**Databricks Unity Catalog Implementation @ High Level**

1. **Overview**

**Databricks Unity Catalog** is a unified governance solution for all data assets in your Databricks workspace. It provides:

**1.Centralized metadata management**:

A single metastore to catalog and organize all data assets across your Databricks environment.

* Data assets across your Databricks environment include tables, views, files, functions, models, and other structured or unstructured datasets registered in the metastore.

**2.Fine-grained access controls (table, column, row)**:

precisely control who can access specific data at any level of detail.

* Specific data at any level of detail refers to entire tables, individual columns, or filtered rows containing sensitive or business-critical information

**3.Data lineage and audit logging**:

Automatically track data flow and user actions for transparency and compliance.

* Databricks tracks data flow through lineage of queries and transformations, and logs user actions like reads, writes, permission changes, and schema modifications.

**4.Consistent security model across workspaces:**

Enforce uniform access policies and governance in every workspace.

* Uniform access policies and governance enforce consistent permissions, security rules, and compliance standards across all workspaces and data assets.

**2. Step-by-Step Navigation Guide**

Below is a clear sequence of steps for configuring and using Unity Catalog:

**2.1 Prerequisites**

* Databricks Premium or Enterprise plan
* Metastore admin role assigned
* Workspace(s) attached to your account
* Databricks CLI configured

**2.2 Create the Unity Catalog Metastore**

1. **Open Databricks Admin Console**
   * Go to **Account Console** > **Data**.
2. **Create Metastore**
   * Click **Create Metastore**.
   * Provide:
     + Name
     + Storage root path (e.g., an S3 bucket or ADLS)
     + Region
3. **Configure Managed Storage Credential**
   * Use Databricks-generated IAM role or create your own.

**2.3 Assign Workspaces to the Metastore**

1. From the Metastore settings, click **Assign to Workspace**.
2. Select the workspace(s) to attach.
3. Confirm the assignment.

**2.4 Configure Access Control**

1. In **Data** tab, navigate to **Grants**.
2. Assign **Data Steward** and **Data Owner** roles.
3. Use SQL Grants:

sql

CopyEdit

GRANT USAGE ON CATALOG <catalog\_name> TO `group\_name`;

GRANT SELECT ON SCHEMA <schema\_name> TO `group\_name`;

**2.5 Create Catalogs, Schemas, and Tables**

1. **Create Catalog:**

sql

CopyEdit

CREATE CATALOG sales\_data;

1. **Create Schema:**

sql

CopyEdit

CREATE SCHEMA sales\_data.monthly\_reports;

1. **Create Table:**

sql

CopyEdit

CREATE TABLE sales\_data.monthly\_reports.june (

order\_id STRING,

amount DOUBLE

);

**2.6 Enable Data Lineage**

* Unity Catalog automatically captures lineage.
* Access via the **Data Explorer > Lineage** tab.

**2.7 Configure Row and Column-Level Security**

1. **Row Filter Example:**

sql

CopyEdit

CREATE OR REPLACE ROW FILTER filter\_region

AS (region = 'US');

sql

CopyEdit

ALTER TABLE sales\_data.monthly\_reports.june

SET ROW FILTER filter\_region;

1. **Column Mask Example:**

sql

CopyEdit

CREATE OR REPLACE FUNCTION mask\_ssn(ssn STRING)

RETURNS STRING

RETURN CONCAT('XXX-XX-', RIGHT(ssn, 4));

sql

CopyEdit

ALTER TABLE customer\_data

ALTER COLUMN ssn

SET MASK mask\_ssn;

**2.8 Monitor Audit Logs**

* Enable audit logs in your workspace.
* Integrate with cloud-native logging (AWS CloudTrail, Azure Monitor).

**3. Best Practices by Project Complexity**

**Simple Projects**

*(e.g., single catalog, small team)*

* Use one catalog to group all data.
* Start with **table-level permissions**.
* Keep schema naming consistent.
* Avoid over-segmentation.
* Document grants in a shared spreadsheet.

**Medium Projects**

*(e.g., multiple teams, multiple schemas)*

* Create separate catalogs per domain or business unit.
* Use **schemas** to isolate environments (dev, test, prod).
* Implement **column masking** for sensitive fields.
* Leverage **groups** rather than individual users for grants.
* Enforce naming conventions (<team>\_<purpose>\_<environment>).
* Automate grants with Terraform or Databricks CLI.

