**GitHub Actions**

Imagine you’re working on a coding project and want to automate tasks like:

Running tests when you update code  
Deploying your website automatically.Sending notifications when something changes

GitHub Actions helps you do all of this automatically! Instead of doing things manually, GitHub Actions writes rules that tell GitHub:  
*"If this happens, do that!"*

**Why is GitHub Actions Needed?**

Saves Time – No need to manually test or deploy code.  
Reduces Errors – Automates tasks, so you don’t forget anything.  
Works with Any Project – Python, Java, Node.js, and more!  
Great for Teams – Everyone gets updates when something changes.

**When is GitHub Actions Used?**

For Developers**:** Run tests every time you change code.  
For Websites: Automatically deploy a website when you push updates.  
For Apps: Build and release an app automatically.  
For Automation**:** Do repetitive tasks without human effort.

**How GitHub Actions Works (Step by Step)**

Trigger (Event happens) – You push code, open a pull request, etc.  
Runs a Workflow – A script that does something (e.g., test or deploy).  
Executes Jobs & Actions – Runs commands to complete the task.

Example Workflow YAML File (this runs a Python test when code is pushed):

name: Run Python Tests

on: push

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v3

with:

python-version: '3.10'

- name: Install dependencies

run: pip install -r requirements.txt

- name: Run tests

run: pytest

**What happens here?**

* **Whenever you push code**, GitHub automatically runs the tests

**Benefits of GitHub Actions**

✔ **Easy to set up** – Just a simple YAML file  
✔ **Fully automated** – No manual work needed  
✔ **Works with any programming language**  
✔ **Improves project quality** – Catches bugs early

**Databricks asset bundles:**

**What are Databricks Asset Bundles?**

* Databricks Asset Bundles are a way to manage and deploy your Databricks projects using infrastructure-as-code (IaC).
* They help organize and deploy all parts of your project in a consistent and reproducible way.
* Think of it as a package that contains everything your project needs to work correctly in Databricks.

**What does a Databricks Asset Bundle include?**

1. Cloud Infrastructure and Workspace Configurations – This defines the setup of your cloud environment and workspace.
2. Source Files – These are the code files (like notebooks, Python files) that have the logic for your project.
3. Databricks Resources – This includes configurations for Databricks jobs, Delta Live Tables, MLflow experiments, registered models, and other Databricks services.
4. Unit and Integration Tests – Tests to make sure your project works as expected.

**Why Do We Use Databricks Asset Bundles?**

1. Consistency and Reproducibility:
   * Bundles ensure that everything (code, configurations, resources) is packaged together, making sure the same setup can be deployed again in different environments (dev, test, prod).
2. Simplified Deployment:
   * Instead of manually configuring resources and settings, all assets are in one package, making deployment simpler and reducing errors.
3. Version Control:
   * Databricks Asset Bundles allow you to track changes to your code and resources. If something breaks, you can easily roll back to a previous working version.
4. Environment Isolation:
   * You can deploy the same bundle to different environments (like development, testing, or production) while keeping each environment isolated from the others.

**How Do Databricks Asset Bundles Work?**

1. Bundle Metadata (YAML Files):
   * You define the content of the bundle using YAML files, which specify what resources and configurations are included in the project.
2. Creating YAML Files:
   * You can write these YAML files manually or use a template provided by Databricks to help generate them.
3. Using Databricks CLI:
   * Once the YAML file is ready, you can use the Databricks CLI (Command Line Interface) to:
     + Validate the bundle.
     + Deploy it to Databricks.
     + Run the bundle directly from the CLI or Databricks UI.
4. Running the Bundle:
   * You can run the bundle from:
     + An IDE (Integrated Development Environment) like VSCode.
     + A terminal (command line).
     + Directly within the Databricks interface.

**Metadata-Driven Medallion Framework[MDMF]:**

A diagram of a data processing process

AI-generated content may be incorrect.

**1. Data Ingestion**

* **Purpose:** Collects data from different source systems.
* **Sources:**
  + **Supported Source Systems** (SQL Databases, Cloud Storage, etc.).
  + **Non-Supported Data Sources** (which require additional handling).
* **How Data Moves:**
  + **Push Method:** Some sources push data into the **Drop Zone**.
  + **Pull Method:** Other sources (like SQL Databases) allow data to be pulled into the **Landing Zone**.
* **Formats:** Data is ingested in its **native format** (raw, unprocessed).

**2. Data Standardization**

* **Purpose:** Converts raw data into a **unified format** (Parquet Format).
* **Process:**
  + Data from the **Landing Zone** is transformed into a **consistent format** before further processing.
  + This ensures compatibility for analytics and processing in later stages.
* **Tools Used:** Databricks and Delta Lake.

**3. Data Segregation**

* **Purpose:** Divides data based on **Personally Identifiable Information (PII)** levels.
* **Storage:**
  + **Medallion Architecture Layers (Delta Lake)**
    - **Bronze Layer:** Stores raw data, split into:
      * bronze\_classified (Sensitive data with PII).
      * bronze\_internal (Non-sensitive internal data).
    - **Silver Layer:** Cleansed and transformed data.
    - **Gold Layer:** Finalized, enriched data for analytics and reporting.
* **Uses Unity Catalog** to **classify and manage** data access.

