

Understand Azure Databricks architecture

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/understand-azure-databricks-architecture/1-introduction>

Introduction

Completed

When you deploy Azure Databricks for data engineering workloads, understanding its architecture becomes essential for making informed decisions about data organization, security configurations, and resource management. The architecture determines where your data resides, how compute resources are allocated, and how different components interact to provide the unified analytics platform you rely on.

Azure Databricks organizes resources through a hierarchical structure that spans from account-level governance down to individual data objects. At the foundation, the **control plane** manages orchestration and configuration while the **compute plane** processes your data—either in **serverless** environments fully managed by Azure Databricks or in **classic compute** running within your Azure subscription. This separation enables you to maintain security and governance while scaling workloads efficiently.

Storage options in Azure Databricks have evolved to meet different organizational needs. **Default storage** simplifies getting started by providing fully managed storage in serverless workspaces without configuration overhead. **External storage** connects Unity Catalog to your existing cloud storage accounts, enabling you to work with data managed outside Azure Databricks while maintaining governance. **Unity Catalog managed storage** lets you define where Unity Catalog stores data at the catalog or schema level, providing fine-grained control over data placement while Unity Catalog handles the lifecycle.

Throughout this module, you explore how these architectural components work together. You learn how the **account hierarchy** organizes resources across workspaces and metastores, how control and compute planes separate responsibilities, and how different storage patterns serve specific use cases. By understanding these fundamentals, you'll be equipped to configure Azure Databricks environments that align with your organization's security, governance, and operational requirements.

2. Understand Azure Databricks architecture

<https://learn.microsoft.com/en-us/training/modules/understand-azure-databricks-architecture/2-understand-azure-databricks-architecture>

Understand Azure Databricks architecture

Completed

When you work with Azure Databricks as a data engineer, understanding its architecture helps you make informed decisions about how to organize your data, configure compute resources, and manage security. The architecture determines how your workloads run, where your data resides, and how different components interact.

Azure Databricks uses a hierarchical structure that organizes resources from the account level down to individual data objects. This structure, combined with the separation between control and compute planes, provides flexibility and security for your data engineering workloads.

Azure Databricks account hierarchy

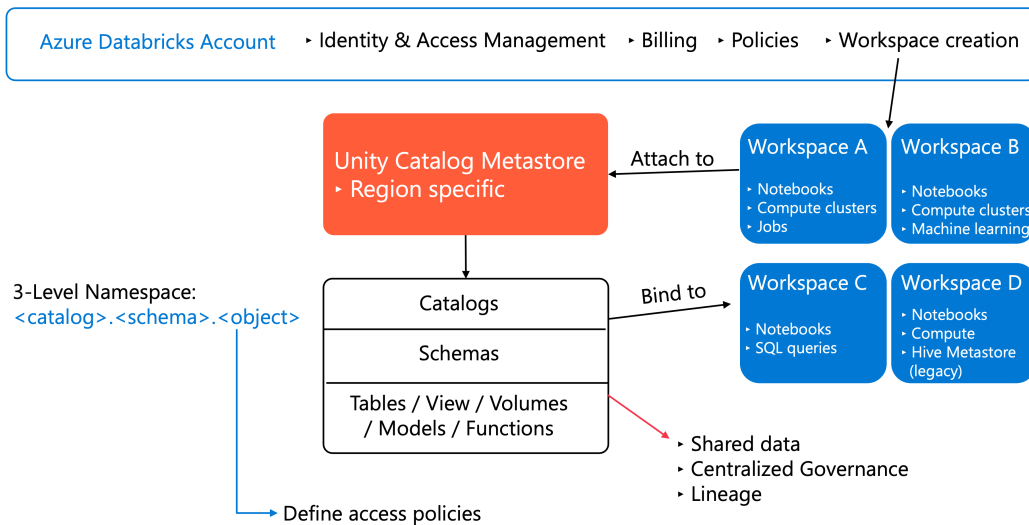
Azure Databricks organizes resources in a hierarchical structure that starts at the account level. The **account** is the top-level construct you use to manage Azure Databricks across your organization. At this level, you manage identity and access, create and configure workspaces, attach Unity Catalog metastores, and oversee billing and policies.

The **metastore** is the top-level container for metadata in Unity Catalog, registering information about your data and AI assets such as tables, views, volumes, models, and functions, along with the permissions that govern access to them. Each metastore is region-specific, and workspaces attached to the same metastore share a unified view of data, enabling centralized governance across your organization.

