AI Factory Unity Catalog & Data Governance

[**Purpose 3**](#_n94bglyder3j)

[**Requirements 3**](#_wl9l2soz5a8r)

[Functional Requirements 3](#_dqcplgpl0cjg)

[FR-01 3](#_hb4lvdx75ge5)

[FR-02 3](#_c2cu1pva4kvi)

[FR-03 4](#_h9f29sj4c6sj)

[FR-04 4](#_c2cu1pva4kvi)

[FR-06 4](#_9e20obxennkp)

[FR-07 4](#_g63oc48bafu)

[FR-08 4](#_6q1818g8nd0j)

[FR-09 4](#_up4ti13lhpn)

[FR-10 4](#_eu1rqf52lxi)

[FR-11 4](#_bu2c2bt8y0hv)

[FR-12 4](#_hwuczpb6xet4)

[Non-Functional Requirements 4](#_go5li7n6nze4)

[NFR-01 5](#_tky2kuidipb)

[NFR-02 5](#_swe36vl545yt)

[NFR-03 5](#_ufwgu2ttnrqq)

[NFR-05 5](#_dbidwykae35r)

[**Architectural Decisions 5**](#_bkxvc3ostgsq)

[AD-01 6](#_lqoqnsem9n6)

[AD-02 8](#_3ibdkhtch4fu)

[AD-03 9](#_5pg7lcgdnjnr)

[AD-04 11](#_xptl0gu36igd)

[AD-05 12](#_w4dyakkhjmw5)

[AD-06 14](#_uy6esljhikwc)

[AD-07 16](#_rv5ol5xetm9c)

[AD-08 18](#_1pbaicn7b9gy)

[AD-09 19](#_uw2ncn64znf9)

[AD-10 21](#_ksymt7h2la7i)

[**Scope validation for Implementation of Assets 22**](#_adno8u1ywpal)

[**Naming Conventions 23**](#_qne0rcb1tlen)

[**Tagging 24**](#_3ht5w68ruha6)

[**UC design Layout 24**](#_lvaiazt5ubwp)

[**Guiding principals 25**](#_za7ibdy4cj2)

[**Business Units 26**](#_tl1lqe3xo8h)

[Workspaces 26](#_9bro98anvmx8)

[Sub Domains 27](#_wc0iz0e01ep0)

[Apps 27](#_x1en36b8hge6)

[Catalogs 27](#_av5rh8wpirtm)

[Schemas 28](#_40coi4hc5g93)

[Catalog-Schema setup for each sub-domain 29](#_wucmyy1glhqs)

[Storage Layout 31](#_xwjveg16x1lq)

[**Data Access 32**](#_x0aok3z5rqya)

[Option-1: Direct Access 33](#_ukee9fpmvbtr)

[Option-2: Data Replication 33](#_7lw9vsabuxjk)

[**Data Sharing 34**](#_alal18ntg20q)

[Overview 35](#_yc3v1uic8qyz)

[Option-1: Multiple published catalogs from each BU 35](#_sgwgmel9c1r0)

[Option 2: Single published catalog from all BUs 37](#_92kh1vpiiini)

[**Automation and CICD 38**](#_cuiqwbd3c3vi)

[Overview 39](#_yk8hfvaikv4x)

[High Level Flow 39](#_iplv4mzbarn6)

[Deployment of Unity Catalog assets 40](#_wmth9phj25vn)

[Deployment of Code assets 41](#_loiv0yn67usf)

[Databricks Asset Bundles 42](#_ubvthmvvidwo)

[High-level view of a development and CI/CD pipeline with bundles: 43](#_1dbjp1t8i93y)

[Components and Configuration 43](#_1l3cy72jb5k1)

[CICD with DABs 44](#_gikwsenlqz4t)

[Bundle commands across environments - 47](#_dp1734ftny7p)

[DABs Project Structure 48](#_7nwmjwckh2ps)

[Configure stages in Databricks Asset Bundles 49](#_6tnhg5b33ep9)

[Overwrite job specific definitions in DABs 50](#_mp5xhkswcv0u)

[Deployment Pattern in DABs 50](#_qjmja9kv6ugq)

