Databricks – Overview and Purpose

What is Databricks

→ Databricks is a unified data and AI platform founded by the creators of **Apache Spark**.
→ Behind the scenes, Databricks uses the **Apache Spark compute engine** to process large volumes of data efficiently.
→ It simplifies data engineering, data science, and machine learning workflows by providing a single, collaborative environment.

Why Databricks

→ Traditional data platforms face several challenges that make large-scale data handling complex, inefficient, and hard to maintain.
→ Databricks was developed to unify the modern data stack—bringing ingestion, ETL, orchestration, and governance into one managed platform.

Challenges with Traditional Data Platforms

Challenge Area	Problem Description
Multiple Tools	Managing separate tools for ingestion, ETL, orchestration, and governance increases complexity and integration difficulties.
Complex Infrastructure	Clusters must be manually configured, scaled, and maintained, which is time-consuming and error-prone.
Slow and Inefficient Pipelines	Traditional ETL tools often run sequentially, leading to performance bottlenecks and delays.

Lack of Collaboration	Teams work in isolation, making it difficult to share notebooks or code effectively.
Unreliable Data Lakes	Data lakes lack ACID transactions, schema enforcement, versioning, and audit history—resulting in broken pipelines and inconsistent data.
Vendor Lock-in	Relying on multiple proprietary tools limits flexibility and increases dependency on specific vendors.

How Databricks Solves These Challenges

Traditional Problem	Databricks Solution
Multiple disconnected tools	A unified ecosystem that includes ingestion, ETL, orchestration, and governance in one platform.
Manual infrastructure management	Fully managed and auto-scaling clusters for simplified maintenance.
Slow data processing	Apache Spark engine provides distributed and high-speed data processing.
Lack of team collaboration	Collaborative notebooks supporting Python, SQL, R, and Scala within a shared workspace.
Unreliable data lakes	Delta Lake introduces ACID transactions, schema enforcement, version control, and time travel.
Duplication between Data Lake and Warehouse	The Lakehouse architecture merges both into a single system, eliminating redundant ETL.

Databricks Ecosystem Overview

→ Databricks provides an integrated ecosystem that addresses every stage of the modern data lifecycle—from ingestion and transformation to orchestration, governance, and storage. This unified approach eliminates the need to manage multiple disjointed tools and ensures a seamless data workflow within a single environment.

→ **Lakeflow Connect** handles data ingestion by enabling easy connectivity to a wide variety of data sources. It simplifies and automates the process of bringing data into Databricks, supporting both batch and streaming ingestion.

→ For **data transformation and ETL**, Databricks uses **Lakeflow Declarative Pipelines (DLT)**. DLT allows users to define data pipelines in a declarative manner, automatically managing dependencies, monitoring data quality, and handling incremental updates efficiently.

→ **Lakeflow** Jobs is used for orchestration. It schedules, manages, and automates data workflows, ensuring that pipelines run in a timely and reliable manner. This replaces the need for external schedulers or workflow managers.

→ For data governance, **Unity Catalog** provides a centralized solution for managing data access, lineage, and compliance. It establishes consistent governance policies across all workspaces, enabling secure and organized access to data assets.

→ At the core of the ecosystem lies **Delta Lake**, which serves as the foundational storage and processing layer. Delta Lake ensures data reliability, consistency, and scalability by adding ACID transactions, schema enforcement, and version control on top of traditional data lake storage systems.

→ Together, these components form a cohesive platform that enables organizations to efficiently manage their entire data lifecycle within Databricks, from ingestion to analytics and machine learning.

The Data Lakehouse Concept

Databricks introduced the **Data Lakehouse** architecture to overcome the limitations of traditional data lakes and data warehouses.

A **data lake** provides scalable and cost-effective storage for large volumes of raw data but lacks reliability features such as ACID transactions and schema enforcement. A **data warehouse**, on the other hand, ensures reliability and performance but is expensive and less flexible for handling unstructured data.

