

Auditing Your Data and Its Usage

By now it should be evident how Unity Catalog can assist organizations in securing data and AI in a unified manner at scale. However, implementing security measures does not guarantee immunity from unexpected events or security incidents. From a security standpoint, many organizations have embraced the “Never trust, always verify” principle by adopting a zero trust architecture (ZTA) model. The core tenet of the ZTA model emphasizes that even with robust identity verification, compliance validation before granting access and least privilege access controls, default trust should be avoided and continuous monitoring of system activities — including the Databricks Data Intelligence Platform — is essential. In this chapter, we’ll explain how this can be achieved using Unity Catalog system tables.

System tables

System tables function as a centralized operational data repository, supported by Delta Lake and supervised by Unity Catalog. They facilitate querying in multiple languages, enabling diverse applications across BI, AI and even generative AI. Increasingly, customers are utilizing system tables for various purposes, including usage analytics, consumption/cost forecasting, efficiency analysis, security and compliance audits, service level objective analytics and reporting, actionable DataOps, and data quality monitoring and reporting. This consolidation of information for auditing purposes significantly streamlines the tasks of security and compliance teams. Prior to the introduction of system tables in Unity Catalog, this data was fragmented across different systems, requiring an ETL process to prepare and render the data usable.

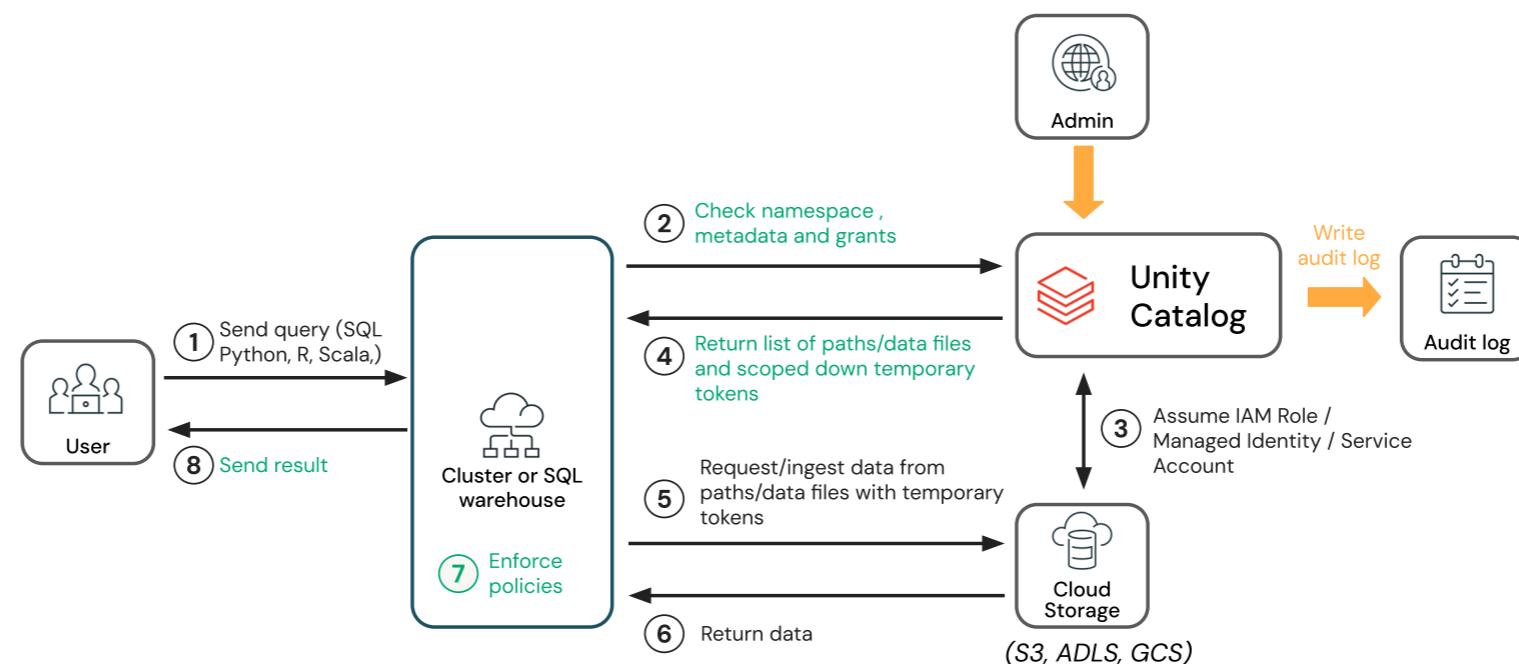
While multiple schemas are available, this chapter predominantly concentrates on the *audit* and *information schema* tables and their role in monitoring and auditing usage of operations conducted on data stored in Unity Catalog.

The audit and information schema system tables

When auditing data and its usage, the two most critical system tables to consider are `system.access.audit`, `system.information_schema` and all tables under `system.information_schema`.

The audit system table encompasses more than just data access-related information. It aggregates all audit logs at both the Databricks workspaces and account levels. Within the account-level logs, records of all significant data accesses facilitated by Unity Catalog are included. For further details, see the [official documentation](#).

The following figure illustrates the lifecycle of a query in the Databricks Data Intelligence Platform. It showcases how Unity Catalog mediates access to underlying data in storage accounts from any compute resource in Databricks, ensuring permissions are validated against each query. This process enables data availability to the Unity Catalog audit system table.



Examining the Unity Catalog-related information within the audit log reveals over 100 events, detailing access to various asset types, including structured (tables/views) and unstructured (volumes) data, as well as AI models.

On the other hand, the information schema table furnishes details about securables in Unity Catalog across all catalogs within the metastore. It offers insights into data at rest, with a particular emphasis on schemas, grants and lineage of various securables within this context.

Questions for audit logs

System administrators, data governance teams or data owners seek to comprehend how the data they oversee is accessed. They want to monitor access patterns and receive alerts if any unusual activity occurs, aligning with the ZTA model introduced earlier in the chapter. This approach enhances security by providing an additional layer of protection, thereby mitigating the risk of security incidents. Proactive monitoring can take various forms, such as real-time monitoring to respond promptly to potentially harmful events or periodic assessments to detect any anomalies.

The Databricks Data Intelligence Platform offers comprehensive tools for implementing an alerting system using data from system tables. Users can employ ETL processes to aggregate data, query it in multiple languages and configure alerts using internal alerting capabilities. The system is also equipped to take action, such as revoking permissions, if instructed, when a user accesses a resource they shouldn't have access to.

The questions that can be addressed using audit logs found in system tables are diverse and contingent upon the nature of the data within the data platform. The following list categorizes such questions into a nonexhaustive list of categories:

- Proactively identifying flaws in the permissions model
- Monitoring access and operations
- Tracking alterations to tables and schemas
- Detecting new and unexpected access patterns
- Monitoring data downloads and uploads

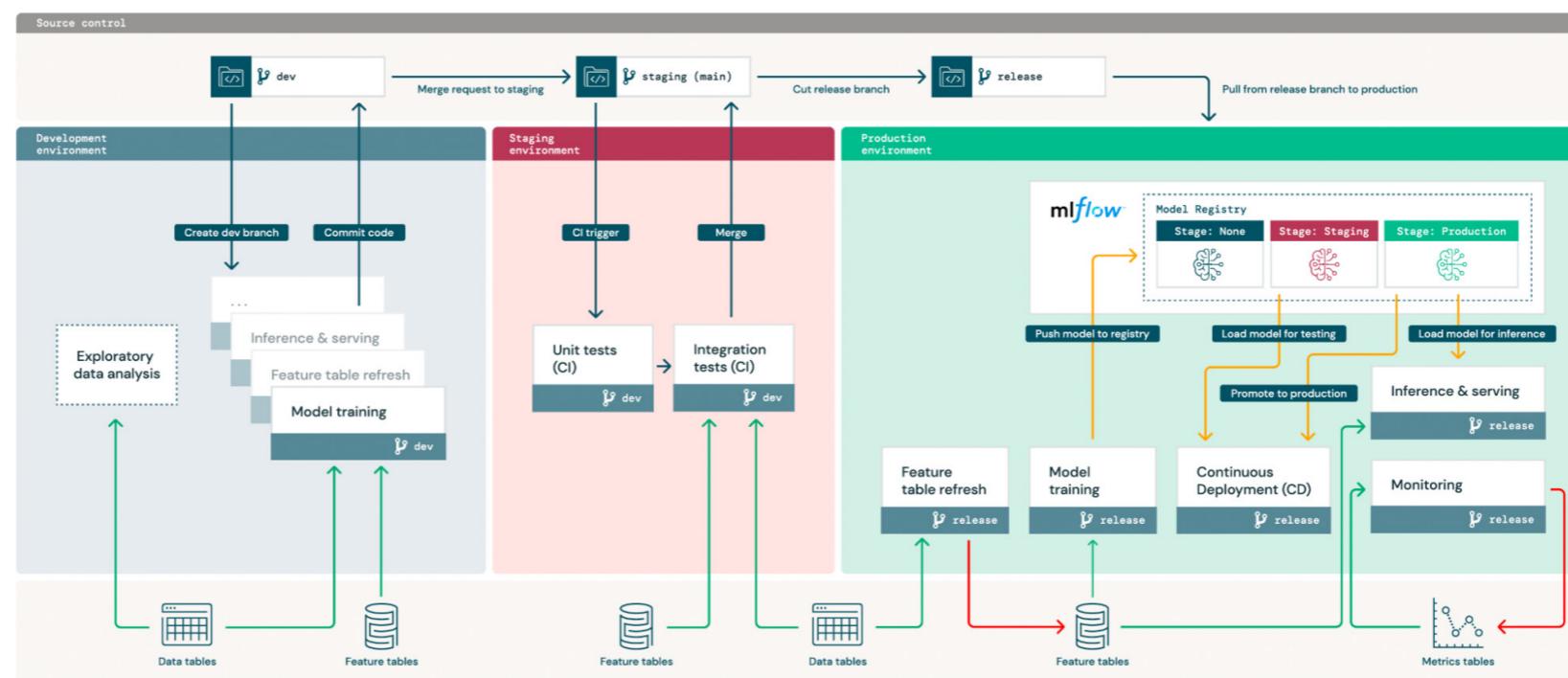
In summary, Unity Catalog system tables provide a powerful tool for organizations to enhance their data security and governance practices within the Databricks Data Intelligence Platform. By leveraging these tables, security teams can centralize and streamline their auditing processes, enabling them to gain comprehensive insights into data access and usage patterns. The integration of a zero trust architecture (ZTA) model further fortifies this approach, ensuring continuous monitoring and verification of all activities. As organizations continue to navigate the complexities of data security, the capabilities offered by Unity Catalog system tables will be instrumental in proactively identifying potential vulnerabilities, monitoring compliance and maintaining robust security standards at scale.