Unity Catalog organizes data using a three-level namespace structure: `<catalog-name>.<schema-name>.<object-name>`. Unlike the legacy Hive metastore that operates per workspace, Unity Catalog metastores operate at the account level, allowing you to define data access policies once and apply them consistently across all attached workspaces.

Before Unity Catalog, Azure Databricks used the **Hive metastore (legacy)** as a workspace-level metadata store. When a workspace is enabled for Unity Catalog, the legacy Hive metastore becomes accessible as a catalog named `hive_metastore`, allowing you to query existing data using the format `hive_metastore.<schema-name>.<table-name>`. Databricks recommends migrating to Unity Catalog for enhanced security, centralized governance, and built-in auditing.

You create multiple **workspaces** within an account. Workspaces are the collaboration environments where you and your team run compute workloads such as data ingestion, interactive exploration, scheduled jobs, and machine learning training. Each workspace provides an isolated environment for your projects while sharing the same account-level governance.



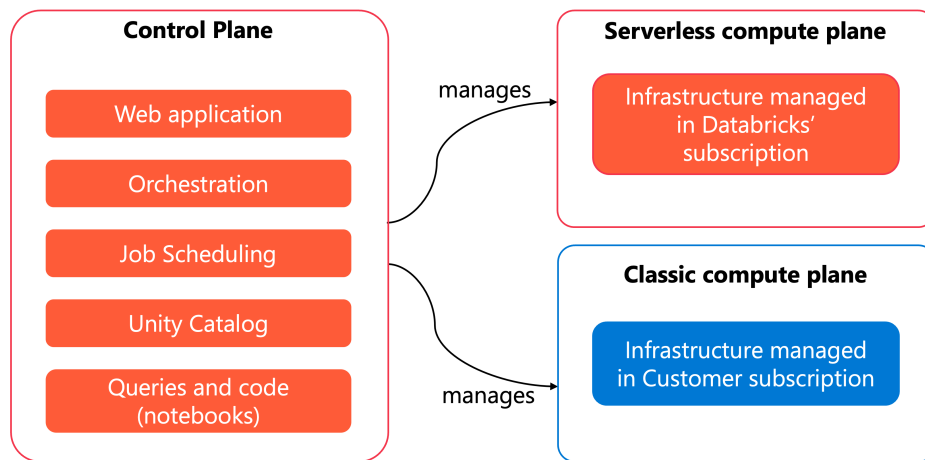
This hierarchical approach allows you to separate concerns: manage organization-wide policies at the account level, isolate projects in workspaces, and govern data centrally through Unity Catalog.

Control plane and compute plane

Azure Databricks separates its architecture into two distinct planes: the control plane and the compute plane.

The **control plane** includes the backend services that Azure Databricks manages in your Azure Databricks account. This plane hosts the web application you use to interact with Azure Databricks, manage configurations, and monitor your workloads. The control plane handles orchestration, job scheduling, and cluster management, but it doesn't process your data.

The **compute plane** is where your data processing happens. Azure Databricks offers two types of compute planes, each designed for different use cases.



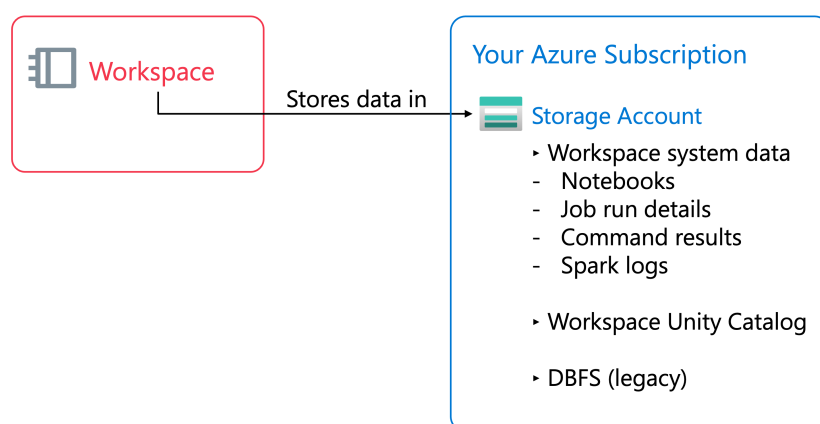
With **serverless compute**, your compute resources run in a serverless compute plane within your Azure Databricks account, not in your Azure subscription. Azure Databricks fully manages the infrastructure, automatically scaling resources based on your workload demands. The serverless compute plane includes network boundaries to isolate workspaces and additional security controls between clusters. This option simplifies operations because you don't need to manage virtual networks or compute resources yourself.