To bridge this gap, Databricks creates a **wrapper called Delta Lake** on top of existing data lakes (such as Amazon S3, Azure Data Lake Gen2, or Google Cloud Storage).
This layer combines the benefits of both systems—**the scalability of a data lake** and **the reliability and performance of a data warehouse**.

In simple terms:
Delta Lake = Data Lake + Data Warehouse

Delta Lake enables ACID transactions, schema enforcement, versioning, and time travel capabilities directly on data lake storage, transforming it into a unified and reliable **Lakehouse** platform.

Modern Data Lakehouse Architecture

Cloud providers such as Azure, AWS, and GCP act as the foundational data lake layer. They provide scalable object storage (like Azure Data Lake Storage, AWS S3, or GCP Cloud Storage) where all raw and processed data resides.

On top of this storage layer sits Delta Lake, which enhances the basic cloud data lake by adding:

- ACID transactions
- Schema enforcement and evolution
- Data versioning and time travel
- Optimized performance for large-scale processing

Above Delta Lake, the Unity Catalog layer is used for:

- Centralized governance
- Fine-grained access control
- Metadata management
- Data lineage tracking

- Auditing and compliance

This combination forms the Lakehouse architecture, which brings together the flexibility of data lakes and the reliability of data warehouses.

Roles such as Data Engineers, Data Scientists, and Data Analysts work on top of this lakehouse stack.

Learning and Practicing Databricks

Databricks previously provided a Community Edition, but this has now been replaced by the Databricks Free Edition.
The Free Edition offers a better, more realistic workspace with support for modern features such as Unity Catalog.

To practice Databricks, you now have two choices:

Databricks Free Edition

- Permanent free access
- Ideal for beginners
- Includes Unity Catalog and notebooks
- Suitable for learning Delta Lake, SQL, Python, and governance concepts

Azure Databricks (via Azure Free Trial)

- Create an Azure account using any Outlook email
- Azure provides a 30-day free trial with $200 (≈ ₹17,000) in credits
- Allows you to create an Azure Databricks workspace for real cloud practice
- After 30 days, the account shifts to a pay-as-you-go model
- Best for understanding cloud deployment, storage integration, real clusters, and production workflows

Steps to Create a Free Azure Account

Step 1 - Create a Microsoft Outlook Account.
Step 2 - Create a Microsoft Azure Profile by filling in the basic information.
Step 3 - You would need bank card details(ideally credit card) for the “identity verification by card” step.

Step 4 - Once the account is verified, you will have to login and enter “portal.azure.com” in the URL bar to access the azure cloud account and services.

Key Points -
→ On creation of an account, a subscription is associated.
→ Every Subscription has a billing unit.
→ Each subscription has a logical unit, Resource Group (logical collection of resources belonging together as part of a project).
→ Every Service on Azure cloud is treated as a Resource(An object used to manage the Service) - Each Resource has to belong to a Resource Group.

Databricks Workspace Setup on Azure

To use Databricks through Azure, you need to create a Databricks Workspace, which requires some Azure resources. The steps below outline the correct setup pattern used in real-world projects.

Step 1: Create a Resource Group

A Resource Group (RG) acts as the folder where all project-related cloud resources will be stored.

Naming Convention:
<project>-<env>-<region>-rg

Example:
databricks-training-centralindia-rg

Once the Resource Group is created, all Azure resources for this training environment will be placed inside or associated with it.

Step 2: Create the Databricks Workspace

Search for Azure Databricks in the Azure portal and create a new workspace.

Workspace Naming Convention:
<project>-<env>-<region>-dbws

Example:
training-centralindia-dbws

This workspace is the UI where you will create clusters, notebooks, jobs, and all Databricks-related work.