**Complex Projects**

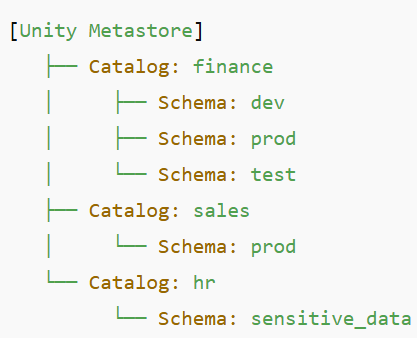
*(e.g., enterprise data mesh, multi-region)*

* Adopt a **catalog per domain model** (data mesh architecture).
* Implement **row-level security policies** for multi-tenant data.
* Enable **data lineage** and integrate with external catalogs (Purview, Glue).
* Standardize all permissions in Infrastructure-as-Code.
* Use CI/CD pipelines for schema/table creation.
* Periodically review and prune obsolete grants.
* Consider **schema evolution policies** for big data ingestion.

**4. Reference Architecture Diagram (Textual)**

**[Account Level]**

Yaml



**[Workspace Level]**

vbnet

CopyEdit

Workspace A:

Linked to Unity Metastore

Cluster with Unity Catalog enabled

Workspace B:

Linked to Unity Metastore

Cluster with Unity Catalog enabled

**5. Security & Governance**

* All access governed by the **Metastore Admins**.
* Leverage **SCIM** to sync user groups.
* Rotate credentials periodically.
* Enable **audit log forwarding**.
* Test permissions before production rollout.

**6. Operational Considerations**

* Monitor performance of catalogs and metadata queries.
* Plan storage in advance to avoid S3/ADLS permission conflicts.
* Version your schemas and track changes.

**7. Resources & Links**

* Unity Catalog Docs : https://docs.databricks.com/data-governance/unity-catalog/index.html
* Terraform Provider : ttps://registry.terraform.io/providers/databricks/databricks/latest/docs
* Databricks CLI : https://docs.databricks.com/dev-tools/cli/index.html

**Details for Reference below**

**2.1 Prerequisites – Detailed Steps**

|  |
| --- |
|  |
| **2.1 Prerequisites – Detailed Steps**  Below are the actions you must complete before setting up **Unity Catalog**:  **1. Ensure Databricks Premium or Enterprise Plan**  **Why:** Unity Catalog requires Premium or Enterprise tiers to enable governance features.  **How to check:**   * Sign in to **Databricks Workspace**. * In the left sidebar, click **Settings > Admin Console > Workspace Settings**. * Look under **Plan Information** to verify you are on **Premium** or **Enterprise**. * If on Standard, contact Databricks Sales to upgrade.   **2. Assign the Metastore Admin Role**  **Why:** Only a **Metastore Admin** can create and manage the Unity Catalog metastore.  **How to assign:**   * Open **Databricks Account Console** at accounts.cloud.databricks.com. * Navigate to **User Management > Users**. * Find your user account or group. * Under **Roles**, check the box for **Metastore Admin**. * Save changes.   **3. Confirm Workspaces Attached to Your Account**  **Why:** Your workspaces must be linked to the account that owns the metastore.  **How to check:**   * Go to **Account Console > Workspaces**. * Verify that all workspaces you plan to govern are listed there. * Note the workspace IDs—you will need them to attach the Unity Catalog metastore later.   **Tip:** If you have separate accounts for workspaces, you must consolidate them under a single account for Unity Catalog.  **4. Configure Databricks CLI**  **Why:** The CLI is required to automate setup steps and manage permissions.  **How to install and configure:**  **a. Install the CLI:**   * On your machine, run:   bash  CopyEdit  pip install databricks-cli  **b. Generate a Personal Access Token:**   * In your **Databricks Workspace**, click your user icon > **User Settings > Access Tokens**. * Click **Generate New Token**. * Copy the token immediately (it will not be shown again).   **c. Configure the CLI:**   * Run:   bash  CopyEdit  databricks configure --token   * When prompted:   + **Databricks Host:** e.g., https://<your-instance>.databricks.com   + **Token:** Paste the token you generated.   **d. Test the configuration:**   * Run:   bash  CopyEdit  databricks workspace ls   * If successful, you will see a list of objects in your workspace root.   **Prerequisites Checklist**  Before proceeding, confirm all of these are complete:   * Databricks Premium or Enterprise plan confirmed * You have Metastore Admin role * Workspaces attached to your account * Databricks CLI installed and configured |