With **classic compute**, your compute resources run in your own Azure subscription in what's called the classic compute plane. Azure Databricks creates new compute resources within each workspace's virtual network in your subscription. This approach provides natural isolation because resources run in your subscription, giving you more control over networking and security configurations.

The separation between control and compute planes allows Azure Databricks to manage orchestration centrally while processing your data in isolated, secure environments.

Workspace storage

Each Azure Databricks workspace has an associated **workspace storage account** that resides in your Azure subscription. This storage account serves multiple purposes and contains different types of data.



The workspace storage account contains **workspace system data** generated as you use Azure Databricks features. This includes notebook revisions, job run details, command results, and Spark logs. The system uses this data to provide versioning, auditing, and troubleshooting capabilities.

If your workspace was enabled for Unity Catalog automatically, the workspace storage account also contains the default **workspace catalog**. All users in your workspace can create data assets in the default schema within this catalog, providing a convenient starting point for organizing data. Users access this data through Unity Catalog's governance layer and don't have direct access to the underlying storage, ensuring security and proper access control.

The workspace storage account may also contain **DBFS (Databricks File System)**, which is a distributed file system accessible under the `dbfs:/` namespace. DBFS root and DBFS mounts are legacy features. Storing and accessing data using DBFS root or DBFS mounts is a deprecated pattern. Instead, you should use Unity Catalog-managed tables and volumes for better governance and security.

Understanding where your data resides helps you implement appropriate security controls, such as enabling firewall support for your workspace storage account to limit access to authorized resources and networks only.

3. Understand Unity Catalog managed storage

<https://learn.microsoft.com/en-us/training/modules/understand-azure-databricks-architecture/3-understand-unity-catalog-managed-storage>

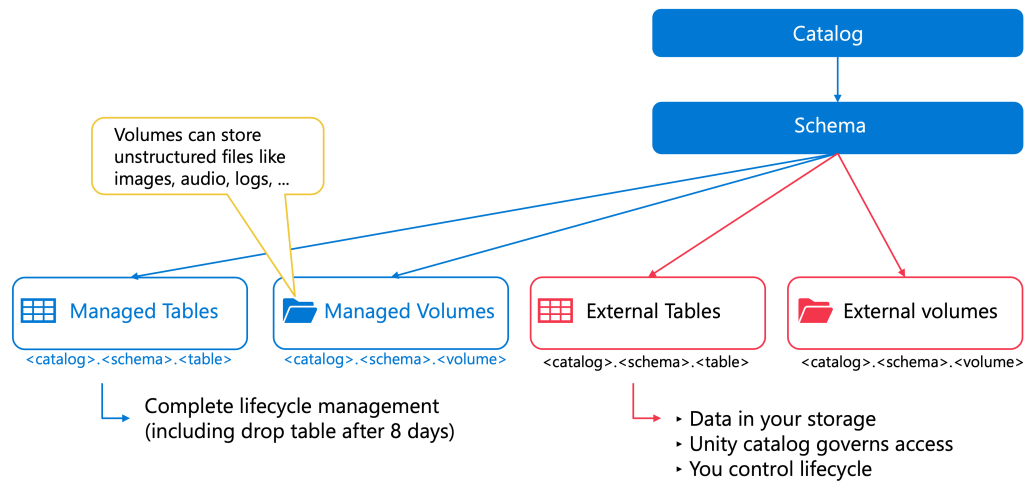
Understand Unity Catalog managed storage

Completed

Your organization likely has data governance policies that require specific types of data to reside in designated cloud storage locations. For example, you might need production data in one storage account and development data in another, or you might need to isolate sensitive customer data from operational data. **Unity Catalog managed storage** helps you meet these requirements while maintaining centralized governance and access control.

What is managed storage?

Managed storage in Unity Catalog refers to cloud storage locations where Unity Catalog stores data and metadata files for **managed tables** and **managed volumes**. Tables store structured data in a tabular format, while **volumes** provide governance for nontabular data files such as images, audio files, logs, or any other unstructured data that doesn't fit into a table schema.