Governing Models and AI

Unity Catalog offers a suite of features that support generative AI models, enhancing their governance, deployment and overall lifecycle management. One of the key features is the Model Registry in Unity Catalog, which extends the benefits of Unity Catalog to ML models. This includes centralized access control, auditing, lineage and model discovery across workspaces. It also provides namespacing and governance for models, allowing you to group and govern models at the environment, project or team level.

As organizations look to scale their use of AI, a typical challenge has been bringing decision support systems based on AI to production. This is where MLOps and extensions of it like LLMOps play a critical role.

These frameworks, when properly implemented, meaningfully integrate data and AI operations, enhance collaboration and facilitate an efficient iteration loop that prioritizes *reproducibility, explainability and auditability*.



Model development, training and evaluation

In the “Tracking and Promoting Data Quality” chapter, we discussed the benefits of Unity Catalog lineage for data and model quality monitoring and tracking. This functionality is further enhanced by the use of MLflow, which allows you to keep track of the experiment and run that produced a model at a given time. You can use this information to understand the evolution of the model and its performance over time. Even more practically, this allows you to keep track of model versions and link them to specific data and code versions.

With model versioning, you can more easily compare how changes in parameters, data and code impact the overall performance of your models.

Unity Catalog unifies the management of modeling resources by giving development teams the option of having data, embeddings and models under a single schema. This extends the cataloging capabilities of data that is both structured (tables) and unstructured (volumes) to AI models.

This also aids in scalability and discoverability because as an organization’s data needs grow, so too does the number of AI assets. A well-architected Unity Catalog structure helps manage this growth, ensuring that the number of use cases can scale smoothly without becoming unmanageable.

The screenshot shows the Databricks Catalog Explorer interface. On the left, there is a sidebar titled "Catalog" with a search bar containing "dbdemos". Below the search bar, there is a tree view of catalogs and their contents:

- dbdemos
 - Tables (12)
 - Volumes (1)
 - Models (1)
 - dbdemos_customer_churn (selected)
 - dbdemos_retail_c360_mgu

On the right, the details for the selected "dbdemos_customer_churn" model are shown. The title is "dbdemos_customer_churn". There are tabs for "Overview" (selected), "Details", and "Permissions". The "Overview" tab shows a "Description" field with "Add description". Below that is a "Versions" section with a table:

Status	Version	Time registered	Tags	Aliases
✓	Version 1	2024-03-13 12:...	@ prod	(info)

Model Serving

Model Serving in Unity Catalog offers a multitude of benefits that streamline the deployment, governance and querying of AI models. It provides a unified interface that allows you to manage all models in one location and query them with a single API, regardless of whether they're hosted on Databricks or externally. This simplifies the process of experimenting with, customizing and deploying models in production across various clouds and providers.

Through native integration with the Databricks feature store and Databricks Vector Search, Model Serving also simplifies the integration of features and embeddings into models. This allows for improved accuracy and contextual understanding, as models can be enhanced with additional context — for example when leveraging a **RAG**-based approach or fine-tuned with proprietary data — and deployed effortlessly.

Furthermore, the Model Serving UI allows you to centrally manage all model endpoints, including those that are externally hosted. You can manage permissions, track and set usage limits and monitor the quality of all types of models. This democratizes access to SaaS and open LLMs within your organization while ensuring appropriate guardrails are in place.

Model deployment via aliases is another feature that enhances the flexibility of model management. If the default catalog for your workspace is configured to a catalog in Unity Catalog, models registered using MLflow APIs are registered to Unity Catalog by default.

Serving endpoints [Provide feedback ↗](#)

Foundation model APIs

 DBRX Instruct Chat · Pay-per-token	Preview
Query URL ⋮	

 Meta Llama 3 70B Instruct Chat · Pay-per-token	Preview
Query URL ⋮	

 Qu	
---	--

Filter serving endpoints by name

Owned by me

Name	State	Served entities	Tags	Task
dbdemos_endpoint...	 Ready	 main.rag_chatb... (v 1)		

Security, safety and permissions

Securing models, AI assets and overall AI safety is an emerging area of focus. Unity Catalog allows for setting permissions on the use and access to models so that only authorized teams can modify and deploy selected models.

Moreover, model access to data can also be enforced to follow data access permissions. For example, consider a model that uses RAG to answer questions about a given subject by using Confluence pages. With Unity Catalog, data access permissions are respected and enforced throughout the whole chain. So if a user queries a model, the model crafts its answer using only Confluence pages the user has access to.

Elements of Unity Catalog discussed earlier in this chapter, like lineage and auditability, further enhance AI safety and security by providing full transparency on who accessed the model, what was done and when it was done.

This transparency, together with a robust data monitoring, evaluation and alerting system, can be used to capture issues with model performance and implement the appropriate corrective actions.

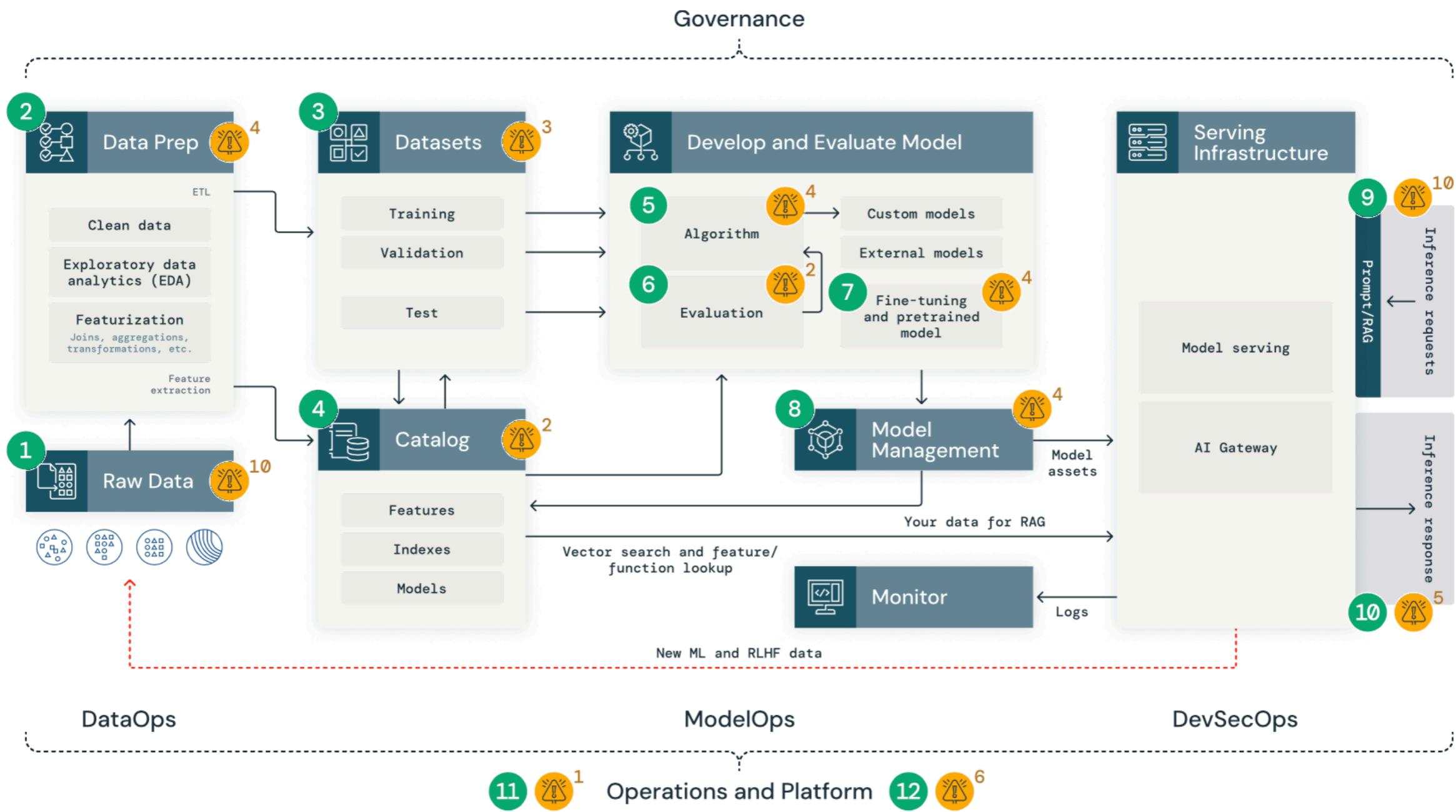


Figure: Foundational components of a generic data-centric AI system. Numbers in orange indicate risks identified in that specific system.

AI component number ! Number of risks

In summary, managing your models with the help of Unity Catalog provides the following benefits:

- **Centralized access control, auditing, lineage and model discovery** — Unity Catalog extends the benefits of centralized access control, auditing, lineage and model discovery across workspaces to ML models. It's compatible with the open source MLflow Python client.
- **Namespacing and governance for models** — You can group and govern models at the environment, project or team level. This allows for better control and organization of your models.
- **Chronological model lineage** — Unity Catalog tracks which MLflow experiment and run produced the model at a given time. This helps in understanding the evolution of the model over time.
- **Model Serving, versioning and deployment via aliases** — Unity Catalog supports Model Serving, versioning and deployment via aliases. For example, you can mark the "Champion" version of a model within your prod catalog. Models registered using MLflow APIs are registered to Unity Catalog by default.
- **Sharing models across workspaces** — To collaborate with other users on a registered model you created, you must grant ownership of the model to a group containing yourself and the users you'd like to collaborate with. Collaborators must also have the USE CATALOG and USE SCHEMA privileges on the catalog and schema containing the model.
- **Annotating models** — You can provide information about a model or model version by annotating it. This could include an overview of the problem or information about the methodology and algorithm used.

Sharing Your Data

In today's digital economy, seamlessly and securely sharing data and AI assets has become crucial for businesses that want to unlock the full potential of their data. The Databricks Data Intelligence Platform is designed for open collaboration with partners, customers and vendors. Unlike the restrictive systems of our competitors, Databricks offers an open platform that facilitates secure data sharing across various regions, clouds and platforms.

