in Databricks. These groups act as roles.
- Grant these exclusive groups access to Unity Catalog objects, such as catalogs or tables, and workspace resources, such as notebooks.

- Enable identity provider-backed users and groups to assume exclusive groups for specific activities.

This approach enables organizations to implement complex governance requirements. For example, a pharmaceutical company typically runs multiple clinical trials in parallel. When a user is involved in multiple trials, they need to be able to access and work on trial-specific data assets at a given time, without mixing things up across the trials. With Databricks RBAC, the user can simply assume a trial-specific exclusive group while working on the corresponding trial. Once the job is done, they can switch to another trial-specific exclusive group. (For more details on the RBAC feature, see this video.)

We believe Databricks ABAC and RBAC will further enhance organizations' ability to implement complex governance requirements on the Databricks Platform and simplify the regulatory compliance journey.

Apart from Databricks releasing more advanced features, we are also excited to see what Databricks customers/users build by leveraging the existing feature set. The introduction of Databricks Apps and, more recently, Lakebase, a fully managed Postgres designed to be the OLTP for the lakehouse, has unleashed a wellspring of creativity. Databricks Apps, Lakebase, and the AI ecosystem, with secured access to data and AI assets in a fully governed environment, is a perfect setup for developing immensely useful applications. We anticipate seeing a plethora of custom apps being developed on Databricks that will enhance the core platform's capabilities without waiting for Databricks to provide it. Some governance features that could be built using Databricks Apps and Lakebase include ontology modeling, entity resolution, and more.

Catering to Business Users

As a business user, you're typically interested in getting value from data, which is usually through visualization, such as BI dashboards or applications such as AI-powered chatbots. You typically don't care much about what the ETL processes look like or the precision of the ML models. In terms of access controls, you don't need direct access to the data that fuels the dashboards or the AI model that powers the applications. Nevertheless, you do want to know whether you can trust the numbers that are shown on the dashboards and the information provided by the AI applications. In addition, you want to be able to rely on a commonly agreed-upon definition for your KPIs. If everyone has their own definition and implementation of common KPIs, the end results will differ, and it's hard to make the right decisions based on inconsistent insights. Therefore, data quality and business metric consistency are crucial for data-driven decision making.

Nexa Boutique's successful adoption of Databricks Unity Catalog resulted in improved data security and governance, along with increased productivity of the data teams across the globe. The number of active Databricks users has grown significantly over the years. Nexa had also started onboarding business users to Databricks so they can use AI/BI dashboards, Genie spaces, and Databricks Apps. However, Nexa had difficulty limiting access, as these new users would have access to all the features in the Databricks workspace, which did not sit well with the company's principle of least privilege. Moreover, these new users were overwhelmed by the Databricks UI, which discouraged them. On the other hand, the introduction of Unity Catalog metrics views (discussed in "Unity Catalog Metrics") was helping Nexa deal with one of its biggest challenges in deriving insights from data: inconsistent definitions of its business KPIs.

Nexa had all its data relevant for analytics already on Databricks, had almost all of its AI workloads on Databricks, had successfully adopted AI/BI dashboards, and most people within the organization found Genie spaces to be extremely valuable. Now with Unity Catalog metric views, it made logical sense to onboard the business users to Databricks. However, given the challenges mentioned previously, Nexa was forced to re-evaluate the decision to onboard business users to Databricks and was considering building an external portal where they would embed AI/BI dashboards and Genie spaces. However, after attending the Data +AI Summit 2025, Nexa decided to go ahead and onboard business users. Let's look at why.

Unity Catalog Metrics

As a retailer, one of Nexa Boutique's most important KPIs is sales revenue, a business metric that measures the total income generated from sales of goods and services during a specific period of time. However, when this metric is calculated and presented through BI dashboards, surprisingly, there's no uniform definition of what sales revenue *is*. In some reports, while computing sales revenue, returns and discounts are considered; in other reports, they are not. In some executive reports, sales revenue is net sales revenue, where both the returns and discounts are subtracted from the gross sales revenue.

Nexa has been trying to unify the definitions for such important metrics so that there would be no confusion and management could rely on the numbers they see on the reports. However, it has been challenging, to say the least. The main challenge is that there is no central place where these KPIs are defined, to which the data and business teams could refer. The second challenge is that even if the company were to succeed in centrally defining these KPIs, making sure that they are implemented in the correct way in every single report and AI application is impractical.

Unity Catalog metric views, which Nexa tried out as a private preview feature, addressed these challenges. Using this feature, Nexa could define the KPIs and

register them in Unity Catalog. Unity Catalog acts as the central place where all the important KPI definitions live. Moreover, the dashboards and AI applications can now refer to the same definition of the KPIs. In other words, there's no need for a custom implementation of the KPI in the dashboards and applications, hence no scope for errors. Figure 12-1 depicts the define once, use everywhere approach of Unity Catalog metrics.