* 1. Create Meta Store Catalog

|  |
| --- |
|  |
| **2.2 Create the Unity Catalog Metastore – Detailed Step**  **Open Databricks Admin Console**  Go to **Account Console** > **Data**  **1. Open the Account Console**   * Go to accounts.cloud.databricks.com. * In the left menu, click **Data** to open the **Metastores** page.   **2. Click *Create Metastore***   * At the top right, click the **Create Metastore** button to start the setup wizard.   **3. Provide Metastore Details** Fill in the required fields:   * **Name** A descriptive name for your metastore (e.g., prod-metastore). * **Storage Root Path** The location for managed tables:   + **AWS:** An S3 bucket URI, such as s3://your-bucket/unity-catalog.   + **Azure:** An ABFSS path, such as abfss://container@storageaccount.dfs.core.windows.net/unity-catalog. * **Region** Select the region where your data will be stored (must match your cloud storage).   **4. Configure Managed Storage Credential**   * Choose one of the following options to allow Databricks to access the storage:   + **Use Databricks-Generated IAM Role:** Let Databricks automatically create and manage the required permissions.   + **Create Your Own Role or Access Policy:** Manually configure IAM policies or Azure RBAC to grant Databricks access to your storage account. * Follow the on-screen instructions to validate permissions.   **5. Finish and Save**   * Review all settings. * Click **Create** to finalize your metastore.   **Tip:** After creation, you can return to the **Metastores** page to assign workspaces and manage settings. |

* 1. Assign Workspace to Metastore

|  |
| --- |
|  |
| **2.3 Assign Workspaces to the Metastore – Detailed Steps**  **1. Open the Metastore Settings**   * In the **Account Console**, go to **Data > Metastores**. * Find the metastore you just created. * Click the **Settings** icon (⚙️) or the metastore name to open its details.   **2. Click *Assign to Workspace***   * In the top-right corner, click the **Assign to Workspace** button to start the assignment wizard.   **3. Select Workspaces**   * In the list, check the box for each workspace you want to attach. * You can assign multiple workspaces at once.   **4. Confirm the Assignment**   * Review your selections. * Click **Assign** to link the workspaces to the metastore.   **Tip:** Once assigned, the workspaces can create catalogs, schemas, and tables governed by this metastore. |

**2.6 Enable Data Lineage – Detailed Steps**

|  |
| --- |
|  |
| **2.6 Enable Data Lineage – Detailed Steps**  **1. Understand Automatic Lineage Capture**   * Unity Catalog **automatically tracks lineage** for all queries, reads, and writes across your tables and views. * No additional configuration is needed to start collecting lineage metadata.   **2. Access Lineage in the UI**   * In your Databricks Workspace, open the **Data** icon in the left sidebar. * Navigate to **Data Explorer**. * Browse to the catalog, schema, and table you want to inspect.   **3. View Lineage Details**   * Select the table or view. * Click the **Lineage** tab at the top. * Explore the visual graph showing:   + Upstream and downstream dependencies.   + Notebooks and jobs that read or write the data.   + Operations and transformations applied. |

Simple Project: Databricks

|  |
| --- |
|  |
| **Simple Projects – Best Practices**  *(e.g., single catalog, small team)*  **1. Use One Catalog to Group All Data**   * Create a single catalog to organize all tables and views for simplicity. * Example: main\_catalog.   **2. Start with Table-Level Permissions**   * Grant access at the **table level** rather than columns or rows. * Keep permissions easy to manage as your team grows.   **3. Keep Schema Naming Consistent**   * Use clear, predictable schema names (e.g., finance, sales, marketing). * Follow a naming convention everyone understands.   **4. Avoid Over-Segmentation**   * Resist creating too many catalogs or schemas. * Keep the structure simple to reduce confusion.   **5. Document Grants in a Shared Spreadsheet**   * Maintain a clear record of all access grants. * Share the spreadsheet with your team to improve visibility and accountability.   **Tip:** Start small and expand gradually as your project or team grows in complexity. |