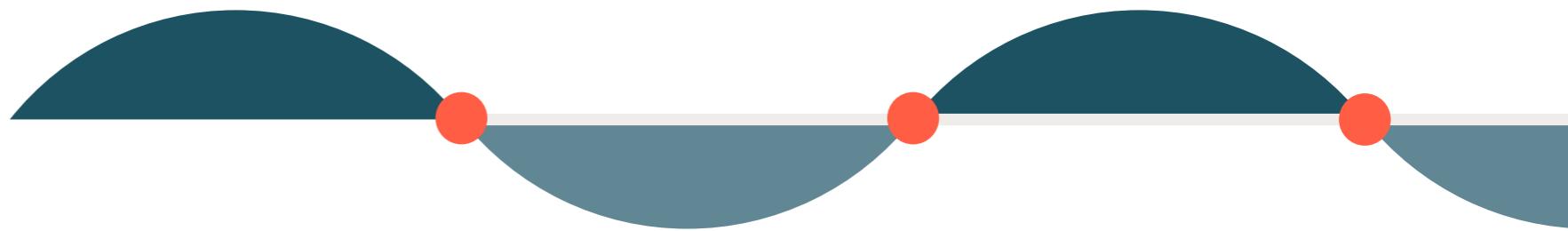
The backbone of this collaboration is **Delta Sharing**, which provides a flexible, secure method for sharing live data with any recipient, regardless of their infrastructure. Developed by Databricks and the Linux Foundation, Delta Sharing is the first open source approach to data sharing across data, analytics and AI. Customers can share live data and AI assets across platforms, clouds and regions with robust security and governance. Whether self-hosting the open source project or using the fully managed Delta Sharing on Databricks, both options provide a platform-agnostic, flexible and cost-effective solution for global data delivery. Databricks customers benefit from a managed environment that minimizes administrative overhead and integrates natively with Databricks Unity Catalog, offering a secure data-sharing experience.

Databricks commitment to innovation and collaboration has yielded significant results in the past year, with the ecosystem seeing impressive growth, including 16,000+ data recipients from a wide range of organizations that have adopted Delta Sharing to collaborate with partners and customers. Since we announced the open source Delta Sharing project three years ago, Delta Sharing has also continued to innovate and make it easy for customers to share live data and AI across platforms, clouds and regions — with no need for replication. We've continued developing new sharing capabilities for both data providers and data recipients, including:

- Volume sharing for large amounts of unstructured data (e.g., images, audio, videos or PDF files)
- AI Model Sharing for Delta Sharing to easily share models with partners and customers and deploy them in their Databricks environment
- Cross-platform view sharing to securely share views to any recipient across platforms
- Cloudflare R2 support to help joint customers take advantage of zero egress fees

Databricks Marketplace is powered by Delta Sharing and offers an open marketplace for discovering, evaluating and installing data and AI assets. Over the past year, Databricks Marketplace has introduced several new features such as foundation and proprietary AI models on Databricks Marketplace, volume sharing on Databricks Marketplace (see [Shutterstock Uses Volume Sharing for Seamless Collaboration](#)), Databricks to open sharing, private exchanges and Solution Accelerators to help data consumers discover and evaluate data products faster and accelerate their analytics and AI initiatives.

Databricks Clean Rooms provides a privacy-safe environment for collaboration for all your data and AI assets without direct access to sensitive data. Organizations are looking for ways to securely exchange their data and collaborate with external partners to foster data-driven innovations. In the past, organizations had limited data sharing solutions, relinquishing control over how their sensitive data was shared with partners, and little to no visibility into how their data was consumed. This created a risk for potential data misuse and data privacy breaches. Customers who tried using other clean room solutions have told us these solutions are limited and don't meet their needs because they often require all parties to copy their data into the same platform, don't allow sophisticated analysis beyond basic SQL queries and offer customers limited visibility or control over their data.



Organizations need an open, flexible and privacy-safe way to collaborate on data. Databricks Clean Rooms meets these critical needs. We recently announced the [Public Preview of Databricks Clean Rooms for AWS and Azure](#).

- **Any cloud, any platform:** Secure, open, flexible collaboration is powered by Delta Sharing. Databricks Clean Rooms allows you to collaborate across clouds, regions and even across platforms using the new Sharing for Lakehouse Federation ([see more about Lakehouse Federation](#)).
- **Any language and workload of your choice:** Unlike other data clean rooms on the market, Databricks Clean Rooms supports any language or workload, including native support for ML and AI with Python. Databricks Clean Rooms is a flexible, interoperable solution, enabling organizations to collaborate with anyone, regardless of cloud or platform without the need for replication.
- **Any scale:** Databricks Clean Rooms also supports collaboration and operational capabilities at scale. With support for APIs, SQL commands and built-in Databricks Workflows orchestration, you can easily automate Databricks Clean Rooms workloads. Collaborators also get approved output data directly in their Unity Catalog that can be conveniently used for subsequent use cases. Coming soon, multiple collaborators can work together in Databricks Clean Rooms.

In this chapter, we'll explore the security architecture of Delta Sharing through three distinct scenarios:

- Databricks customer to Databricks customer
- Databricks customer to open sharing
- Cross-cloud data sharing

We'll highlight the advantages of incorporating Delta Sharing into a modern data collaboration strategy, such as improved operational efficiency through streamlined and secure data exchanges across various platforms and clouds while minimizing complexity and risk. This robust framework accelerates the time to insight, facilitating quicker decision-making while ensuring strong privacy protections that build stakeholder trust. Moreover, the versatility of Delta Sharing supports a wide array of data formats and applications, making it adaptable to evolving business requirements in a secure manner. Each scenario is accompanied by a customer testimonial that provides firsthand insight into the transformative impact of the solution.

Data sharing with Databricks Delta Sharing

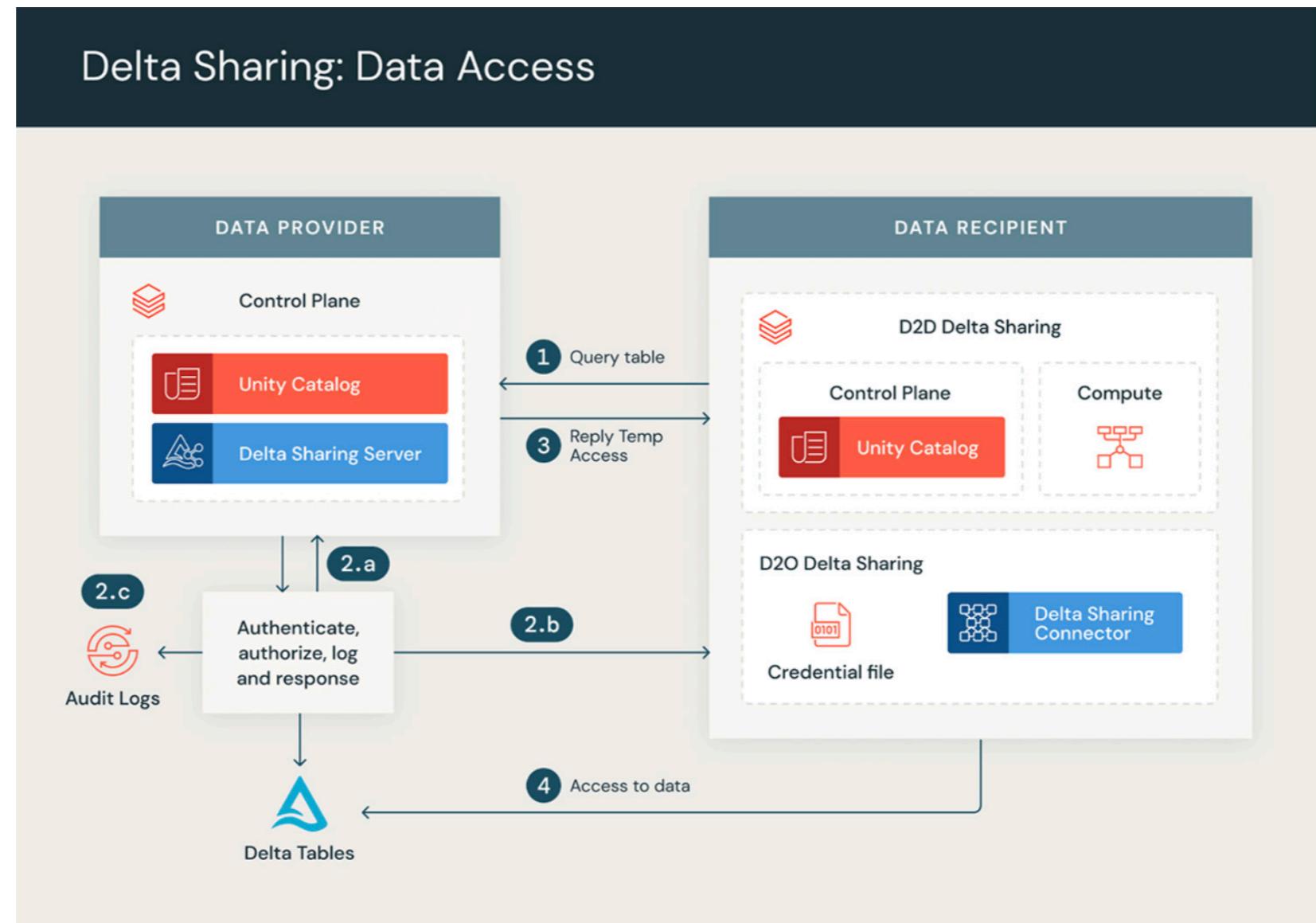
The following sections look at Databricks Delta Sharing scenarios, specifically when the data provider is utilizing the managed version of the Databricks Platform.

DATABRICKS TO DATABRICKS (D2D) DATA SHARING

The Databricks to Databricks (D2D) data sharing scenario illustrates secure, efficient data exchange between two Databricks customers within the Databricks ecosystem. It features managed connections and a no-token exchange system, ensuring simplicity and security. Customers benefit from Delta Sharing's native integration with Unity Catalog, which offers unified governance and security for sharing operations. Sharing isn't limited to data alone — Unity Catalog extends to volumes, notebooks and AI models, showcasing a broad range of functions. Intra-account sharing is enabled by default, while external sharing requires admin-level access for activation. Setting up Databricks Delta Sharing requires a Databricks workspace enabled for Unity Catalog and metastore, along with admin role privileges. Unity Catalog provides a unified governance layer throughout the sharing process — from creating a recipient and establishing shares to granting access. The Delta Sharing service processes API requests, performs thorough authorization checks and maintains detailed activity logs, ensuring transparent and secure operations.

Data access

Unity Catalog plays a crucial role in postauthorization data access. It determines the access method — either cloud tokens or presigned URLs — based on asset type and sharing arrangement. For cloud tokens, a read-only, scoped-down SAS token is created by the provider's Unity Catalog and forwarded to the recipient's compute plane, providing secure, limited-time storage access to the table root directory. Similarly, presigned URLs are generated and sent to the recipient's compute plane for secure, temporary access to storage files. This methodology ensures data sharing is both flexible and secure, meeting a wide array of business needs.