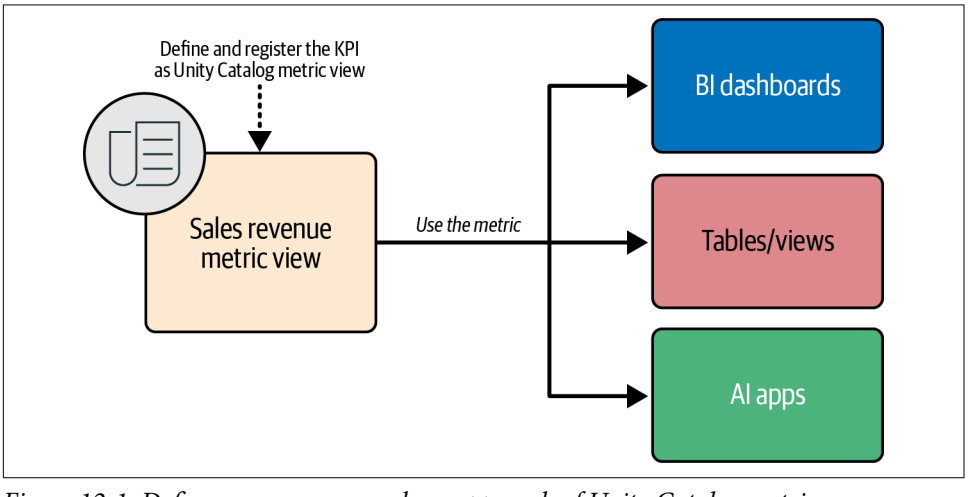


Figure 12-1. Define once, use everywhere approach of Unity Catalog metrics

Apart from reliability and consistency, using Unity Catalog metric views could also improve performance. These performance improvements are an upcoming feature. (For more details on this topic, check out this video.)

Business User-Friendly Interface

How do you make a data platform appeal to business users? The answer is simple: make it easier for them to derive value from data without any distractions. This means a simple, intuitive, consistent UI with a focus on what is relevant for business users. For example, for the Databricks workspace, the first step is to minimize options and show only what's relevant. To achieve this, Databricks has introduced a new entitlement type, *consumer access*, for the Databricks workspace. A user with only this entitlement can do the following:

- Read/run dashboards, Genie spaces, and Databricks Apps
- Query SQL warehouses using BI tools

And that's pretty much it. This removes all the clutter and allows you to enforce the principle of least privilege even when you grant access to business users within your organization.

In terms of UI, there are two upcoming features: the enhanced discovery experience and Databricks One. The new enhanced discovery experience is introduced as a new tab in the Databricks workspace, which shows the relevant assets to the users. The assets are grouped into domains, and the platform's Data Intelligence Engine determines their relevance to users based on their popularity, quality, and trustworthiness.

Databricks One is intended to be a new Databricks experience designed specifically for business users. According to this public announcement, Databricks One users will have only the consumer entitlement to the Databricks workspace. They can access specific AI/BI dashboards, Genie spaces, and Databricks Apps and browse content by business domains.

Doubling Down on Openness and Interoperability

Most Databricks users we speak to typically have more than one data platform within their organization. Recently, we've seen a trend that more of them want to centralize and unify the metadata layer. In other words, they want a single catalog for metadata management while the data can reside on any platform. Databricks Unity Catalog seems to be best suited to be this unified metadata layer because it supports multiple table formats, namely Delta and Iceberg, as well as being the most interoperable catalog. Chapter 9 described how data cataloged in Unity Catalog can be accessed by external platforms and compute engines.

As the list of features supported in the open source Unity Catalog expands and Databricks Unity Catalog becomes more interoperable with the wider data ecosystem, the future seems to be heading toward more openness. A strong indication of this comes from the recent announcement of full support for the Apache Iceberg format. But what does it mean to fully support Apache Iceberg? To use Apache Iceberg format, you need a *catalog*, a foundational component of Apache Iceberg that enables it to deliver transactional guarantees, schema evolution, and multiengine interoperability.

Until now, solutions such as HMS and AWS Glue have been extensively used to serve as a catalog for Apache Iceberg. However, starting with Version 0.14.0, Iceberg introduced the REST open API specification, enabling clients to communicate with any Apache Iceberg catalog over HTTP. This standard defines how essential catalog operations—like listing, creating, updating, and deleting tables and namespaces—are handled via RESTful endpoints, independent of the catalog's backend technology or programming language. As a result, REST-compliant catalogs simplify the management of Iceberg tables across various environments, enhancing interoperability,

governance, and advanced data lakehouse capabilities. In short, you're better off using a catalog that supports the Apache Iceberg REST API specification.