**Unity Catalog**

* **Hive Metastore (HMS)**: A metadata repository storing information about tables, databases, partitions, and schemas in **Databricks.**
* **Unity Catalog (UC)**: A unified governance solution for **centralized metadata management, enhanced security, and data lineage tracking.**
* **Why Unity Catalog?** It provides **multi-workspace governance, advanced security features, data lineage, and support for unstructured data** compared to Hive Metastore.

**2. Availability**

* **Hive Metastore:** Available **by default** in all **Databricks** workspaces.
* **Unity Catalog:** **Not available by default**—needs to be **enabled manually.**

**3. Scope & Hierarchy Comparison**

**Hive Metastore Scope:**

* **Limited to a single workspace or cluster.**
* Basic **SQL-based access control.**
* **Manages only tables, views, and partitions.**

**Hierarchy:**

1. **Hive Metastore:** Stores all metadata.
2. **Database:** A namespace for organizing tables.
3. **Table:** Stores structured data.

**Unity Catalog Scope:**

* **Supports multiple workspaces, multi-cloud, and regional governance.**
* **More structured with fine-grained access control.**
* **Manages tables, schemas, catalogs, and non-tabular data.**

**Hierarchy:**

1. **Metastore:** Holds metadata for all databases & tables.
2. **Catalog:** Logical grouping of schemas & tables.
3. **Schema:** Similar to "database" in Hive Metastore.
4. **Table/View:** Stores structured datasets.

**4. Storage Access**

**Hive Metastore Storage**

* **Stores tables** at /user/hive/warehouse/database/table by default.
* For external storage (ADLS, S3), **manual mounting is required.**

**Unity Catalog Storage**

* Uses **Storage Credentials** (Managed Identity/Service Principal).
* **External Locations** link storage like **ADLS, S3.**
* **No need for mounting—direct access to cloud storage.**

**5. Data Lineage (Tracking Data Flow)**

**Hive Metastore**

* **No built-in lineage tracking.**
* **Users must manually track source-to-target table relationships.**

**Unity Catalog**

* **Captures runtime data lineage automatically.**
* Supports **all programming languages (SQL, Python, Scala, R).**
* **Tracks column-level lineage, queries, dashboards, and jobs.**
* **Retains lineage data for 1 year.**

**6. Handling Different Data Formats**

**Hive Metastore**

* **Only supports structured tabular data.**

**Unity Catalog**

* Introduces **Volumes**—governs **structured, semi-structured, and unstructured data.**
* Supports **files, images, and other non-tabular datasets.**

**7. Sensitive Data Management & Security**

**Hive Metastore**

* **Uses Role-Based Access Control (RBAC).**
* **Column-level access control only.**

**Unity Catalog**

* **Row-level & column-level access control.**
* **Dynamic Data Masking**—hides sensitive data based on user roles.
* **Stronger security for compliance (GDPR, HIPAA, etc.).**

**8. Checking if Unity Catalog is Enabled**

**1. Databricks UI Method**

* **Go to "Data" tab** in Databricks.
* If **"Catalogs"** and **"Delta Sharing"** are visible → **Unity Catalog is enabled.**

**2. SQL Command Method**

* Run SHOW CATALOGS:
  + If only hive\_metastore appears → **Hive Metastore is used.**
  + If multiple catalogs appear → **Unity Catalog is enabled.**

**9. Conclusion**

* **Hive Metastore** is a **basic metadata store** but lacks **modern governance features.**
* **Unity Catalog** provides:
  + **Multi-workspace metadata management.**
  + **Built-in data lineage & access control.**
  + **Better security (fine-grained access + dynamic masking).**
  + **Support for both structured & unstructured data.**
* **For enterprise-scale governance, Unity Catalog is the best choice.**

**Unity Catalog**

Unity Catalog is a centralized data governance solution in Databricks that helps manage data access, security, and auditing across multiple workspaces.

**Why Use Unity Catalog?**

**Centralized Access Control** – Manage permissions in one place.  
**Column & Row-Level Security** – Restrict sensitive data.  
**Data Lineage** – Track data flow and usage.  
**Works Across Cloud Storage** – Supports S3, ADLS, GCS.

**Key Features of Unity Catalog**

🔹 **Catalog → Schema → Tables** (Organized like a database)  
🔹 **Fine-Grained Access Control** (Users, groups, service accounts)  
🔹 **Data Lineage** (Track where data comes from and how it’s used)  
🔹 **Secure Data Sharing** (Share data across teams securely)  
🔹 **Works Across Multiple Clouds** (AWS, Azure, GCP)

**Example: How Data is Organized in Unity Catalog**

📂 **Catalog** → 📂 **Schema** → 📊 **Table/View**

USE CATALOG my\_catalog;

USE SCHEMA sales;

SELECT \* FROM transactions;

**How to Enable Unity Catalog?**

**Create a Metastore** (Stores all metadata)  
**Attach Workspaces** (Connect to Databricks)  
**Grant Permissions** (Manage access for users & roles)  
**Start Querying Data!**

**Why is Unity Catalog Important?**

Makes **data governance easy** across multiple teams  
Helps with **compliance** (GDPR, HIPAA, etc.)  
Supports **multi-cloud environments**