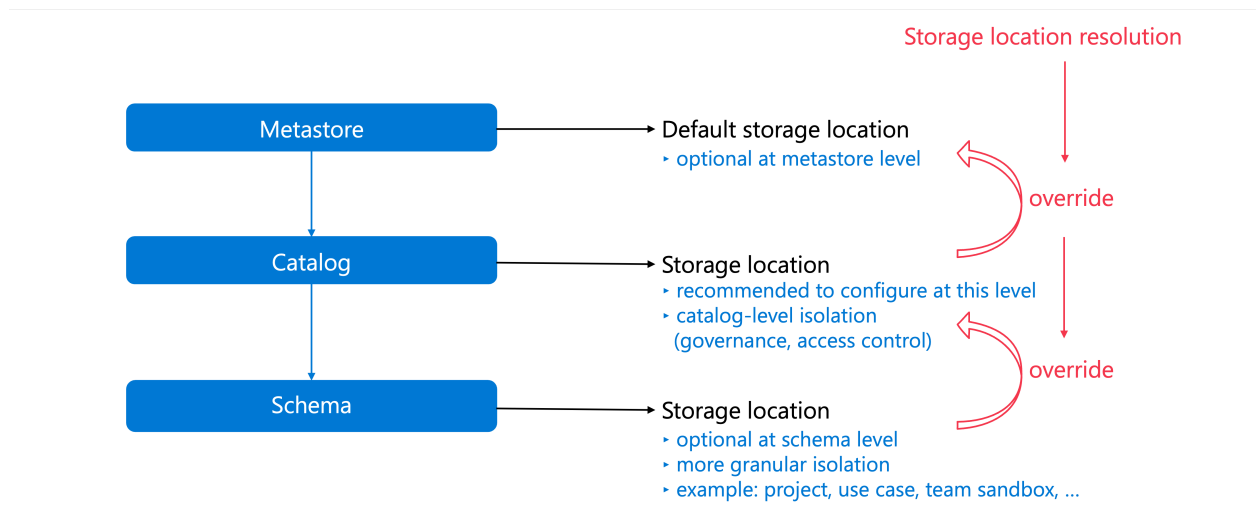
When you create a managed table or volume, Unity Catalog handles the complete lifecycle of the data—including where it's stored, how it's organized, and when it's deleted. This differs from **external tables** and volumes, where you manage the data lifecycle in your cloud storage, and Unity Catalog only governs access from Azure Databricks.

With managed storage, you specify a cloud storage path, and Unity Catalog takes care of the rest. This simplifies data management because you don't need to worry about file layouts or cleanup operations. When you drop a managed table, Unity Catalog automatically marks the underlying data files for deletion after eight days, along with removing the metadata.

Managed storage locations serve two key purposes. First, they provide **physical data isolation** by storing data in separate cloud storage containers or paths. Second, they align with your organizational structure and compliance requirements by letting you map catalogs and schemas to specific storage locations that meet regulatory or security policies.

How managed storage relates to Unity Catalog hierarchy

Unity Catalog organizes data using a **three-level namespace**: **catalog**, **schema**, and **table**. Managed storage locations can be defined at three corresponding levels in this hierarchy: the **metastore level**, the **catalog level**, and the **schema level**. Each level represents a different scope of data organization and isolation.



At the **metastore level**, you can optionally define a default storage location that serves as a fallback for any catalog or schema that doesn't have its own managed storage location. However, new workspaces enabled for Unity Catalog don't automatically include metastore-level storage, and Databricks recommends using **catalog-level storage** instead for better data isolation.

At the **catalog level**, you define storage locations that align with your organizational structure. Catalogs typically represent major organizational units, development lifecycle stages, or data classification categories. For example, you might create a catalog for production data with managed storage in a specific container, and another catalog for development data with storage in a different container. This catalog-level isolation is the recommended approach because it provides a clear boundary for data governance and access control.

At the **schema level**, you can define even more granular storage locations within a catalog. Schemas organize data into logical categories that are more specific than catalogs—typically representing individual projects, use cases, or team sandboxes. By assigning managed storage at the schema level, you can achieve **fine-grained data isolation** when needed.

Storage location resolution

When you create a managed table or volume, Unity Catalog determines where to store the data by following a **resolution hierarchy**. This hierarchy evaluates storage locations from the most specific (schema) to the most general (metastore).

Unity Catalog first checks whether the containing **schema** has a managed storage location. If a schema-level location exists, the data is stored there. This provides the most granular level of control over where your data resides.