Databricks Unity Catalog fully supports the Apache Iceberg REST API specification and is, therefore, suitable for use as a catalog for Iceberg tables. External engines can read and write to managed Apache Iceberg tables cataloged in Databricks Unity Catalog. Moreover, Unity Catalog-managed Iceberg tables benefit from the platform's optimization capabilities, such as predictive optimization and automatic liquid clustering, resulting in a better price:performance ratio.

In essence, Unity Catalog makes it much easier for external engines and platforms to interact and integrate with it by providing reliable interfaces. We believe this will continue, and Unity Catalog will end up being an obvious choice as a centralized metadata layer across compute engines, platforms, and clouds.

Nexa Boutique was once again excited about all the latest features announced by Databricks, some of which will address the challenges discussed earlier. Nexa's plan to open up the Databricks Platform to business users is going ahead, and the company is looking forward to continuing to reap the benefits of the time and effort it has put into adopting Unity Catalog and following best practices. In retrospect, it was all worth it.

Summary

The journey of Unity Catalog that began as a substitute for HMS and evolved into a governance solution has been adventurous, and there are many things we couldn't capture fully in this book. Some desired features remain as future roadmap items; others, as private preview features as we write this. Personally, we're anticipating many exciting developments that will come to Unity Catalog. The ecosystem is evolving rapidly, and we watch these spaces closely. But the foundation covered in this book should remain the same. You can align current implementations with future capabilities. This reduces technical debt later. As Databricks Unity Catalog matures into a full-fledged governance solution and the open source Unity Catalog gets more traction and adoption, who knows—there might be another edition of this book.

Remember that Unity Catalog marks a significant shift in data governance for the Databricks Data Intelligence Platform. We hope you enjoyed reading this book, and we wish you luck with your journey to implement data and AI governance with the Databricks Data Intelligence Platform.

Index

Symbols

A

B

C

D

E

F

G

H

M

N

Q

R

S

T

U

V

W

Z

About the Authors

Kiran Sreekumar is a data architect with deep expertise in engineering and data and AI governance. Since 2012, he has been designing and building data engineering, data warehousing, and data governance solutions, accumulating more than 15 years of experience across a variety of industries. While at Tata Consultancy Services (2012–2015)—a period when Hadoop and big data were just taking off—Kiran helped create one of the first data archival and governance products on the Apache Hadoop platform, Active Archive. Following that, he contributed to numerous large-scale data analytics programs. Today, he works at Databricks as a specialist in data and AI governance and engineering, where he partners with customers to embed robust governance practices into modern data pipelines. He is an active speaker on data and AI topics at conferences and a passionate advocate for trustworthy data stewardship. Kiran holds a bachelor of technology degree in electronics and communications and is based in the United Kingdom.

Karthik Subbarao is a specialist in data + AI platform and governance at Databricks, helping organizations build secure, scalable data and AI platforms. With over a decade of experience in software engineering, architecture, big data, and AI, he combines deep technical expertise with a pragmatic approach to solving complex challenges. Karthik holds an MSc degree in computer science from the Technical University of Munich and regularly shares insights through blogs, conference talks, and community events. Based in Germany, Karthik is passionate about building secure, trusted data platforms for the AI era.

Colophon

The animal on the cover of *Data Governance with Unity Catalog on Databricks* is a chevrotain *(Tragulus kanchil)*, or lesser mouse-deer.

The chevrotain is a small, elusive ungulate. Despite its name, it is neither a true deer nor a rodent. Native to parts of Southeast Asia and West-Central Africa, chevrotains favor dense tropical forests near water sources such as streams and wetlands, where they can forage and hide from predators. They are the smallest hoofed mammals in the world, with some species—like the lesser Malay chevrotain—weighing as little as two pounds.

Chevrotains have a compact, rounded body with thin legs, a short tail, and no antlers. Males of some species possess small, sharp canines used for territorial defense. These nocturnal, solitary creatures feed on fruits, leaves, flowers, and other plant matter, occasionally consuming insects or small animals. One of their most fascinating behaviors is their ability to "freeze" for long periods when threatened, often followed by a zigzag sprint into cover. Females give birth to a single offspring, which is often hidden in dense vegetation for protection. Lifespan in the wild is estimated at 10–12 years.

Although some species of chevrotain are considered relatively stable, others—such as the Vietnam mouse-deer (*Tragulus versicolor*)—are considered a species of conservation concern due to deforestation and hunting. The rediscovery of this species in 2019 after being "lost" for nearly 30 years has brought renewed conservation interest. Overall, habitat preservation is crucial to ensure the survival of these timid and unique mammals.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Monica Kamsvaag, based on a black-and-white engraving from *Lydekker's Royal Natural History*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.