**Medium Project: Databricks**

|  |
| --- |
|  |
| **Medium Projects – Best Practices**  *(e.g., multiple teams, multiple schemas)*  **1. Create Separate Catalogs per Domain or Business Unit**   * Organize data by department or data domain. * Example catalogs: finance\_catalog, sales\_catalog, product\_catalog.   **2. Use Schemas to Isolate Environments**   * Create dedicated schemas for each stage of development:   + dev   + test   + prod * Keep data and permissions clearly separated.   **3. Implement Column Masking for Sensitive Fields**   * Protect sensitive columns like SSNs, emails, or salary data. * Apply column masks to ensure compliance and privacy.   **4. Leverage Groups Rather Than Individual Users for Grants**   * Assign permissions to **groups** instead of individual users. * Simplify access management as team members join or leave.   **5. Enforce Naming Conventions**   * Use consistent names for clarity:   + Format: <team>\_<purpose>\_<environment>   + Example: analytics\_orders\_prod   **6. Automate Grants with Terraform or Databricks CLI**   * Script permissions and object creation to avoid manual errors. * Use Infrastructure-as-Code to track changes over time.   **Tip:** Regularly review access and naming to ensure your environment stays organized as it scales.  Ask ChatGPT |

Complex Project: Databricks

|  |
| --- |
|  |
| **Complex Projects – Best Practices**  *(e.g., enterprise data mesh, multi-region)*  **1. Adopt a Catalog per Domain Model (Data Mesh Architecture)**   * Create separate catalogs for each data domain or business capability. * Example: customer\_domain, sales\_domain, supply\_chain\_domain.   **2. Implement Row-Level Security Policies for Multi-Tenant Data**   * Use row filters to restrict access to tenant-specific records. * Ensure compliance with privacy and data residency requirements.   **3. Enable Data Lineage and Integrate with External Catalogs**   * Turn on Unity Catalog lineage tracking for all assets. * Connect to external catalogs like Azure Purview or AWS Glue for enterprise-wide governance.   **4. Standardize All Permissions in Infrastructure-as-Code**   * Define grants and access policies in Terraform or automation scripts. * Version-control your configurations to track changes over time.   **5. Use CI/CD Pipelines for Schema and Table Creation**   * Automate deployment of schemas, tables, and policies through CI/CD workflows. * Reduce manual intervention and improve consistency.   **6. Periodically Review and Prune Obsolete Grants**   * Regularly audit who has access to what. * Remove outdated permissions to maintain least-privilege security.   **7. Consider Schema Evolution Policies for Big Data Ingestion**   * Define clear rules for handling schema changes, such as adding or renaming columns. * Prevent accidental data loss or pipeline failures during updates.   **Tip:** Align all practices with your enterprise data strategy to keep governance consistent across regions and domains.  Ask ChatGPT  Top of Form  Bottom of Form |