[Table Deployment 51](#_7vfekz3gqzss)

[Option-1: Using Databricks Asset Bundles 51](#_wripevijf30k)

[Option-2: Using Terraform (databricks\_sql\_table resource) 52](#_4pm281nvoa8u)

[Persona Permissions 53](#_cdi3bf7wlvee)

[Permissions Consideration related to SC environment 54](#_u2n0n8sfhff0)

[**Security and Compliance 56**](#_vfgjw7eg056x)

[Production Data Masking 56](#_ickrvrs7a210)

[System Table 56](#_mye66wkbvjs7)

[Information Schema 57](#_e5b98ml26216)

[**Reference Links 58**](#_7307k9jzgdza)

[**Limits 58**](#_9gn7dwhe8eug)

[Trimmed part - 60](#_4zbw579m2aud)

[Stages and Lifecycle 60](#_9t8h9uw05jgf)

[EDA & Experimentation (Sandbox) 60](#_s6ggrkemvzx8)

[Dev & Staging 60](#_nla033lxmx7k)

[Production 61](#_nk25ofa1ilrd)

[Within Metastore 62](#_odlcy3bgt2b9)

# 

# Purpose

This document aims to develop a Unity Catalog and Data Governance framework for the AI Factory via reusable Terraform templates and CI/CD pipelines. These tools will simplify deploying Unity Catalog across various Business Units.

# Requirements

These requirements detail the core functionalities necessary for the solution, aiming to establish the fundamental capabilities required to build the foundation of the Databricks Unity Catalog for Standard Chartered Bank.

## Functional Requirements

These requirements focus on the functionality required from the solution. The intention here is to capture the information around “What” are the core capabilities needed to create the base of the Databricks Platform.

| **ID** | ***Type*** | **Description** |
| --- | --- | --- |
| FR-01 | *Platform* | The system shall provide the capability for separation of the sandbox, dev, staging and prod environments, in terms of storage and access. |
| FR-02 | *Data Sharing* | Capability to share data within and across metastore for different BUs. |
| FR-03 | *Platform* | The system shall provide the capability to restrict data and compute access |
| FR-04 | *Data Access* | Access to Production data in masked form from the Sandbox environment. |
| FR-06 | *Security* | The system shall provide the capability to integrate users with Microsoft Entra ID (Azure AD) |
| FR-07 | *Security* | Ensure that PII data is stored in the data lakehouse in a secure manner and is only accessible to users who are authorized to use PII data |
| FR-08 | *Security* | The system shall provide the capability to support fine-grained access control on data assets. |
| FR-09 | *Data Storage* | The data lake shall be built on Azure Data Lake Storage (ADLS) Gen2. The format of data storage must not be proprietary, and should be accessible given the right security privileges. |
| FR-10 | *Data Storage* | The System shall provide the capability to segregate data storage by Business Units, Teams and Apps as dictated by CDO Guidelines and business requirements. |
| FR-11 | *Data Storage* | The system shall provide the capability to create a logical data layout which can be segregated based on data maturity i.e. Bronze, Silver, Gold which aligns with the widely-used medallion architecture |
| FR-12 | *DevOps* | Ability to deploy various platform artifacts from one environment to another environment in the most automated way possible. |

## 

## Non-Functional Requirements

These sets of requirements focus on the non-functional aspects such as availability, scalability, reliability etc that impact the performance of the overall solution. Since non-functional requirements can significantly influence cost and are heavily influenced by business needs, it is important to take these into consideration when an end business use case is on the table. As of the date of writing this document, these weren’t captured for a specific business use case but kept it at platform level NFRs.

| **ID** | **Type** | **Description** |
| --- | --- | --- |
| NFR-01 | Open Source | The system should support Open Source Software (OSS), Open Standards, Open Protocols and Open Formats without having a vendor lock-in. |
| NFR-02 | Scalability | The system should be scalable to meet business requirements at the peak workload and should be able to scale out and scale in automatically with minimum human intervention. |
| NFR-03 | Performance | The system should be able to provide the expected performance and should be adaptable for changes in load without impacting the performance. |
| NFR-05 | Compatibility | The system should be interoperable so that it can co-exist with existing systems with the ability to exchange data to and from it. |

# Architectural Decisions

This section showcases key technical decisions that impact the overall solution architecture. Each decision influences the overall design of the solution and choice of components moulded into the architecture. The intent of this section is to showcase the thought process that went into making critical decisions around architecture.