If the schema doesn't have a managed storage location, Unity Catalog checks the containing **catalog**. If a catalog-level location exists, the data is stored in that catalog's managed storage location. This is the most common scenario and the recommended approach for most organizations.

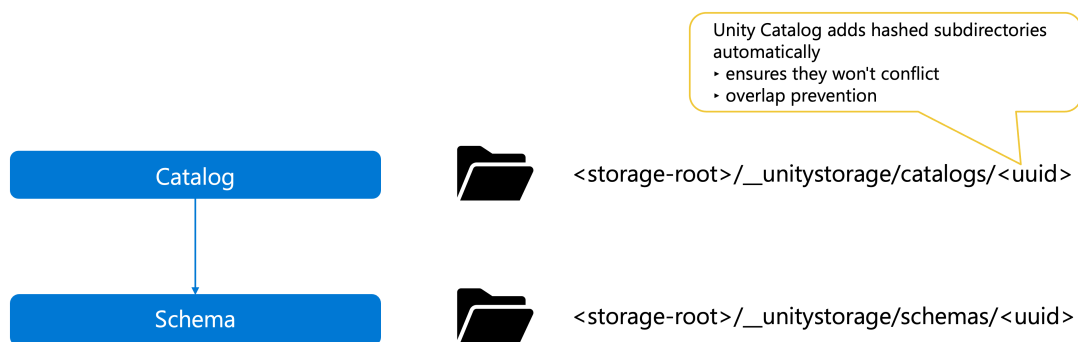
If neither the schema nor the catalog has a managed storage location, Unity Catalog falls back to the **metastore-level** storage location. If no metastore-level location exists either, you won't be able to

create managed tables or volumes, and you'll need to configure a managed storage location at one of these levels first.

This hierarchy gives you flexibility in how you organize and isolate data. You can start with catalog-level storage for most use cases and add schema-level storage when you need more granular control.

Storage root and storage location

When you define managed storage locations for catalogs or schemas, you specify a cloud storage path that Unity Catalog uses as the **storage root**. However, Unity Catalog doesn't store data directly in this path. Instead, it **automatically adds hashed subdirectories** to ensure each catalog and schema has a unique location, even if multiple objects share the same storage root.



For catalogs, Unity Catalog appends subdirectories in the format `__unitystorage/catalogs/<uuid>`, where `<uuid>` is a unique identifier. For schemas, it uses `__unitystorage/schemas/<uuid>`. The complete path—the storage root plus these automatically generated subdirectories—becomes the **storage location** where managed tables and volumes actually store their data.

This automatic path management provides important benefits. You can configure multiple catalogs or schemas with the same base storage root, and Unity Catalog ensures they won't conflict by creating distinct subdirectories for each. You don't need to manually organize or manage these paths—Unity Catalog handles this automatically while maintaining the data isolation and governance you need.

Unity Catalog also enforces **overlap prevention rules** to maintain data governance integrity. When you create a catalog or schema with a managed storage location, Unity Catalog validates that the storage path doesn't overlap with other managed storage locations, external tables, or external volumes. This prevents conflicts and ensures clear boundaries between different data assets. If you attempt to create a managed storage location that overlaps with an existing location, Unity Catalog will reject the operation to protect your data governance model.

4. Understand external storage

Understand external storage

Completed

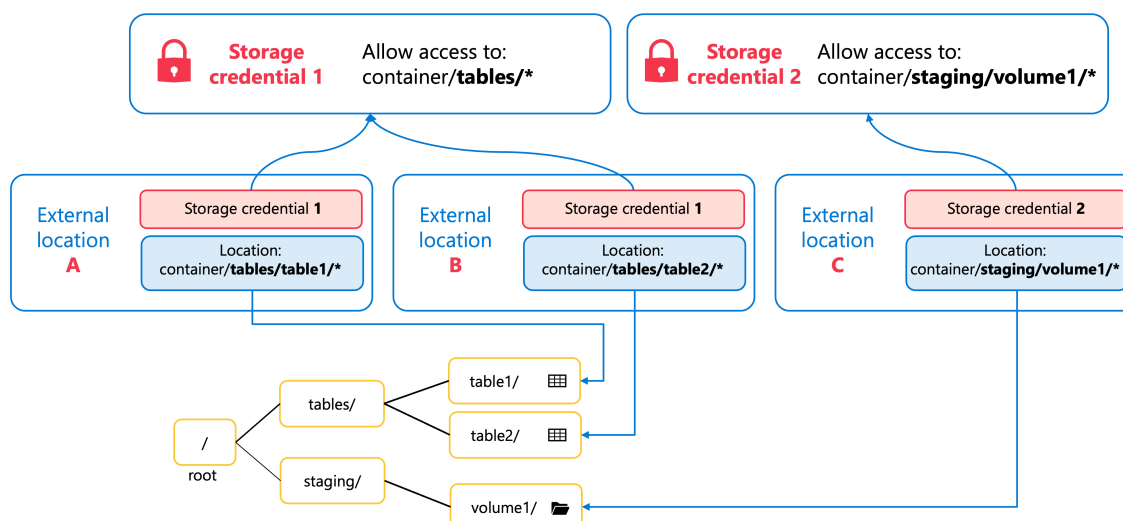
As a data engineer, you need to connect Azure Databricks to data stored in cloud storage locations like Azure Data Lake Storage containers. You might have production data in one storage account and development data in another, or you might need to work with data that's managed by teams outside of Databricks. **External locations** in Unity Catalog allow you to securely connect cloud storage to your workspaces while maintaining governance and access control.