Understanding Databricks Cost Structure

When you use Databricks, the cost is determined by two components:

1. Infrastructure Cost (Azure Cost)

- Databricks clusters run on Azure compute (VMs).
- If you create a 3-node cluster, you are actually running 3 Azure VMs.
- You pay for:
  - Size of VM (e.g., Standard_DS3_v2)
  - Number of nodes
  - Hours running

2. Software Cost (Databricks Cost)

- Databricks charges separately using DBUs (Databricks Units).
- DBUs depend on:
  - Cluster type (All-Purpose, Job, or Serverless)
  - Runtime (Standard, Premium, Pro)

So, the total cost =
(Azure VM cost) + (Databricks DBU cost)

Azure Free Trial Impact

- Azure gives 30 days free trial with $200 (≈ ₹17,000) credit.
- This covers Azure infrastructure cost for 30 days.
- Databricks cost (DBUs) is still charged separately, but Databricks Free Trial or Free Tier may reduce or eliminate most DBU costs depending on your region.

Important:
Creating a cluster in Databricks → automatically starts VMs in Azure → these VMs consume the free credits.

Backend Azure Resources for Databricks

When you create a Databricks workspace, Databricks automatically creates its own managed resource group for backend resources such as:

- Worker VMs
- Driver VM
- Networking components
- Storage for cluster logs

Where do these resources reside?

Azure automatically assigns backend VMs to a special resource group:

Resource Group Name Format:
databricks-rg-<random-string>

This is a managed resource group created by Databricks.
You should NOT manually modify resources inside this group.

Your primary Resource Group (the one you created) is only for:

- The Databricks workspace itself
- Other optional resources (Storage Accounts, Key Vaults, VNets, etc.)

Using Notebooks to Run Spark Code

To run Spark code in Databricks, you primarily work inside notebooks.
Notebooks support multiple languages such as:

- Python
- SQL
- Scala
- R

→ **DBFS** is a distributed file system built on top of cloud storage (Azure, AWS, GCP).
It behaves like a regular file system but internally stores files in cloud storage.

Compute Options in Databricks

To run any notebook or Spark code, you need to attach a compute resource.
Databricks offers two main types:

1. Standard/Classic Compute (Clusters)

- User-managed clusters
- You choose:
    - Node type
    - Number of nodes
    - Runtime version
- Backend Azure VMs run as driver and worker nodes

2. Serverless Compute (Auto-Managed)

If you open a notebook and don't attach a cluster, Databricks may automatically choose Serverless Compute, depending on your workspace and region availability.

Serverless Compute characteristics:

- No VM provisioning or management
- Starts instantly
- Optimized auto-scaling
- Ideal for ad-hoc queries, notebooks, and lightweight Spark operations
- Databricks manages the entire infrastructure

This makes it faster and simpler for beginners who just want to run Spark code without configuring clusters.

High-Level Architecture of Databricks

High-Level Overview

Databricks operates using a two-plane architecture:

1. Control Plane (Databricks Cloud Account — Managed by Databricks)
2. Data Plane / Compute Plane (Customer Cloud Account — Azure/AWS/GCP)

This ensures security, governance, and separation of customer data from Databricks-managed infrastructure.

1. Control Plane (Managed by Databricks)

This is the portion of Databricks fully controlled and hosted by Databricks.

What resides in the Control Plane?

- Metadata
    - Workspace metadata
    - Table metadata
    - Notebooks and job metadata
    - ACLs (Access Control Lists)
- Configurations
    - Cluster configuration templates
    - Notebook settings
    - Job information
    - User roles & permissions
- Web Application / UI

The URL you use to access Databricks:

https://adb-1578161282373452.12.azuredatabricks.net/

- Workspace ID
    - Example: 1578161282373452
    - Used internally to identify tenant and workspace

Key point:

→ No customer data is stored in the control plane.
Only metadata and configurations.

2. Data Plane / Compute Plane (Customer Cloud Account)

This is where your actual data and compute resources live.

What resides in the Data Plane?