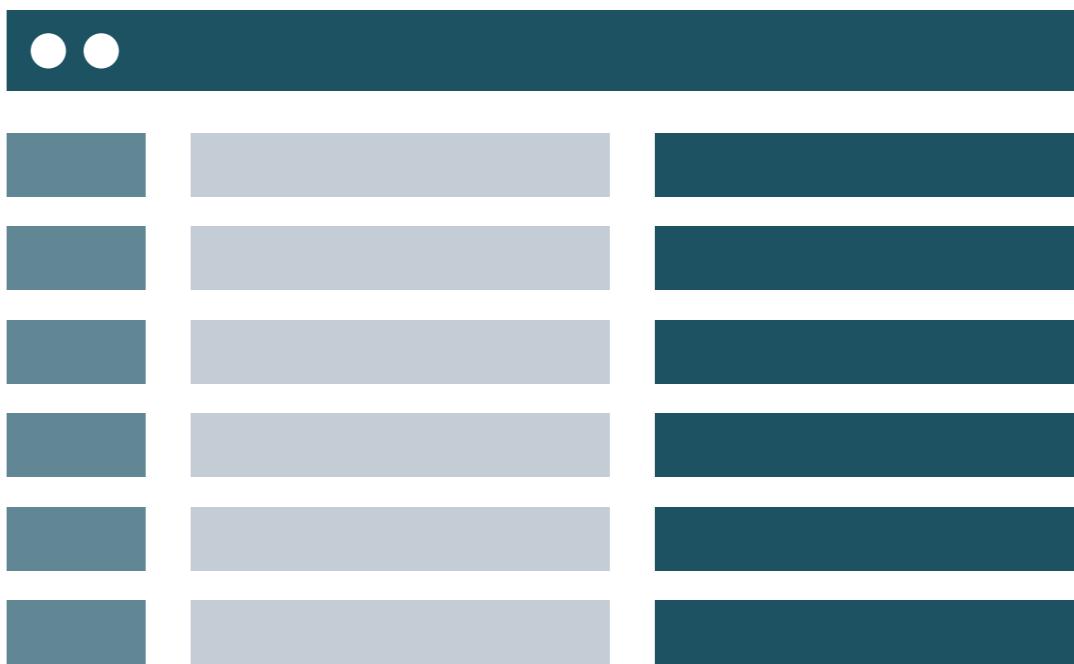
DATABRICKS TO OPEN (D2O) DATA SHARING

In the Databricks to Open (D2O) data sharing scenario, strict security protocols are upheld for Databricks customers sharing data with external third-party users not on the Databricks Platform. Recipients can directly connect to shared data using Delta Sharing connectors that support various systems like pandas, Tableau, Apache Spark™, Rust and more without needing a specific compute platform.

Upon creating an open recipient in Databricks, a secure, one-time activation URL is generated, allowing the recipient to download a credential file containing a Delta Sharing endpoint address and a token. Providers can take immediate action in case of a security breach, such as changing recipient credentials or revoking read permissions.

Data access workflow

When a recipient queries a shared table using the supported connectors, Delta Sharing verifies the recipient using tokens from the credential file and provides presigned URLs for accessing the data. This ensures compatibility with various open source connectors, safeguarding the integrity and security of the shared assets.



CROSS-CLOUD DATA SHARING

Many enterprises adopt cross-cloud strategies to support diverse functionalities across different cloud platforms, facilitate partnerships or integrate data after acquisition. Delta Sharing enables seamless and secure sharing within and across multiple cloud platforms, ensuring operational continuity, fostering innovation and driving growth. Delta Sharing cross-platform sharing capabilities support multicloud environments, making it a clear advantage. It facilitates secure and efficient data exchanges whether within a single cloud or across multiple cloud platforms. Many customers have reported that Delta Sharing promotes interoperability and enhances security across their cloud ecosystems.

Network and storage configuration

Delta Sharing integrates seamlessly with the cloud's native storage security architecture without significant modifications. Designed for Databricks on Azure, AWS and GCP, it aligns with Unity Catalog's requirements and supports data sharing through cloud storage solutions (ADLS Gen2, S3, GCS) using private communication channels or IP address allowlisting for enhanced security. The network and storage configuration works across both intracloud and cross-cloud scenarios, ensuring secure data exchange within the same cloud ecosystem using private endpoints, storage firewalls and network gateways, and leveraging existing cross-cloud private connections for secure data access across different platforms.

Data filtering

Delta Sharing employs two primary methods for data filtering:

- **Partition filtering:** Shares specific table partitions based on recipient properties, allowing controlled access to needed data portions
- **Dynamic views:** Enables sharing subsets of data with recipients using dynamic functions like `current_recipient`, offering fine-grained control over data access and improved manageability

Security, flexibility and seamless integration with Databricks Delta Sharing

Data sharing and collaboration across organizations and platforms are vital for modern business operations.

Delta Sharing, a cutting-edge open data sharing protocol, empowers organizations to securely share and access data across diverse platforms, emphasizing security and scalability without vendor or data format constraints.

It's important to focus on exploring data replication options within Delta Sharing, providing architectural guidance for specific data sharing scenarios. Drawing from our extensive experience with Delta Sharing clients, our objective is to reduce egress costs and improve performance by offering targeted data replication alternatives. While live sharing is suitable for many cross-region data sharing scenarios, there are instances where replicating the entire dataset and establishing a data refresh process for local regional replicas prove to be more cost efficient. Delta Sharing facilitates this through Cloudflare R2 storage, change data feed (CDF) Delta Sharing and Delta deep cloning functionalities. These capabilities make Delta Sharing highly valued for its exceptional flexibility in meeting diverse data sharing needs.

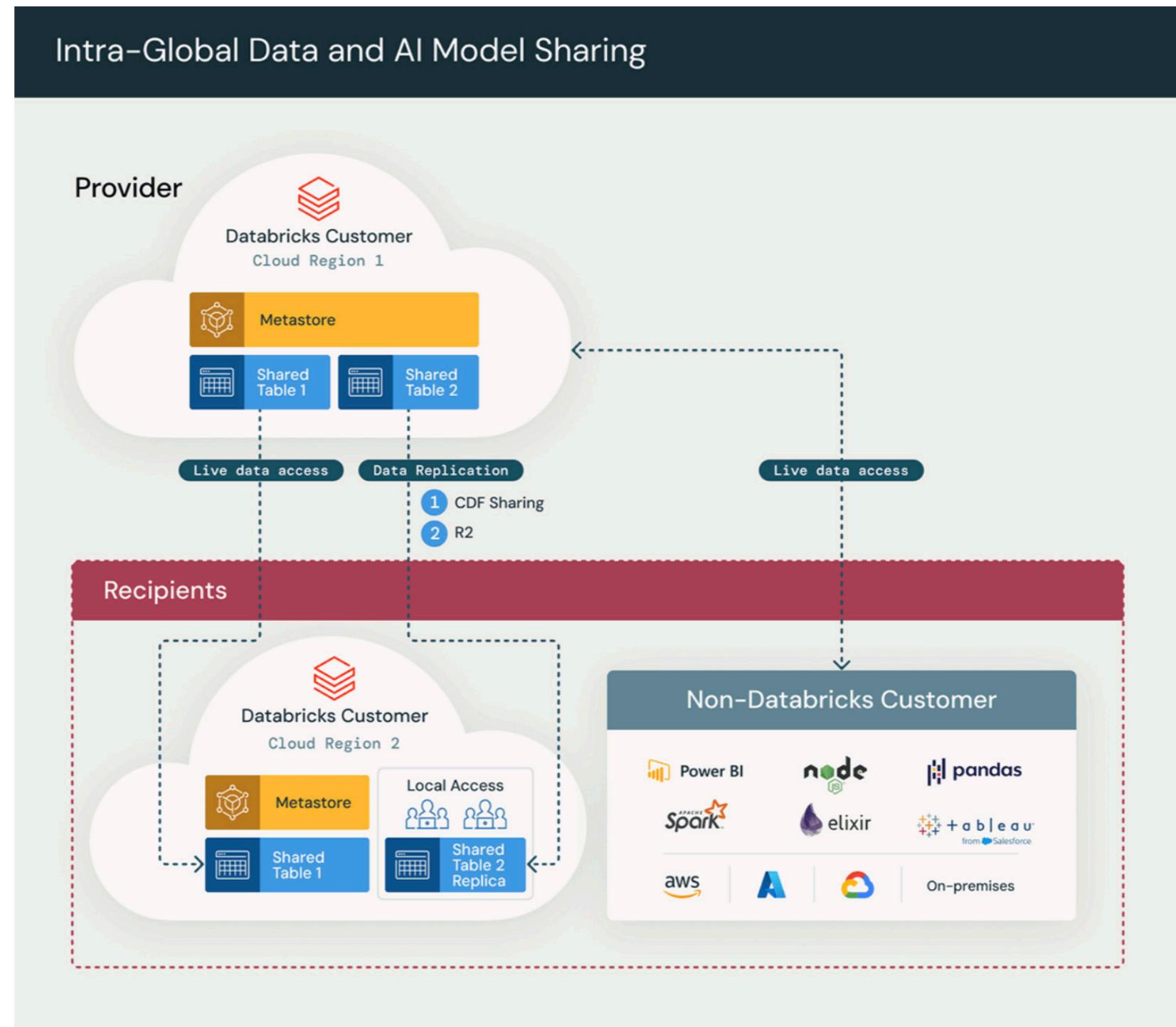
Since its general availability in August 2022, Delta Sharing on Databricks has seen widespread adoption across various collaboration scenarios. Let's explore two common architectural patterns where Delta Sharing has played a pivotal role in enabling and enhancing critical business scenarios:

- Intra-enterprise cross-regional data sharing
- Data aggregator (hub and spoke) model

INTRA-ENTERPRISE CROSS-REGIONAL DATA SHARING

In this scenario, Delta Sharing enables the sharing of data across regions, such as when a QA team is in different regions or a reporting team needs global business activity data. Typically, this involves:

- **Sharing large tables:** Sharing large tables in real time, where recipients execute diverse queries with different predicates. An example is clickstream and user activity data.
- **Local replication:** To enhance performance and manage egress costs, data is replicated to create a local copy, especially when a significant number of users in the recipient's region frequently access these tables.



In this scenario, the business units for both the data provider and the data recipient share the same Unity Catalog account but have different metastores on Databricks.