**Simple, Medium, Complex Categorization in databricks**

|  |
| --- |
|  |
| **Small Projects (Simple)**  **Common in:**   * Startups * PoC initiatives * Single-team analytics * Departmental reporting   **Thumb Rules:**   * **Team Size:** 1–5 users * **Workspaces:** 1 workspace * **Data Volume:** < 5 TB * **Data Domains:** Single domain * **Security:** Table-level permissions only * **Environment Separation:** Usually in folders (not separate schemas/catalogs) * **Automation:** Minimal (mostly manual setup) * **Governance Tools:** Mostly built-in Unity Catalog, no external catalog integration * **Example:** A marketing analytics team building dashboards on ad performance data   **Medium Projects**  **Common in:**   * Mid-sized enterprises * Department-wide analytics * Data product teams   **Thumb Rules:**   * **Team Size:** 5–30 users * **Workspaces:** 1–3 workspaces (e.g., dev/test/prod) * **Data Volume:** 5–100 TB * **Data Domains:** Multiple domains or business units * **Security:** Table- and column-level permissions * **Environment Separation:** Separate schemas for dev/test/prod * **Automation:** Partial (Terraform or Databricks CLI for provisioning) * **Governance Tools:** Unity Catalog + optional external lineage * **Example:** Finance and sales teams sharing curated data for forecasting   **Complex Projects (Enterprise)**  **Common in:**   * Large enterprises * Regulated industries (finance, healthcare) * Data mesh architectures * Multi-region deployments   **Thumb Rules:**   * **Team Size:** 30–500+ users * **Workspaces:** 5–50 workspaces across business units or regions * **Data Volume:** >100 TB–petabytes * **Data Domains:** Dozens of domains managed independently * **Security:**   + Row-level security for tenant isolation   + Column masking   + Detailed auditing * **Environment Separation:**   + Dedicated catalogs per domain   + Dev/test/prod isolation * **Automation:**   + Full Infrastructure-as-Code (Terraform, GitOps)   + CI/CD pipelines for deployment * **Governance Tools:**   + Unity Catalog   + External catalog integrations (Azure Purview, AWS Glue)   + Lineage reporting * **Example:**   + A global bank ingesting customer, transaction, and risk data across regions while complying with GDPR and SOC2   **How Architects in Large Brands Decide**  When defining whether a project is *small*, *medium*, or *complex*, most enterprises look at **6 factors**:   1. **Data Volume & Variety**    * More data = more governance complexity. 2. **Number of Users & Roles**    * More users = more fine-grained security and group management. 3. **Domains & Business Units**    * Multiple domains = likely need separate catalogs. 4. **Regulatory Requirements**    * Compliance drives complexity (e.g., PCI DSS, HIPAA). 5. **Environment Separation**    * Whether you need strong dev/test/prod isolation. 6. **Automation Maturity**    * Manual vs. fully automated deployments.   **Quick Decision Table**   |  |  |  |  | | --- | --- | --- | --- | | **Criteria** | **Small** | **Medium** | **Complex** | | Data Volume | <5 TB | 5–100 TB | >100 TB / Petabyte scale | | Team Size | <5 | 5–30 | 30–500+ | | Workspaces | 1 | 1–3 | 5–50 | | Domains | 1 | Few | Many | | Security | Table-level | Table + column-level | Row-level + column masking | | Automation | Manual | Partial Terraform | Full IaC + CI/CD pipelines | | Governance Tools | Unity Catalog | Unity Catalog + optional | Unity + External Catalogs + Lineage | |

**Databricks Project Complexity Assessment Template**

|  |
| --- |
|  |
| **Databricks Project Complexity Assessment Template**  Fill in the details for your project:   |  |  |  | | --- | --- | --- | | **Criteria** | **Your Project** | **Notes** | | **Data Volume** | *(e.g., 25 TB)* | Estimated total size of raw and processed data | | **Team Size** | *(e.g., 12 users)* | Number of data engineers, analysts, data scientists | | **Number of Workspaces** | *(e.g., 3)* | How many workspaces you plan to use (dev/test/prod or business unit separation) | | **Data Domains** | *(e.g., Sales, Finance)* | Distinct business areas whose data will be governed | | **Security Requirements** | *(e.g., Table + column masking)* | Level of access controls needed (table, column, row, masking) | | **Environment Separation** | *(e.g., separate schemas per environment)* | Whether you require isolation for dev/test/prod | | **Automation Maturity** | *(e.g., Terraform for setup)* | Whether you plan to automate infrastructure and permissions | | **Governance Tools** | *(e.g., Unity Catalog only)* | Whether you will integrate with external catalogs or only use Unity Catalog | | **Compliance Requirements** | *(e.g., GDPR)* | Any regulations that influence security and governance |   **Example: Completed Assessment**   |  |  |  | | --- | --- | --- | | **Criteria** | **Your Project** | **Notes** | | **Data Volume** | ~50 TB | Growing ~10 TB per year | | **Team Size** | 20 users | Mix of analysts and data engineers | | **Number of Workspaces** | 3 | dev, test, prod | | **Data Domains** | Finance, Marketing | 2 domains | | **Security Requirements** | Table-level + column masking | Some PII columns require masking | | **Environment Separation** | Separate schemas per environment | Each environment has its own schema | | **Automation Maturity** | Partial Terraform automation | Plan to expand IaC coverage | | **Governance Tools** | Unity Catalog | No external catalogs yet | | **Compliance Requirements** | GDPR | Must keep EU customer data in region |   **How to Use This Table**  After filling it out:  **Small Project Indicators**   * <5 TB data * <5 users * 1 workspace * Simple table-level permissions * Minimal automation   **Medium Project Indicators**   * 5–100 TB data * 5–30 users * 1–3 workspaces * Column masking or moderate security * Partial automation (Terraform, CLI)   **Complex Project Indicators**   * 100 TB or multi-petabyte scale * 30+ users across teams * Multiple workspaces (multi-region or data mesh) * Row-level security and column masking * CI/CD and full IaC * Integration with external catalogs and regulatory compliance |