What are external locations?

External locations are Unity Catalog objects that define secure connections to cloud storage. Each external location combines two components: a cloud storage path and a storage credential that authorizes access to that path.

A **storage credential** represents an authentication mechanism, such as an Azure managed identity or service principal. The storage credential provides the authentication required to access your cloud storage. Storage credentials are defined once in Unity Catalog and reference the identity configured in your Azure tenant.

An external location specifies which storage credential to use and which **cloud storage path** to access. This separation between credentials and locations allows one storage credential to be referenced by multiple external locations if they all access storage in the same security boundary.



External locations can reference storage in Azure Data Lake Storage containers, AWS S3 buckets (read-only), or Cloudflare R2 buckets. For Azure Databricks workloads, you'll typically use Azure Data Lake Storage containers because they integrate natively with Unity Catalog and support both read and write operations.

Why do we need external locations?

External locations serve two purposes in Unity Catalog: they enable you to work with external data assets, and they allow you to define managed storage locations at different levels of your data hierarchy.

External tables and volumes let you work with data that's stored and managed outside of Unity Catalog. When you create an external table, you're essentially registering existing data in cloud storage so that Unity Catalog can govern access to it. The data files remain in their original location, managed by your cloud provider or other data platforms. This approach is useful when you have large amounts of existing data, when other systems also need to access the data, or when you want to maintain control over the data lifecycle outside of Databricks.

Managed storage locations use external locations to define where Unity Catalog should store managed tables and volumes. Even though the data is managed by Unity Catalog, it still needs to reside in cloud storage that you own. An external location specifies the storage path, which is then assigned as the managed storage location for a catalog or schema. This allows you to determine where your data physically resides while Unity Catalog handles the data lifecycle.

The key difference lies in who manages the data. With external tables and volumes, you manage the data files and Unity Catalog governs access. With managed storage locations, Unity Catalog manages both the data lifecycle and access control, but stores the data in a location you specify through an external location.

Aspect	External Tables/Volumes	Managed Storage Locations
Data Management	You manage the data files	Unity Catalog manages the data
Data Location	Data remains in original location	Data stored in location you specify
Use Case	Existing or shared data	New data managed by Unity Catalog

Understand storage credentials

Before you can create external locations, you need **storage credentials**—Unity Catalog securable objects that encapsulate the authentication mechanism for accessing cloud storage. Storage credentials centralize credential management. Instead of handling authentication in notebooks or queries, you define credentials once in Unity Catalog and reference them across multiple external locations.

Azure managed identities are the recommended authentication mechanism. A managed identity is an Azure resource that provides an identity for applications connecting to resources that support

Microsoft Entra ID authentication. Azure automatically manages the credential lifecycle—no passwords, no secrets to rotate, and support for storage accounts protected by network firewalls.

Service principals are a legacy alternative. They require you to create an application identity in Microsoft Entra ID and manage client secrets manually, including periodic rotation. Managed identities are preferred for these reasons.

Storage credentials are metastore-level objects available to all attached workspaces. Only users with appropriate privileges can use them to create external locations.