# 

| **ID** | **Type** | **Questions** | **Status** |
| --- | --- | --- | --- |
| [AD-01](#_lqoqnsem9n6) | Azure Databricks Platform | How many workspaces need to be created per Business Unit? | Design completed |
| [AD-02](http://ad-02) | Azure Databricks Platform | How should we structure the Unity Catalog Layout? | Design completed |
| [AD-03](http://ad-03) | Azure Storage | How should we structure the storage accounts for the Databricks Platform? | Design completed |
| [AD-04](http://ad-04) | Azure Databricks Platform | How should we structure the groups for data and compute segregation on the databricks platform? | Design in review |
| [AD-05](#_w4dyakkhjmw5) | UC asset CICD | How shall the Unity catalog assets be deployed via CI/CD pipelines? | Design completed |
| [AD-06](#_uy6esljhikwc) | Data asset CICD | How shall Table Definitions be deployed as part of the CI/CD process? | Design in review |
| [AD-07](#_rv5ol5xetm9c) | Data Access | How should developers access prod data in the sandbox workspace? | Design in review |
| [AD-08](#_1pbaicn7b9gy) | Data Sharing | How should data be published for sharing across BUs? | Design in review |
| [AD-09](#_uw2ncn64znf9) | Data Access | Who will own the process for publishing data from production to sandbox/dev env? |  |
| [AD-10](#_ksymt7h2la7i) | Data Sharing | Who will own the process for creating the delta shares and managing the access? |  |

| AD-01 | How many workspaces need to be created per Business Unit? |
| --- | --- |
| Category | Azure Databricks Platform |
| Description | The [workspace layout](#_9bro98anvmx8) for a Business Units, needs to consider not only the separation of billing that each business unit might accrue but should also balance the storage separation (i.e. data owned/hosted by each BU) with the data requirements for solving a use case which might require data from multiple BUs. |
| FRs/NFRs Addressed | FR-01 |
| Options | **Option - 1**:  Create *4 workspaces per Business Unit* separated into the various environments as follows:   * Sandbox - Sandbox workspace for a single BU for Exploratory Data analysis, Experimentation etc with read only access to production masked data. * Dev - Dev workspace for BU for running CI tests with only metadata. * Staging - Pre Prod like workspace with synthesized data. * Prod - Prod workspace to run scheduled or triggered prod pipelines for the BU.   **Option - 2**:  *Create 3 workspaces per Business Unit*. In such a scenario, the 3 workspaces required per BU would be as follows:   * Sandbox : Sandbox for Exploratory Data analysis, Experimentation etc with read only access to production masked data. * Dev/Staging : Workspace for BU for running CI tests with synthesized metadata. * Prod: Prod workspace to run scheduled or triggered prod pipelines for the BU. |
| Justification | **Pros/Cons of Option-1:**  ➕Enables progressive promotion pipelines (Sandbox → Dev → Staging → Prod), with checks and validations at each step.  ➕Different roles (e.g., data scientists vs. ML engineers) can be better segmented per environment.  ➖Could run into workspace capacity problems since each workspace has a limited capacity which is governed by Number of Cluster Nodes that be created, which is governed by the IP CIDR range assigned to a VNET.  ➖Four workspaces per BU increase operational and compute cost, especially if BUs are many.  ➖New teams may take longer to understand the full flow and workspace strategy.  **Pros/Cons of Option-2:**  ➕Fewer workspaces mean reduced admin overhead for permissions, secrets, Unity Catalog bindings. Saves on workspace costs — compute, storage, access control management.  ➕Still enables CI/CD with a “dev/staging” hybrid model — suitable for agile or fast-moving teams.  ➖Harder to apply fine-grained controls (e.g., Data Engineers testing in dev vs. Business Users validating in staging). |
| Decision | Option 1 is preferred by SC |