- Your Customer Data
  - Files in ADLS / S3 / GCS
  - Delta tables
  - DBFS
- Classic Compute Clusters
  - VMs (Azure virtual machines)
  - Driver/worker nodes
  - Runtime execution happens here
- Networking components
  - VNETs
  - Subnets
  - Private endpoints

Key point:

→ All data processing happens in your own cloud account, not in Databricks' account.

So when you create a cluster:

- Azure provisions actual VMs inside your resource group (the databricks-managed RG)
- Spark jobs run inside these machines
- Data never leaves your cloud environment

Architecture Diagram

```
+---------------------------------------------------+

|      CONTROL PLANE (Databricks)     |

|---------------------------------------------------|

| - Workspace UI / Web App                  |

| - Notebooks metadata                        |

| - Cluster configs & job configs           |
| - ACLs & user permissions                |
| - Management, monitoring, orchestration|
+--------------------▲-----------------------------+
                      |
                      | Secure control channel
                      |

+-----------------------------▼-----------------------------+
 |       DATA PLANE / COMPUTE PLANE        |
 |       (Customer Cloud Account)                          |
 |------------------------------------------------------------|
 | - Customer data (ADLS / S3 / GCS / DBFS)    |
 | - Driver & Worker VMs (clusters)                  |
 | - Spark execution engines                        |
 | - Networking (VNET, subnets, NSGs)             |
 | - Delta Lake storage                                  |
 +------------------------------------------------------------+
```

Databricks Accounts, Workspaces, and Unity Catalog

1. Databricks Accounts and Workspaces

An organization typically operates under a single Databricks account, but it will almost always create multiple Databricks workspaces within that account. Each workspace acts as a separate development or operational environment.

There are several reasons why multiple workspaces are required:

- **Environment Separation:**
  Most organizations maintain separate workspaces for Development, Staging, and Production. This ensures that experimentation and testing activities do not interfere with critical production workloads.

- **Business Unit Isolation:**
  Different departments such as Sales, Finance, HR, or IT Analytics may require their own isolated workspace for operational and compliance purposes. Each department can manage its own compute, access policies, and governance without affecting others.

Workspaces therefore allow structured separation of workloads while remaining under the same central Databricks account.

2. Understanding Tables in Databricks

Every table in Databricks consists of two fundamental components:

Data

This represents the actual information stored in physical files, often in formats like Parquet or Delta.
Importantly, this data always resides in your own cloud storage account (Azure Data Lake, AWS S3, or Google Cloud Storage). Databricks does not physically store your data.

Metadata

Metadata contains information about your tables, such as schema definitions, column names, data types, file pointers, table properties, and more.
This metadata is stored in a Metastore, which acts as the cataloging and organizational system for tables.

3. The Legacy Hive Metastore

Before Unity Catalog became the default standard, each Databricks workspace was automatically provisioned with its own Hive Metastore. This resulted in metadata being isolated per workspace.

Hive Metastore used a 2-level namespace structure, such as:

schema.table
retaildb.orders

Although widely used, Hive Metastore had several significant limitations:

- It lacks fine-grained access control, making it difficult to restrict permissions at a detailed level.
- It does not provide built-in lineage, making it hard to track data movement across pipelines.
- It lacks auditing capabilities, restricting the ability to monitor and verify user actions.
- It has minimal governance support, which complicates compliance in enterprise environments.
- It only supports structured data, leaving unstructured assets ungoverned.
- It cannot be shared across multiple workspaces, leading to duplicated metadata and inconsistent governance.

Because of these issues, Hive Metastore is no longer suitable for modern enterprise use cases.

4. Introduction of Unity Catalog

Databricks has now moved towards Unity Catalog as the unified metadata and governance system across the platform.
Modern workspaces are Unity Catalog–enabled by default.

When a new workspace is created:

- A Unity Catalog Metastore is provisioned automatically.
- A default catalog is created, named after the workspace.
- The workspace receives an improved metadata and governance framework.

Unity Catalog introduces a 3-level namespace structure:

catalog.schema.table

This extended structure enhances organization, discoverability, and governance across multiple teams and environments.