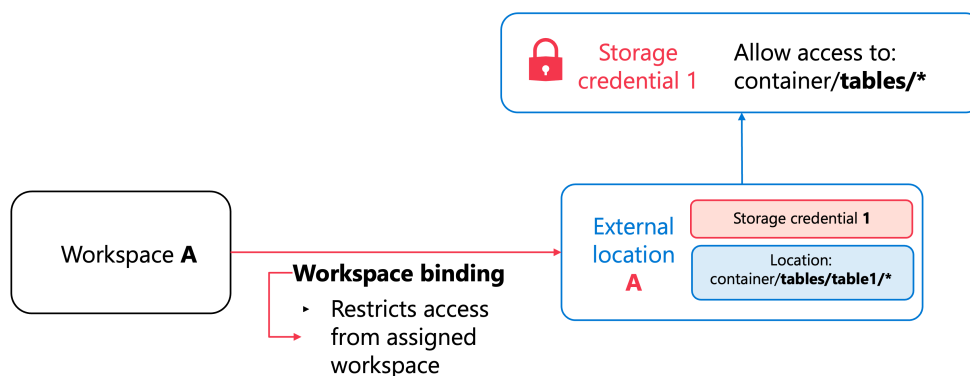
Note

A detailed walkthrough of creating and configuring external storage will be covered in a subsequent module.

Workspace binding for external locations

By default, an external location is accessible from all workspaces attached to your metastore. This means that any user with appropriate privileges can use the external location from any workspace. However, you might want to restrict access to specific workspaces to align with your organization's security boundaries or data governance policies.

Workspace binding restricts an external location to designated workspaces. When enabled, users can only access the external location from assigned workspaces, regardless of their Unity Catalog privileges. This creates an additional layer of access control beyond user-level permissions.



Workspace binding is relevant when workspaces represent different environments (production versus development) or when compliance requirements mandate that certain data remains accessible only from specific compute environments. For example, production data can be bound exclusively to production workspaces, preventing access from development environments even for privileged users.

5. Understand default storage (serverless compute)

Understand default storage (serverless compute)

Completed

When you deploy Azure Databricks workspaces, you typically need to configure cloud storage accounts, set up access credentials, and manage storage permissions. **Default storage** in Azure Databricks simplifies this process by providing ready-to-use, fully managed storage that's automatically available in serverless workspaces.

What is default storage?

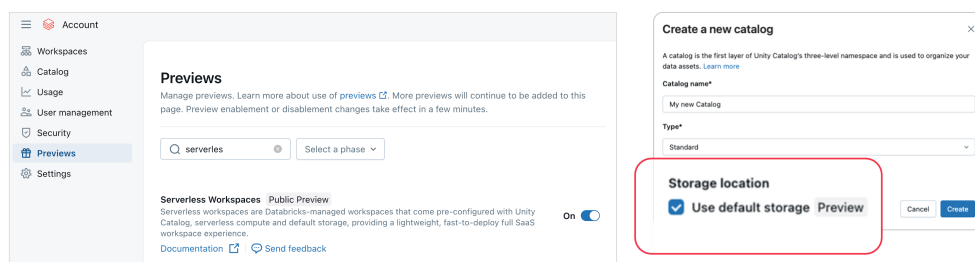
Default storage is a fully managed object storage platform built into Azure Databricks. It provides immediate storage capabilities without requiring you to configure external cloud storage accounts or manage access credentials.

This storage platform serves multiple purposes within your Azure Databricks environment. It stores workspace-level artifacts, control plane metadata, and data for catalogs you create. Unlike external storage where you maintain separate cloud storage accounts, default storage is managed entirely by Azure Databricks.

When you create a serverless workspace, Azure Databricks automatically provisions a default catalog that uses default storage. You can also create additional catalogs that use either default storage or your own cloud object storage, giving you flexibility in how you organize your data.

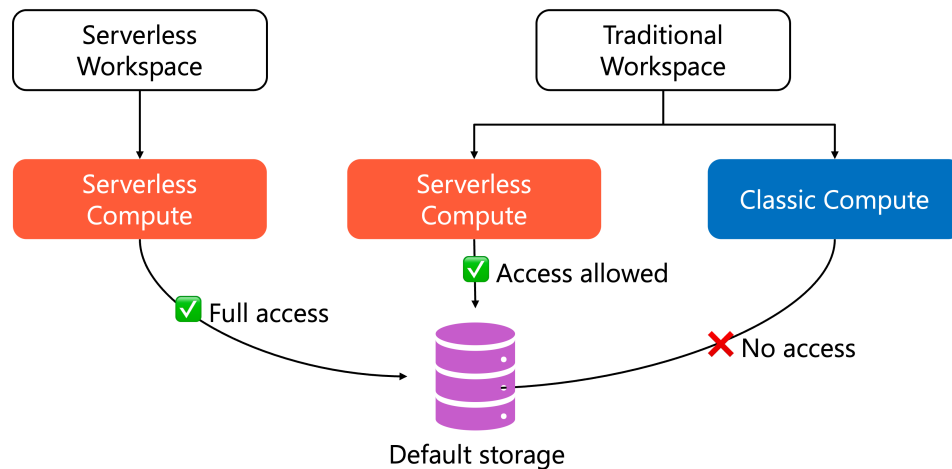
Where is default storage available?