Best Practise Databricks

|  |
| --- |
|  |
| **Simple Projects (Foundational Best Practices)**  **Data Organization**   * Create **one catalog** for all data assets. * Keep schema count minimal (e.g., default and analytics). * Use **clear table naming** (sales\_data, customer\_metrics).   **Permissions**   * Use **table-level permissions only**. * Assign grants directly to small groups (e.g., analysts). * Avoid overly complex role hierarchies.   **Governance**   * Document all grants and access in a spreadsheet or Confluence page. * Enable **basic audit logs** in your workspace.   **Operational**   * Manage schemas and tables through the Databricks UI. * Keep workspace configuration simple. * Do manual data quality checks.   **When to Re-evaluate:**   * When your data volume exceeds ~5 TB. * When more than 5–6 people need to collaborate. * When you start handling sensitive data (e.g., PII).   **Medium Projects (Scaling and Formalizing)**  **Data Organization**   * Create **separate catalogs per domain or business unit** (e.g., finance\_catalog, marketing\_catalog). * Use **schemas to isolate environments** (dev, test, prod). * Define **naming conventions** (<domain>\_<purpose>\_<env>).   **Permissions**   * Move from direct user grants to **group-based access**. * Introduce **column masking** for sensitive data. * Set up approval workflows for granting access.   **Governance**   * Enable **data lineage** in Unity Catalog. * Start integrating **basic Terraform** to manage catalogs, schemas, and permissions. * Maintain a **permissions registry** (e.g., in Git).   **Operational**   * Automate cluster configurations and permissions via Terraform or Databricks CLI. * Create **environment-specific workspaces or schemas** to reduce risk. * Establish **regular reviews of permissions and data usage**.   **Data Quality**   * Implement **expectations and validation** in ETL pipelines (e.g., using Delta expectations).   **When to Re-evaluate:**   * When your team grows beyond ~30 users. * When data volume approaches 100 TB. * When multiple regions or regulatory requirements appear.   **Complex Projects (Enterprise Maturity)**  **Data Organization**   * Adopt a **catalog-per-domain model** (data mesh architecture). * Use **dedicated workspaces per domain and environment**. * Maintain **strict naming and versioning policies** for all schemas and tables.   **Permissions**   * Implement **row-level security** for tenant isolation. * Combine **row filters and column masks** to protect sensitive data. * Use **SCIM** integration for automated group provisioning. * Enforce **least-privilege access** and periodic re-certification of permissions.   **Governance**   * Integrate Unity Catalog with **external data catalogs** (e.g., Azure Purview, AWS Glue). * Enable **full data lineage tracking** across all pipelines. * Maintain **Infrastructure-as-Code (IaC) for everything** (Terraform modules, GitOps). * Use **CI/CD pipelines** to deploy schema and table changes automatically.   **Operational**   * Automate workspace creation and configuration. * Use **cluster policies** to standardize compute governance. * Monitor **resource utilization and query performance**. * Build **data quality frameworks** into every ingestion pipeline.   **Compliance & Audit**   * Enable **audit logs** in centralized storage (e.g., S3, ADLS). * Set up **alerting** for unusual data access patterns. * Validate that data residency and encryption requirements are met.   **Data Quality**   * Define and enforce **schema evolution policies**. * Track **data freshness, completeness, and accuracy**. * Maintain **SLA dashboards** to monitor pipeline health.   **Pro Tip for Architects**  **If your project is in the Medium-to-Complex range, you should plan for:**  ✔ A **Data Platform Team** to own governance ✔ A **Product Owner per domain** to define policies ✔ A **DevOps pipeline** to version all changes ✔ A **shared metadata and compliance strategy** across regions |