Intraglobal data and AI model sharing

The high-level architecture of the Delta Sharing solution highlights key steps in the process:

- 1. Creation of a share:** Live tables are shared with the recipient for immediate data access
- 2. On-demand data replication:** Generating a regional duplicate of the data improves performance, reduces the need for cross-region network access and minimizes egress fees

Data replication can be achieved through:

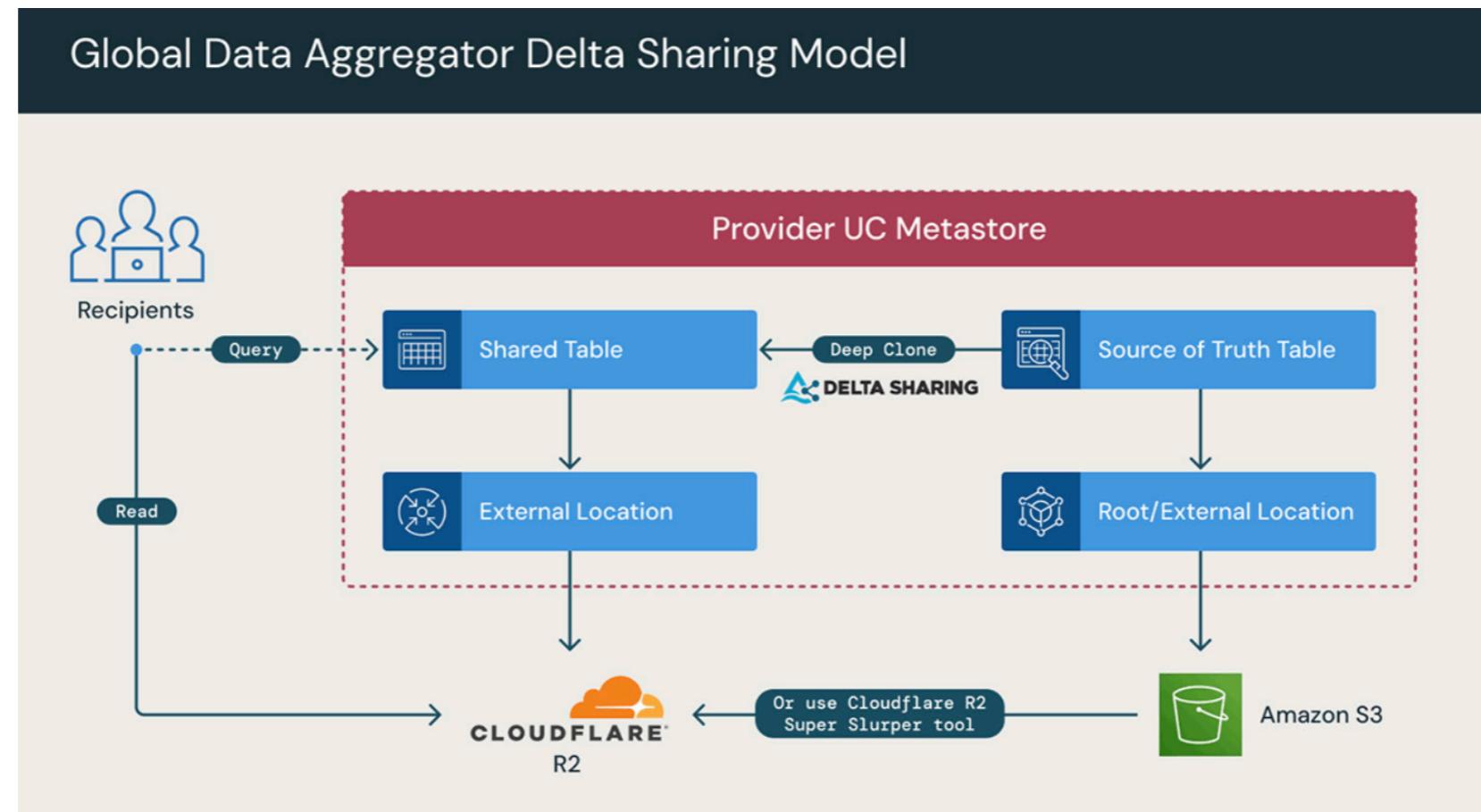
- **Change data feed (CDF) on a shared table:** Sharing table history and enabling CDF for incremental data updates
- **Cloudflare R2 with Databricks:** Using R2 for large-scale data sharing without egress charges
- **Delta deep clone:** Copying source table data and metadata to the clone target within the same Databricks cloud account for efficient data refresh

DATA AGGREGATOR (HUB AND SPOKE) MODEL

This scenario involves sharing data with clients, particularly for data aggregator enterprises collecting and merging data from various sources. This model supports:

- **Connecting recipients across clouds:** AWS, Azure and GCP
- **Supporting a diverse platform:** From Python code to Excel spreadsheets
- **Scalability:** For the number of recipients, shares and data volumes

Using Cloudflare R2 with Databricks simplifies and secures data sharing, eliminating egress fees and complex data transfers. This integration is currently in Private Preview.



In conclusion, Delta Sharing is a cornerstone of the Databricks Data Intelligence Platform, offering secure, flexible and cross-platform data sharing capabilities. Supporting both structured and unstructured data, as well as AI models, Delta Sharing stands out among data exchange platforms. It's widely trusted across industries, as reflected in customer testimonials highlighting its significant impact on operational efficiency and innovation. As the data sharing landscape evolves, Delta Sharing remains future-proof, prioritizing security, flexibility and seamless integration across diverse ecosystems and making it an invaluable asset for enterprises worldwide.

Upgrading to Unity Catalog

With Databricks Unity Catalog, governance capabilities have undergone a significant transformation, overcoming the limitations of the traditional Hive metastore which used to be the default catalog for Databricks workspaces. Unity Catalog represents a paradigm shift towards a centralized governance solution compared to the former approach of distributed governance at the workplace level, and it requires careful planning, attention to detail and collaboration among multiple stakeholders to ensure successful implementation.

The process for upgrading to Unity Catalog

Upgrading to Unity Catalog is a seamless process enabled by inbuilt capabilities for table migration and by using utilities like UCX that are purpose-built for the upgrade process. The UCX tool streamlines the process of assessing existing workspaces to identify assets requiring migration and automates this task. It also aids in planning the upgrade process and determining its complexity.

Once the assessment is completed, for clarity and ease of understanding, the process can be broken down into different stages, which are outlined in the following sections.

GLOBAL SETUP

Global setup involves tasks that are performed once for the entire Databricks Unity Catalog upgrade process.

- **Account console setup:** The Databricks account console is the centralized hub to manage all the workspaces within an enterprise Databricks subscription. An account admin role allows access to the account console to manage workspaces, users and metastores centrally. Identity federation including the automated user sync using SCIM is also set up at the account console.
- **Architecture design:** The upgrade of Unity Catalog warrants a well-designed architecture that aligns with the upgraded governance patterns enforced by Unity Catalog. By adhering to these best practices, the user experience is optimized, ensuring a smooth transition without impacting existing downstream consumers of the system.
- **Set up the metastore:** The metastore acts as the top-level container for data governed by Unity Catalog for a specified cloud region. Unity Catalog metastores register metadata about securable objects (such as tables, volumes, external locations and shares) and the permissions that govern access to them.

PER-APPLICATION TASKS

These tasks are carried out for each individual application during the upgrade process and may vary based on the deployment architecture, specifically whether one application is part of one workspace or multiple applications are deployed in the same workspace.

- **Set up the catalogs:** The first layer of the object hierarchy in Unity Catalog is the *catalog*. It serves as the top-level container for organizing and managing all data assets, including tables, views, functions, volumes and machine learning models. By setting up meaningful catalogs, teams can efficiently structure and govern their data assets, enhancing data discoverability and productivity.
- **Code changes:** The upgrade to Unity Catalog also introduces features such as governed file systems like volumes, which replace DBFS, and a three-level namespace. This allows for more efficient and organized data management. However, it's essential to note that any unsupported code patterns must be migrated to Unity Catalog-compatible ones. For instance, any RDD operations should be updated to use DataFrames in the Spark API.

PER-WORKSPACE TASKS

These tasks are completed for each workspace as part of the Databricks Unity Catalog upgrade.

- **Enable the workspace:** A workspace is enabled for Unity Catalog — and in turn identity federation — when a metastore is attached to the workspace from the account console.
- **Upgrade tables and workflows:** Existing tables can be upgraded, and the process depends on whether it's a managed or an external table. Databricks Workflows needs to be updated to use the Unity Catalog-enabled compute.

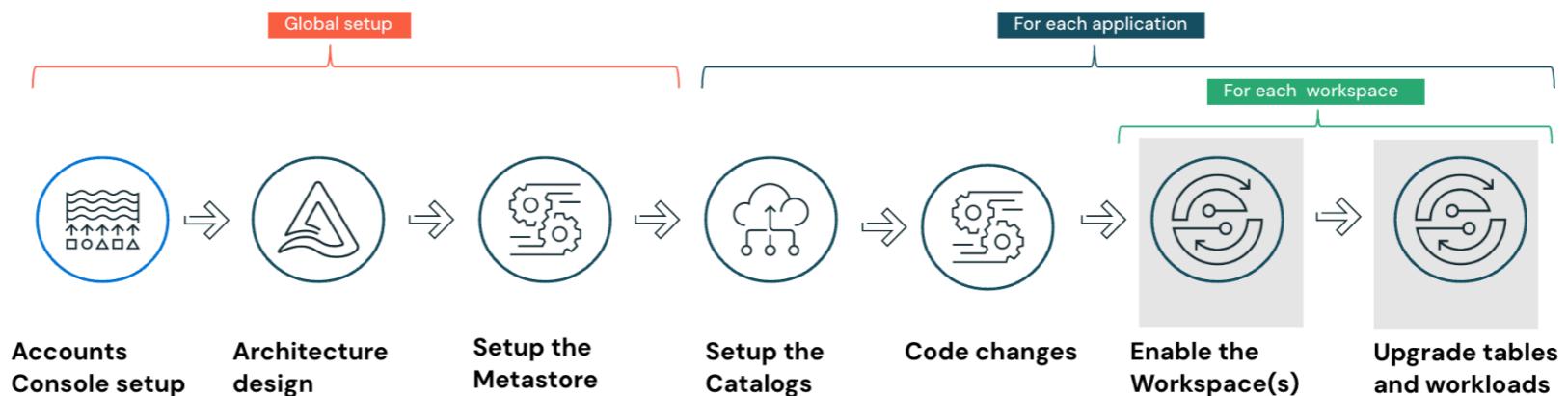


Figure: Unity Catalog upgrade process

IDENTITY FEDERATION

When Unity Catalog is enabled for a workspace, it also enables identity federation by default. With identity federation, you configure Databricks users, service principals and groups once in the account console, rather than repeating configuration separately in each workspace. This both reduces friction in onboarding a new team to Databricks and enables you to maintain one SCIM provisioning application with your identity provider to the Databricks account, instead of a separate SCIM provisioning application for each workspace. Once users, service principals and groups are added to the account, you can assign them permissions on workspaces. You can only assign account-level identities access to workspaces that are enabled for identity federation.