Default storage is available exclusively in serverless workspaces.



Serverless workspaces use default storage for three key areas. First, they use it for internal workspace operations and control plane metadata. Second, they store workspace-level files and artifacts there. Third, catalogs you create can use default storage to store managed tables and volumes.

Traditional workspaces can access catalogs stored in default storage, but only when using serverless compute. This means you can share data across workspace types while maintaining the requirement for serverless compute when accessing default storage.



Default storage benefits and considerations

The following table summarizes the key benefits and considerations when using default storage:

Feature	Benefits	Considerations
Configuration and setup	No need to create separate cloud storage accounts, configure access credentials, or manage storage permissions. Azure Databricks handles infrastructure automatically, enabling immediate data access.	Only available in serverless workspaces. Not suitable for organizations that require full control over storage infrastructure.
Compute requirements	Serverless compute provides instant, scalable resources without provisioning. Consistent compute model across all default storage access.	All access requires serverless compute. Classic compute clusters cannot read from or write to catalogs on default storage. Existing classic compute workloads need migration.
Governance and security	Catalogs integrate with Unity Catalog's privilege model. Manage access using SQL GRANT statements instead of cloud storage RBAC (Role Based Access Control), creating a consistent security model across all data assets.	Access control limited to Unity Catalog mechanisms. Organizations requiring cloud-native RBAC integration should use external storage.
Workspace isolation	Catalogs are only accessible from the workspace where they were created by	Cross-workspace access requires explicit catalog binding configuration. Plan

Feature	Benefits	Considerations
	default. Provides natural isolation for development, testing, and production environments.	workspace architecture carefully to ensure data accessibility where needed.
Data lifecycle management	Automatic cleanup of underlying data files when you drop managed tables or volumes. Prevents orphaned data and reduces storage costs over time.	Only applies to managed tables and volumes. External tables must still manage their own storage lifecycle.
External access	BI tools like Power BI and Tableau can access data through Azure Databricks ODBC and JDBC drivers.	External tools that directly read Delta Lake or Iceberg metadata files cannot access default storage. Direct file access not supported. External data pipelines that read files directly need external storage.

Default storage works best for new serverless workloads, development environments, and features that specifically require it. For production workloads with external access requirements or those using classic compute, external storage provides more flexibility.

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/understand-azure-databricks-architecture/6-knowledge-check>

Module assessment

Completed

7. Summary

<https://learn.microsoft.com/en-us/training/modules/understand-azure-databricks-architecture/7-summary>

Summary

Completed

Azure Databricks architecture provides a flexible foundation for data engineering workloads through its hierarchical organization and separation of **control and compute planes**. Throughout this module, you explored how accounts organize resources across workspaces and metastores, how **Unity Catalog** governs data centrally, and how different compute models serve specific needs. The **control plane** handles orchestration while **compute planes**—whether **serverless** or **classic**—process your data in isolated, secure environments.

Storage patterns in Azure Databricks have evolved to address different scenarios. **Default storage** simplifies development by providing fully managed storage in serverless workspaces without configuration overhead, though it requires serverless compute for access. **External locations** bridge Unity Catalog with your existing cloud storage, enabling governance over data managed outside Azure Databricks while maintaining security through storage credentials. **Unity Catalog managed storage** gives you control over data placement at catalog and schema levels while Unity Catalog handles the complete lifecycle, including automatic cleanup of managed tables and volumes.

Understanding these architectural components helps you design Azure Databricks environments that align with organizational requirements. The **account hierarchy** provides clear governance boundaries, the separation of planes enables secure scaling, and the storage options let you balance convenience with control. As you implement Azure Databricks solutions, consider how each component serves your specific use cases, evaluate security implications of different storage patterns, and leverage Unity Catalog's governance capabilities to maintain data quality and access control across your organization.