Unity Catalog governs far more than just tables. It manages:

- Tables
- Views
- Schemas
- Volumes (unstructured data)
- Functions
- Machine Learning models

Unity Catalog therefore provides unified governance over the entire data and AI lifecycle.

5. Unity Catalog Metastore Behavior

Regional Architecture

A Metastore in Unity Catalog is regional. This means:

- The first workspace you create in a region (e.g., Central India) triggers the creation of a single Metastore for that region.
- Any subsequent workspaces created in the same region will be automatically attached to the same Metastore.
- If you later create a workspace in a different region (e.g., US East), Databricks will provision a new Metastore for that region.

Why Regional Metastores?

Since metadata includes security configurations, permissions, and governance settings, keeping it within the same region improves both performance and compliance.

Association Rules

- Can a workspace use more than one Metastore?
  No. A workspace can be attached to only one Metastore at a time.

- Can a Metastore be connected to multiple workspaces?
  Yes, but only if those workspaces are in the same region.

Best Practice Recommendation

For each region, create one Metastore only and attach all region-level workspaces (Dev, Stage, Prod) to this single Metastore.
This ensures centralized governance and eliminates duplication.

6. Default Catalogs in New Workspaces

Whenever a Unity Catalog–enabled workspace is created:

- A catalog with the same name as the workspace is automatically generated.
- Inside this catalog, you will find:
    - A default schema, where user-created tables typically go.
    - An information_schema, containing system-level metadata about tables, columns, and other objects.

These catalogs act as the foundational structure for organizing data assets.

7. Managed Resource Group Note (Azure)

→ When Databricks is deployed on Azure, each workspace automatically creates its own managed resource group.
→ Databricks does not allow this managed resource group to be shared among multiple workspaces.
→ This is because each workspace requires isolated compute, network, and storage resources.

8. Accessing the Databricks Account Console

Administrative tasks such as managing workspaces, metastores, users, and account-level settings are performed via the Databricks Account Console:

https://accounts.azuredatabricks.net/

This is accessible only to account admins, not regular workspace users.

Databricks Tables and Storage Access

There are three types of tables in Databricks: **managed tables**, **external tables**, and **foreign tables**. Each of these differs in how the data files are stored and how the metadata is managed in Unity Catalog.

A **managed table** allows Databricks to manage both data and metadata. The metadata is stored in the Unity Catalog metastore, while the data files are stored in a storage location that Unity Catalog manages automatically. When data is inserted into a managed table, Databricks writes the files into the metastore's managed storage path. Dropping a managed table removes both metadata and the underlying data files.

An **external table** stores metadata in the metastore but keeps the data files in a location that you control in your cloud storage account. When you drop an external table, the metadata is removed, but the data files remain intact. To use external tables, you must set up a storage location that Databricks can access using a storage credential and an external location.

A **foreign table** references data stored in an external system such as a relational database (via JDBC). Metadata resides in the metastore, while the data continues to remain in the external system. Querying the table retrieves the data at runtime, and deleting the table removes only the metadata in Databricks.

Unity Catalog oversees the metadata layer. It contains metastores, catalogs, schemas, and tables. A metastore stores metadata and governance configurations. A catalog organizes schemas and tables, while schemas organize tables within catalogs. In Unity Catalog, tables use a 3-level namespace:

catalog.schema.table

Managed Tables

A managed table is the default type of table created in Databricks.

Key Characteristics

- Databricks manages both the table metadata and the physical data files.
- When you run:

```
spark.sql(""
CREATE TABLE IF NOT EXISTS workspace.default.students (
id INT,
name STRING,
score FLOAT
);
```

or upload a dataset without specifying a location, Databricks automatically stores the data inside the workspace-managed storage.

If you drop the table - Both metadata and actual data files are permanently deleted.

When to Use Managed Tables

- When you want Databricks to fully handle data storage.
- Ideal for internal datasets used within the workspace.
- Best choice when you don't need fine-grained control over file locations.