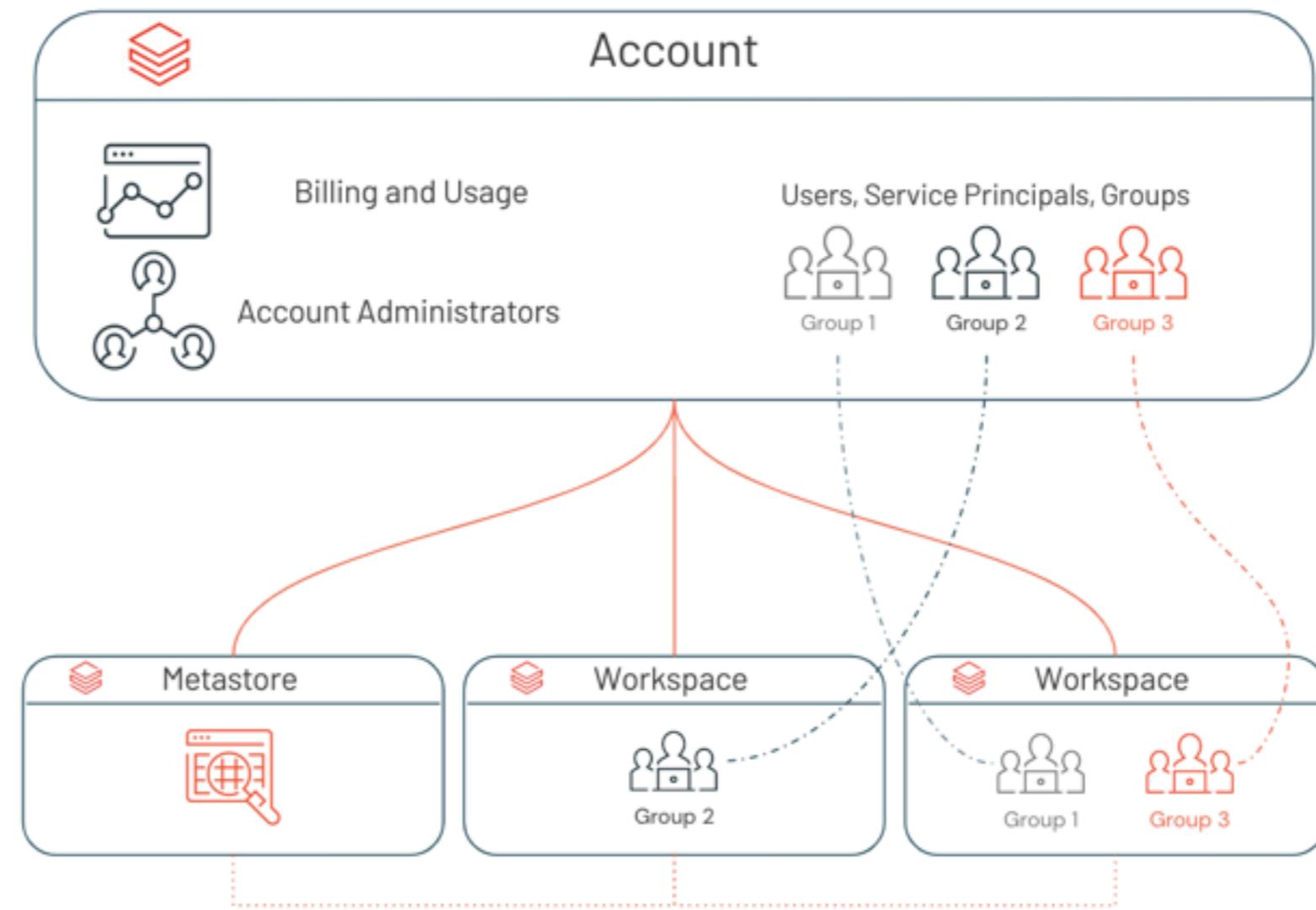


Figure: Identity federation with principals at the account level

Migrate workspace-level SCIM provisioning to the account level

If you have workspace-level SCIM provisioning set up in your workspace, you should set up account-level SCIM provisioning and turn off the workspace-level SCIM provisioner. Workspace-level SCIM will continue to create and update workspace-local groups. Databricks recommends using account groups instead of workspace-local groups to take advantage of centralized workspace assignment and data access management using Unity Catalog. Workspace-level SCIM doesn't recognize account groups that are assigned to your identity federated workspace, so workspace-level SCIM API calls will fail if they involve account groups.

Note: Microsoft Entra ID doesn't support the automatic provisioning of nested groups to Azure Databricks. Microsoft Entra ID can only read and provision users that are immediate members of the explicitly assigned group.

ADMIN ROLES IN DATABRICKS

As part of the upgrade process, a few key users need to be identified for the admin roles within the platform.

The **account admin** is responsible for:

- Creating workspaces
- Creating and configuring metastores
- Creating users, groups and service principals
- Granting users access to workspaces
- Setting billing budget threshold alerts
- Enabling system tables and delegating access to them

The **metastore admin** is responsible for:

- Creating CATALOG, CONNECTION, STORAGE CREDENTIAL, EXTERNAL LOCATION, FOREIGN CATALOG
- Creating SHARE, RECIPIENT, PROVIDER, ALLOWLIST, MATERIALIZED VIEW
- Changing OWNER of any securable object

Note: Through their ability to transfer ownership of all objects, metastore admins have the ability to grant themselves read and write access to any data in the metastore. There's no direct access by default. Granting of permissions is audit-logged. A metastore admin role is an optional role, and the responsibilities can be delegated to another group/user or the workspace admin.

The **workspace admin** is responsible for:

- Adding users and groups to workspace
- Creating clusters and cluster policies
- Changing OWNER of clusters, workflows, notebooks, queries, dashboards

If your workspace was enabled for Unity Catalog automatically, workspace admins have the following privileges on the attached metastore by default.

- Creating CATALOG, EXTERNAL LOCATION, STORAGE CREDENTIAL
- Creating CONNECTION, SHARE, RECIPIENT, PROVIDER, MATERIALIZED VIEW

Apart from the admin roles, the **data owner** role is also available and is responsible for:

- Changing ownership of securable objects (CATALOG, SCHEMA, TABLE, VIEW, etc.)
- Granting any privilege to any principal

Note: The ownership of the data can be defined at the catalog, schema and table levels. Each securable object in Unity Catalog has an owner. The owner can be any user, service principal or account group, known as a *principal*. The principal that creates an object becomes its initial owner.

Using the Databricks Labs UCX tool for accelerating migration

UCX is a Databricks Labs project that provides tools to help you assess your environment and help you upgrade your non-Unity Catalog workspace to Unity Catalog.

Assessing the existing workspace (assessment workflow)

One of the key benefits of the UCX tool is its ability to save time and resources by automatically assessing and identifying assets that need migration. This allows businesses to focus on other critical aspects of the upgrade process and make informed decisions about their workspace upgrades. UCX accelerates the Unity Catalog upgrade process by deploying an assessment job which generates a detailed report on the assets that need to be migrated.

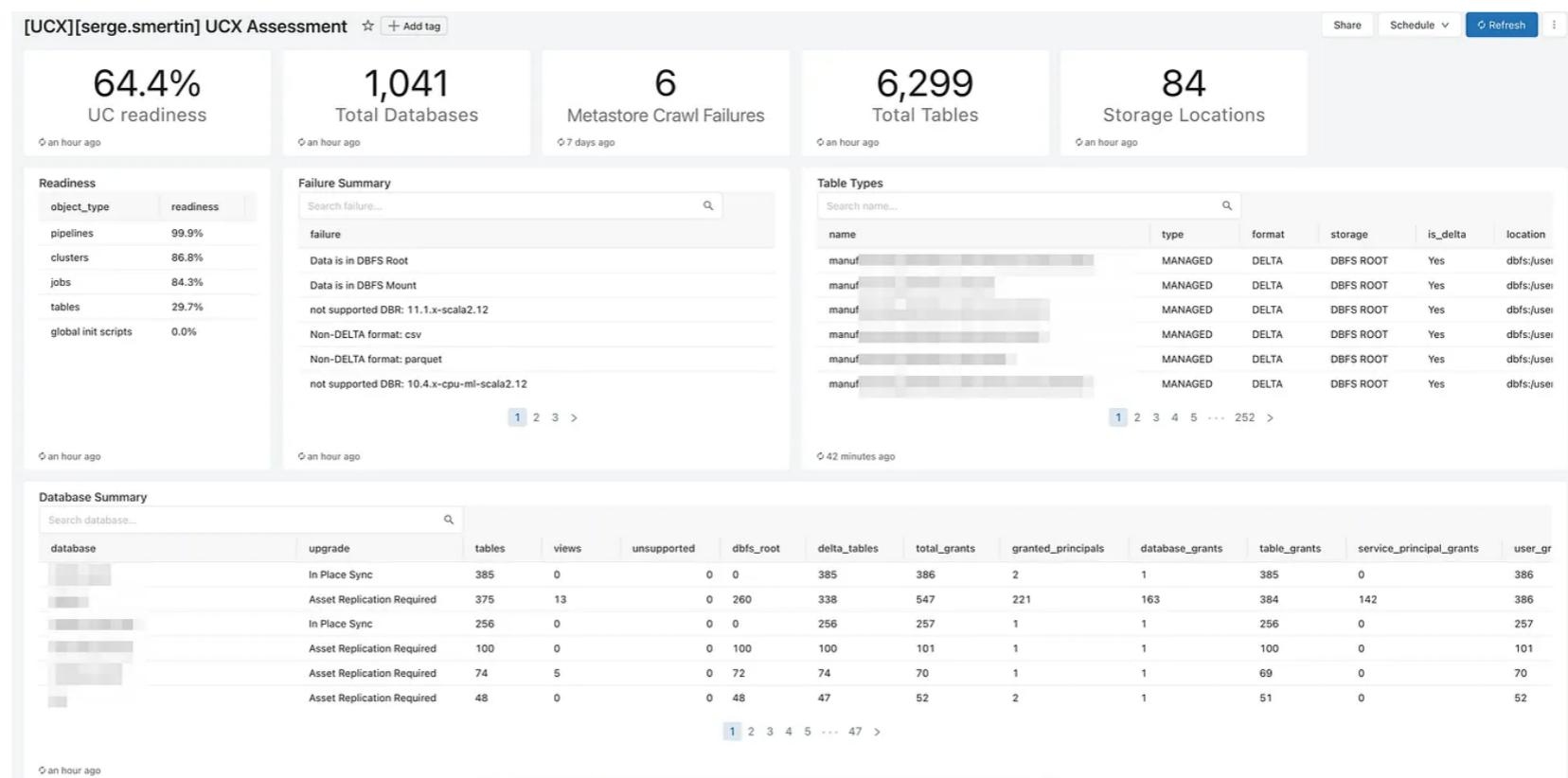


Figure: Sample UCX assessment report

After the assessment report becomes available, the upgrade planning process can commence by examining the report for complexity and the number of assets, including tables, groups, storage locations and more. The Unity Catalog upgrade can be executed on a per-application or per-workspace basis, and it's recommended that you carry out the upgrade in a phased manner to reduce downtime during the process.

Group migration from workspace to the account (**group migration workflow**)

Unity Catalog introduces identities at the identity federation and account levels, enhancing security and streamlining access management. Identity federation enables you to configure users, service principals and groups in the account console and then assign those identities access to specific workspaces. This simplifies Databricks administration and data governance. However, groups created or synced to existing workspaces using the SCIM API won't be supported in Unity Catalog and must be upgraded to the account level to benefit from these new features.