Tool Selection – Databricks

|  |
| --- |
|  |
| **Tooling Stack by Project Complexity**  I’m grouping this into:   1. **Provisioning & IaC** 2. **Automation & CI/CD** 3. **Data Catalog & Lineage** 4. **Data Quality** 5. **Monitoring & Observability**   **Simple Projects**  *(Small teams, low data volume)*  **Provisioning & IaC**   * **Databricks UI** Create clusters, schemas, and tables manually. * **Optionally:** Use **Databricks CLI** for occasional automation.   **Automation & CI/CD**   * Manual deployment or lightweight scripts. * Git repository for notebooks (no CI/CD pipeline needed yet).   **Data Catalog & Lineage**   * **Unity Catalog (basic)** for metadata and permissions. * No external catalog needed.   **Data Quality**   * Manual checks or simple PySpark validation notebooks.   **Monitoring**   * Databricks built-in job and cluster metrics.   **🟡 Medium Projects**  *(Multiple teams, multiple schemas, growing compliance needs)*  **Provisioning & IaC**   * **Terraform** Use the Databricks Terraform Provider to manage:   + Unity Catalog resources (catalogs, schemas)   + Clusters and cluster policies   + Permissions * **Databricks CLI** for quick configuration.   **Automation & CI/CD**   * **GitHub Actions** or **Azure DevOps Pipelines**   + Automate deployment of Terraform configurations.   + Sync notebooks to workspaces. * **Databricks Repos** for versioning notebooks.   **Data Catalog & Lineage**   * **Unity Catalog**   + Table and column-level lineage.   + Automated metadata management. * Optionally integrate with:   + **Collibra** (if you have enterprise metadata initiatives).   **Data Quality**   * **Delta Expectations** in pipelines. * **Great Expectations** or **Databricks Data Quality (DQ)**   + Build validation suites.   **Monitoring**   * **Databricks Jobs API** * **Cloud-native logging**   + AWS CloudWatch   + Azure Monitor * Alerts on failed jobs.   **Complex Projects**  *(Enterprise scale, data mesh, multi-region)*  **Provisioning & IaC**   * **Terraform (full Infrastructure-as-Code)**   + Unity Catalog objects   + Workspace configurations   + SCIM provisioning   + Cluster policies * **Databricks Terraform Modules**   + [databricks-terraform/common-workspace](https://github.com/databrickslabs/terraform-databricks-examples)   **Automation & CI/CD**   * **Azure DevOps Pipelines** or **GitLab CI**   + Deploy Terraform plans.   + Promote notebooks and pipelines across environments.   + Automate schema/table deployment. * **Databricks Repos**   + Managed notebook code synced with Git.   **Data Catalog & Lineage**   * **Unity Catalog + External Data Catalogs**   + Azure Purview   + AWS Glue   + Collibra * **Data Lineage**   + Unity Catalog lineage graphs.   + Optional integration into enterprise lineage tools.   **Data Quality**   * **Databricks Data Quality (DQ) with Expectations** * **Great Expectations**   + Validation integrated into ETL jobs. * **Deequ** (for Scala/Spark-based validations).   **Monitoring & Observability**   * **Databricks audit logs**   + Stream to storage for compliance. * **Cloud-native monitoring**   + Azure Monitor Log Analytics Workspace   + AWS CloudWatch Logs and Metrics * **Custom dashboards**   + Grafana (for cluster and job metrics). * **Alerts**   + Data quality failures   + SLA breaches   + Permission changes   **Recommendation Matrix**   |  |  |  |  | | --- | --- | --- | --- | | **Capability** | **Simple** | **Medium** | **Complex** | | **IaC** | Databricks UI/CLI | Terraform + CLI | Full Terraform (modular) | | **CI/CD** | Git only | GitHub Actions or Azure DevOps | Azure DevOps or GitLab CI/CD | | **Catalog & Lineage** | Unity Catalog (basic) | Unity Catalog | Unity Catalog + Purview/Glue/Collibra | | **Data Quality** | Manual checks | Delta Expectations + Great Expectations | Great Expectations + Databricks DQ + Deequ | | **Monitoring** | UI monitoring | Cloud-native metrics + alerts | Cloud-native observability + audit log pipelines |   **How to Select the Stack for Your Project**  If you want to **shortlist**:   * **Simple:** Use **Databricks UI + Unity Catalog** out-of-the-box. * **Medium:** Add **Terraform and GitHub Actions**, start **Delta Expectations**, enable **lineage**. * **Complex:** Go all-in with:   + Terraform IaC   + CI/CD pipelines   + External catalog integration   + Data quality frameworks   + Cloud-native monitoring pipelines |