Managed Tables With External Location (Unity Catalog)

These tables are still managed tables, but their underlying storage is an external location instead of the internal workspace-managed storage.

How it works

- You first create:
    1. **Storage Credential**
       (gives UC permission to access cloud storage)
    2. **External Location**
       (a named abstraction pointing to a folder in S3/ADLS/GCS)
- Then Databricks stores managed table data *inside that external location*.

External Tables:

For external tables, you choose the storage account, container, and folder manually. For example:

- Storage account: databricksdemo101sa
- Container: unitycatalog
- Folder: catalog

Databricks requires explicit permission to access this storage. In Azure, this is done by creating an **Access Connector for Azure Databricks**, which acts as a managed identity. After creating the connector, it must be granted access to the target storage account (e.g., Storage Blob Data Contributor/Owner).

Once access is granted, the next step is to register this identity inside Databricks using a **Storage Credential**. The storage credential represents the authentication method Databricks will use to read and write data in your storage account.

After the credential is created, you define an **External Location**, which maps a cloud path (e.g., abfss://.../catalog) to the storage credential. This allows Unity Catalog to perform fine-grained access control, ensuring that only authorized users can read/write to that path.

Managed Tables vs External Tables

Managed Tables (Unity Catalog)

- Unity Catalog manages both metadata and data files.
- You create them without specifying a LOCATION.
- Data is stored automatically in the catalog/schema storage location.
- Default format is Delta Lake (unless specified otherwise).
- Fully optimized and maintained by Databricks:
    - Auto-optimize
    - Auto-compaction
    - Better performance
    - Reduced storage cost
    - Improved caching
- Supports both Delta Lake and Iceberg formats.
- Best for workloads fully inside Databricks.
- When you drop a managed table:

    - Metadata is deleted immediately.
    - Data is retained for 7 days (safe delete).
    - Can be recovered using UNDROP TABLE during retention period.
- Recommended when:
    - Data lifecycle should be Databricks-managed.
    - You want maximum governance and performance.
    - Fine-grained access control is required.
    - No external system needs raw file access.

External Tables (Unity Catalog)

- Metadata is stored in Unity Catalog, but data files remain outside in your cloud storage.
- Requires LOCATION '<path>' while creating the table.
- Data lifecycle is not managed by Databricks.
- When dropped:
    - Only metadata is removed.
    - Data files remain untouched.
- Useful when:
    - You already have data files in ADLS/S3/GCS that must be registered as tables.
    - External systems (Spark, Azure Data Factory, Synapse, etc.) need direct file access.
    - You want to keep raw data separate from Databricks-managed storage.
- Supported formats include:
    - Delta, CSV, JSON, Parquet, ORC, Avro, Text
- Governance behavior:
    - Unity Catalog controls table-level permissions within Databricks.
    - External systems bypass Unity Catalog; rely on cloud ACLs instead.

External Locations & Storage Credentials (Quick Points)

Storage Credential

- A secure object that stores permissions for Databricks to access cloud storage.
- Example: adls_raw_storage_cred.

External Location

- A cloud storage path that has been approved for table creation or access.
- Must be bound to a Storage Credential.
- Example:

```
CREATE EXTERNAL LOCATION adls_raw_storage_catalog_ext_loc
URL
'abfss://unitycatalog@databricksdemo101sa.dfs.core.windows.net/catalog'
WITH (CREDENTIAL adls_raw_storage_cred);
```

Choosing Between Managed vs External Tables

Use Managed Table when:

- You want maximum optimization (auto-optimize, compaction, indexing).
- You want full governance and catalog-level security.
- Data is primarily consumed inside Databricks.
- You require 7-day safe-delete recovery.

Use External Table when:

- Data already exists in storage and shouldn't move.
- Multiple tools outside Databricks also access the data.
- You want to control storage layout yourself.
- You work with non-Delta data formats like CSV/JSON.