The UCX tool automates the migration of groups from workspace to the account level and helps you to upgrade all Databricks workspace assets: legacy table ACLs, entitlements, AWS instance profiles, clusters, cluster policies, instance pools, Databricks SQL warehouses, Delta Live Tables, jobs, MLflow experiments, MLflow Model Registry, SQL dashboards and queries, SQL alerts, token and password usage permissions that are set on the workspace level, secret scopes, notebooks, directories, repositories and files.

Table migration from Hive metastore to Unity Catalog (**table migration workflow**)

To enable asset governance, existing tables in the Hive metastore must be migrated to Unity Catalog. The migration process varies depending on whether the table is an external table or a managed table in Hive. External tables in the Hive metastore are upgraded as external tables in Unity Catalog, using **SYNC**. Managed tables in the Hive metastore that are stored in workspace storage (also known as DBFS root) are upgraded as managed tables in Unity Catalog using **DEEP CLONE**.

Hive-managed tables must be in Delta or Parquet format to be upgraded. External Hive tables must be in one of the following data formats:

- DELTA
- CSV
- JSON
- AVRO
- PARQUET
- ORC
- TEXT

Note: UCX, like all projects in the [Databricks Labs GitHub account](#), is provided for your exploration only and isn't formally supported by Databricks with service-level agreements (SLAs). It's provided as is. We make no guarantees of any kind. Don't submit a Databricks support ticket relating to issues that arise from the use of this project. Instead, file a [GitHub issue](#). Issues will be reviewed as time permits, but there are no formal SLAs for support.

Upgrading assets

The following sections document the different assets involved in upgrading to Unity Catalog.

UNITY CATALOG ARTIFACT CREATION

There are numerous artifacts that should be created before starting your upgrade process.

Metastore

A metastore is the top-level container that stores metadata about all of your data assets (such as tables, volumes, external locations and shares) and the access permissions associated with them. You should have one metastore for each region in which your organization operates. To enable a workspace with Unity Catalog, you must attach the workspace to a metastore. You can find the steps to create a metastore and attach it to the workspace in the [official documentation](#).

Storage credentials

Storage credentials authenticate and authorize Databricks to access your data stored in cloud storage.

Principals can be granted access to it. Storage credentials are primarily used to create external locations.

External locations

An external location is a reference to a location in cloud storage. When you create an external location, you specify the storage credentials to use and the path to the location in the cloud storage.

Catalogs

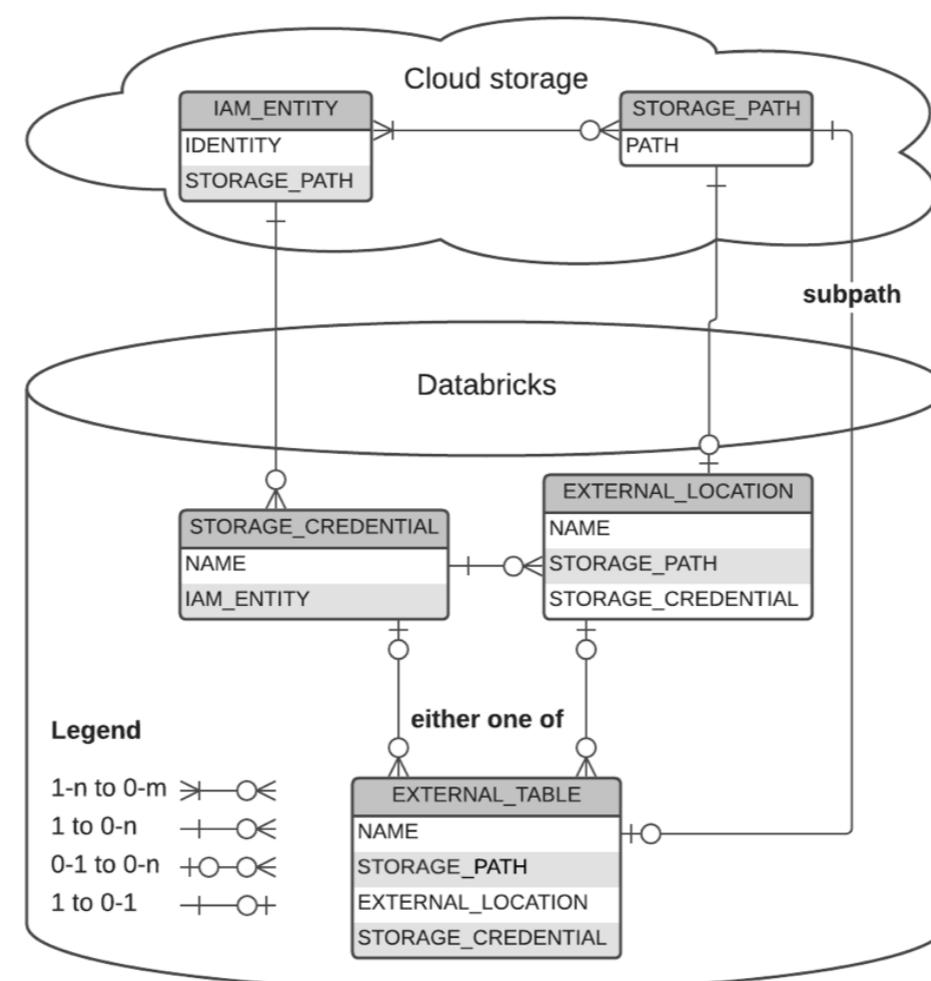
A catalog in Databricks is a container for a collection of schemas. It provides a way to organize your databases in a hierarchical manner, making it easier to manage and access them.

Schemas

A schema in Databricks is a logical grouping of data and AI assets like tables, views, volumes, functions and models. It provides a way to organize your data and AI assets, making it easier to manage and access them.

Volumes

Volumes are Unity Catalog objects representing a logical volume of storage in a cloud object storage location. Volumes provide capabilities for accessing, storing, governing and organizing files. Volumes can be either managed or external.



UPGRADING TABLES

When it comes to table upgrade, the main goal is to avoid or minimize data movement as much as possible.

Remember, a table can either be managed or external. The location of the data can either be on DBFS root storage location, DBFS mounted cloud object storage or directly specified cloud storage (such as S3://, abfss:// or gcs://). The table file format and interface can be a file format such as Delta, Parquet, Avro or an interface such as Hive SerDe.

For managed tables, there are multiple scenarios:

- **Tables on DBFS root location:** The data files for the managed tables reside within DBFS root, which is the default location for the Databricks-managed HMS database
- **Tables on DBFS mounted location:** This is when the parent database has its location set to external paths, e.g., a mounted path from the object store
- **Tables on cloud storage location:** This is when the parent database has its location set to external paths, e.g., a cloud object store

For external tables, you can have the following scenarios:

- **Tables on DBFS root location:** The data files for the external tables reside within DBFS root, which is the default location for the Databricks-managed HMS database. The table definition has the "location" clause which makes the table external.
- **Tables on DBFS mounted location:** This is when the table is created with a location clause with a path specifying a mounted path from the object store
- **Tables on cloud storage location:** This is when the parent database has its location set to external paths, e.g., a cloud object store
- **Hive SerDe tables:** These are the tables which were created using the Hive SerDe interface
- **Azure tables:** Tables on Gen 1 blob storage require managed tables to be moved to a Gen 2 blob storage prior to upgrade in order to avoid a schema error

MIGRATION MATRIX

The following matrices identify the different scenarios and the methodology for upgrading them.

HMS storage format using DBFS root storage

HMS Table Type	Target Unity Catalog Table Type	Target Unity Catalog Data File Format	Methodology
Managed	External or Managed	<ol style="list-style-type: none"> 1. Delta for managed (Preferred) 2. As that of the HMS source data file format or Delta (Preferred) in case of external <p>Note: Preferably change it to Delta while you're doing the migration using CTAS</p>	CTAS
External	External or Managed	<ol style="list-style-type: none"> 1. Delta for managed (Preferred) 2. As that of the HMS source data file format or Delta (Preferred) in case of external <p>Note: Preferably change it to Delta while you're doing the migration using CTAS</p>	CTAS

HMS Hive SerDe table

HMS Table Type	Target Unity Catalog Table Type	Target Unity Catalog Data File Format	Methodology
Hive SerDe external or managed Note: Regardless of the underlying storage format, Hive SerDe follows the same migration path	External or Managed	<ol style="list-style-type: none"> 1. Delta for managed (Preferred) 2. As that of the HMS source data file format or Delta (Preferred) in case of external <p>Note: Preferably change it to Delta while you're doing the migration using CTAS</p>	CTAS

HMS storage format using DBFS mounted storage

HMS Table Type	Target Unity Catalog Table Type	Target Unity Catalog Data File Format	Methodology
Managed	External	As that of the HMS source data file format	<ul style="list-style-type: none"> 1. Run Sync to create Unity Catalog external table 2. Convert HMS managed to HMS external (code provided below) 3. Drop HMS table after all dependencies are resolved so that there's no way to access the data using HMS table 4. Unmount the mount point after all dependencies are resolved so that there's no way to access the data using mount points <p>Note: Ensure that the HMS table is dropped individually after conversion to an external table. If the HMS database/schema was defined with a location and if the database is dropped with the cascade option, the underlying data will be lost and the migrated Unity Catalog tables will lose the data.</p>
Managed	Managed	Delta	CTAS
External	External	As that of the HMS source data file format	<ul style="list-style-type: none"> 1. Run Sync to create Unity Catalog external table 2. Drop HMS table after all dependencies are resolved so that there's no way to access the data using HMS table 3. Unmount the mount point after all dependencies are resolved so that there's no way to access the data using mount points
External	Managed	Delta	CTAS

HMS storage format using cloud object storage

HMS Table Type	Target Unity Catalog Table Type	Target Unity Catalog Data File Format	Methodology
Managed	External	As of the source data file format	<ol style="list-style-type: none"> Run Sync to create Unity Catalog external table Convert HMS managed to HMS external (code provided below) Drop HMS table after all dependencies are resolved so that there's no way to access the data using HMS table <p>Note: Ensure that the HMS table is dropped individually after conversion to an external table. If the HMS database/schema was defined with a location and if the database is dropped with the cascade option, the underlying data will be lost and the migrated Unity Catalog tables will lose the data.</p>
Managed	Managed	Delta	CTAS
External	External	As of the source data file format	<ol style="list-style-type: none"> Run Sync to create Unity Catalog external table Drop HMS table after all dependencies are resolved so that there is no way to access the data using HMS table
External	Managed	Delta	CTAS

Use this code to upgrade HMS managed to HMS external:

```

• • •

1 import org.apache.spark.sql.catalyst.catalog.{CatalogTable,
2 CatalogTableType}
3 import org.apache.spark.sql.catalyst.TableIdentifier
4 val tableName = "table"
5 val dbName = "dbname"
6 val oldTable: CatalogTable =
7 spark.sessionState.catalog.getTableMetadata(TableIdentifier(tableName,
8 Some(dbName)))
9 val alteredTable: CatalogTable = oldTable.copy(tableType =
10 CatalogTableType.EXTERNAL)
11 spark.sessionState.catalog.alterTable(alteredTable)

```

Before upgrading the views, make sure that you've upgraded all of the referenced tables and views to Unity Catalog. After you upgrade all of a view's references to the same Unity Catalog metastore, you can re-create a new view that references the new tables.

If dynamic views are present, they should be modified to use the `is_account_group_member` function. Additionally, [use the row filters and column masking](#) to enable fine-grained access at the row and column levels. Functions also need to be re-created in the corresponding catalog - schema.

We recommend you use [Databricks Labs UCX](#) to manage the table, view and function upgrades seamlessly, instead of developing custom scripts.

UPGRADING CLUSTERS TO UNITY CATALOG

Prerequisites

Before initiating the upgrade, ensure that Unity Catalog is enabled for your account. A metastore should be created and assigned to the workspace, and catalogs/databases/tables should be created with proper access control lists (ACLs). The cluster should operate in either shared access mode or single user access mode to leverage Unity Catalog. Shared access mode is recommended for most scenarios unless your workload requires certain functionalities supported by this mode. For production workloads, it's recommended to restrict the access modes using cluster policies to ensure consistent usage of Unity Catalog-enabled clusters.

Upgrade scenarios

Consider the following scenarios when upgrading:

- **Clusters running Databricks Runtime (DBR) earlier than 11.1:** In this case, you must dismantle the existing cluster and rebuild it. Upgrading clusters to use Unity Catalog isn't possible in this scenario as Unity Catalog requires DBR 11.1 or later. Upgrade the DBR to version 11.1 or later, with the latest long-term support (LTS) version being highly recommended.
- **Clusters running DBR 11.1 and later:** To upgrade from any of the legacy cluster modes to the Unity Catalog-leveraged security mode, edit your cluster and change the access mode to either single user or shared from the access mode drop-down menu in the cluster UI.

Interactive and automated clusters

For an interactive cluster, switch to Unity Catalog mode by clicking the Edit button on the cluster page and changing the access mode to either single user or shared.

For job clusters, the prerequisites are the same as for interactive clusters. Navigate to the job details screen and click Configure under Compute.

To minimize effort, change and fix/default the access mode in the compute policy and roll out the changes to all existing clusters. Also, ensure that credential pass-through isn't enabled in a cluster (either in Unity Catalog or fixed in compute policies), as this would disable Unity Catalog access. To add a default catalog, use the following configuration in the Spark settings of the cluster: `spark.databricks.sql.initial.catalog.name name_of_catalog`.

Limitations

Note that Databricks Runtime for Machine Learning and Spark machine learning library (MLlib) aren't supported in shared access mode. Spark-submit jobs are also not supported in shared access mode. Single-node cluster isn't supported in the SHARED mode.

The use of dynamic views for row-level or column-level security isn't supported in single user mode. Also, to read from a view, you must have SELECT on all referenced tables and views in this mode. Refer to [this documentation](#) to see the full list of limitations in both access modes.

UPGRADING MOUNT POINTS AND DBFS

Mount points and file storage in DBFS aren't supported by Unity Catalog, so all data stored in mount points or DBFS needs to be moved to external tables or volumes.

Prior to Unity Catalog, mount points were used to read tabular data from paths to the object storage and unstructured data. In Unity Catalog, these patterns are covered by external tables and volumes respectively.

Premigration assessment

Before migrating, collect information about the existing mount points, the underlying storage location and the impacted code assets. This can be accomplished via `dbutils.fs.mounts()` or by running the UCX assessment.

Data migration

Data stored in the DBFS root location should be copied to an external location for Unity Catalog to access it. For DBFS-mounted cloud paths, the data doesn't need to be moved. Depending on the type of data, there are two upgrade scenarios:

- **Tabular data:** To read tabular data from paths to the object storage, use [external tables](#). Follow the steps mentioned in the “[Upgrading tables](#)” section of this chapter.
- **Unstructured data:** To read unstructured data in the object storage, define external volumes over the cloud path that contains the data. Unity Catalog volumes come under a schema, just like a table, hence Unity Catalog volume location is a subfolder of the external location pointed to by its parent schema or catalog. You can reuse the same external location to create multiple volumes as subfolders. You can manage permissions for each volume separately, even though they share the same parent location.

Migration execution

During the migration, once you've defined volumes, you can access the same cloud storage location in cloud storage via mount point and Unity Catalog (external volume). If you upload a file via Unity Catalog Volumes, the file will be available via the mount point and vice versa. This allows you to migrate the mount points with zero downtime for your application.

Code adjustment and cleanup

Once your volume is in place, update your code to use the volume instead of the mount point. If there are multiple notebooks and files in your workspace, consider scanning all of them and replacing the respective paths. As you eliminate the need for each mount point, clean up permissions and storage credentials.

Best practices

Remember, mount points have workspace-level scope, while Unity Catalog volumes have metastore-level scope. It's always good practice to use parameters in your code. This is even more critical for Unity Catalog volume paths.

UPGRADING BATCH AND STREAMING WORKLOADS

To upgrade both batch and streaming workloads, make sure all of the upgrades mentioned in the previous section that are relevant to the job are already in place, including table upgrades, cluster upgrades, mount points and DBFS upgrades. The code should be adjusted to refer to the volumes created and it should read and write to Unity Catalog tables. To avoid too many changes in the code, **set the default catalog at the cluster level**, so that you can continue using a two-level namespace instead of the three-level namespace. If this isn't a viable option, you can also consider adding "USE <catalog-name>" at the top of the job, so that rest of the queries in the job don't change.

When upgrading a streaming workload, stop the streaming job to assure a seamless transition. For streaming workloads, the checkpoint locations in the code should be modified to refer to the corresponding external location or volume path. If the checkpoints are stored in a DBFS root location, they should be copied over to an external location/volume. Since the checkpoint location points to the same location or contains the previous checkpoints, the job will resume from the previous point upon restart.

Any use of global init scripts should also be replaced with **cluster-scoped init scripts**.

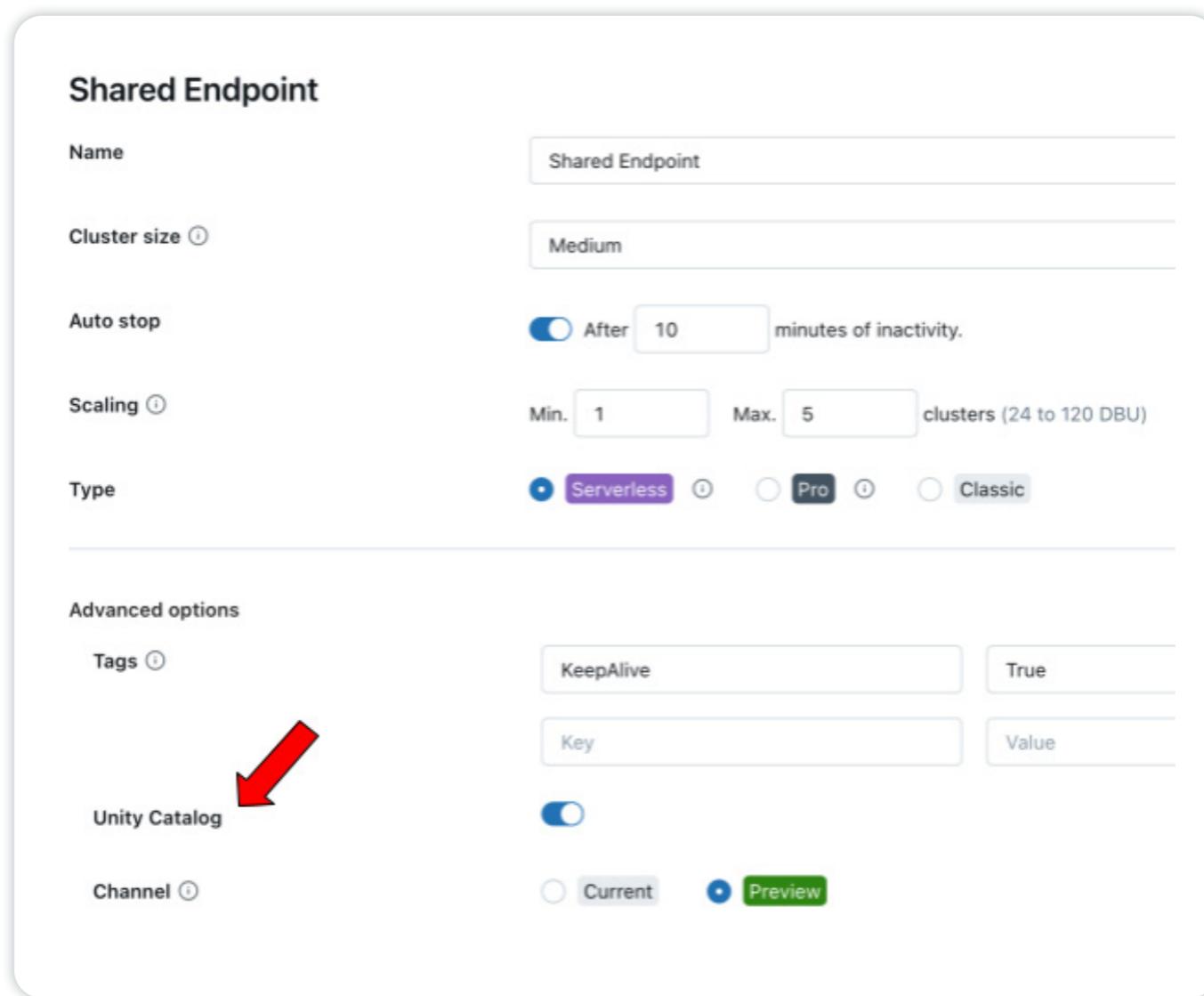
UPGRADING DLT PIPELINES

To begin the migration process, ensure that your DLT job cluster runs on Databricks Runtime 13.1 or later and follow these steps.

1. Temporarily halt the DLT job you intend to upgrade to prevent new data ingestion during the migration.
2. Create a new Unity Catalog table to store data previously handled by your DLT pipeline as mentioned in the **table migration** section.
3. When migrating the table, ensure that there is schema compatibility with the existing DLT table.
4. Once the new table is in place, modify the DLT job definition to use it as the data sink, replacing references to the old DLT table.
5. Restart the DLT job, which will now write data to the Unity Catalog table.

UPGRADING WAREHOUSES

Once your workspace is attached to the metastore, any new warehouse created will be enabled for Unity Catalog by default. For existing warehouses, edit the configuration, and enable the Unity Catalog toggle under "Advanced options" as shown below:



If the majority of your queries in a workspace point to the same catalog, then you can make that catalog the default catalog for the workspace, and hence for the warehouses. You can change the default catalog using the admin console as described [here](#).