| AD-02 | How should we structure the unity catalog layout? |
| --- | --- |
| Category | Azure Databricks Platform |
| Description | Unity Catalog in Databricks provides a 3 level namespace (catalog.schema.table) to organize the data assets and govern them. This section outlines the options related to design patterns in the unity catalog. A more detailed layout view as per discussion with the SC teams has been captured in the section [Unity Catalog Layout](#_lvaiazt5ubwp). |
| FRs/NFRs Addressed | FR-01, FR-10, FR-11 |
| Options | **Option - 1**:  Create *sandbox, dev, staging and prod catalogs at sub-domain level ( BU+ Teams)* and schemas per application. Tables related to an application go in a particular schema as described in the [**layout**](#_wucmyy1glhqs).    **Option - 2**:  Create *sandbox, dev, staging and prod catalogs for each BU* and schemas per sub-domain(BU + Team).  Note: The platform provides the technical flexibility for both options to co-exist if such a scenario is encountered arising out of business requirements or other technical limitations. |
| Justification | **Pros/Cons of Option-1:**  ➕Data governance per sub-domain is slightly simpler since users or groups can be separated at a Catalog level.  ➕Data governance per app use case is much simpler since users or groups can be restricted at a schema level.  ➖There is a limit of *1000 catalog*s per metastore and in a single account model, the metastore per region is shared across all LOBs.  **Pros/Cons of Option-2:**  ➕Less number of catalogs are required that means less operational overhead and *1000 catalogs* in a single account limitation *an issue*.  ➖Data governance per BU is slightly more complicated as now users of a particular BU have to restricted to multiple schemas belonging to a particular BU. |
| Decision | Option 1 is preferred. |

| AD-03 | How should we structure the storage accounts for the Databricks Platform? |
| --- | --- |
| Category | Azure Databricks Platform |
| Question | Storage accounts refer to the underlying cloud storage (i.e. Azure ADLS Gen 2) which will need to be created at a catalog, schema or table level as per separation and storage requirements. This has been discussed with the SC Teams and captured in the section [Storage Account Layout](#_xwjveg16x1lq). As a thumbrule, we suggest segregating data assets based on data maturity i.e. bronze, silver and gold, where bronze refers to raw data, silver refers to cleansed/ conformed data and gold refers to aggregated data. However it must be noted that this is only a logical denomination and may vary based on scenario, complexity and business requirements. For more details on the medallion architecture methodology please refer to the databricks documentation [here](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/medallion). |
| FRs/NFRs Addressed | FR-09 |
| Options | **Option - 1**:  Manage [storage layout](#_xwjveg16x1lq) by segregating the storage *Storage Accounts* **per env, and BU** and *Container* **per application schema**.  **Option - 2**:  Segregate *Storage Accounts* **per env and sub-domain** and *Container* per **application schema**.  Note: The platform provides the technical flexibility for both options to co-exist if such a scenario is encountered arising out of business requirements or other technical limitations. Also note that larger more valuable tables (i.e. clickstream events) which have a higher access rate, it might be necessary to segregate at a ADLS Gen2 Storage Account level to avoid throttling of requests. |
| Justification | **Pros/Cons of Option-1:**  ➕Separate containers per Team and Application schema, with a shared storage account per BU.  ➕Lesser number of private endpoints required per workspace to access the storage account leading to lesser operational overhead and network costs.  ➖There could be scenarios where data volume is not very high and option-1 would be preferable for operational simplicity and vice-versa due to technical limitations.  **Pros/Cons of Option-2:**  ➕Separate storage accounts per sub-domain with containers created by application schema  ➖More number of private endpoints required per workspace to access each storage account (per BU) leading to potential increase in operational overhead and network costs. |
| Decision | Option 1 is preferred by the SC platform team. |

| AD-04 | How should we structure the groups for data and compute separation on the databricks platform? |
| --- | --- |
| Category | Platform |
| Description | Data governance can be achieved on the databricks platform using the unity catalog governance model, by first creating ***groups*** and then ***granting access*** to data assets by those groups. Similarly Compute governance is achieved through creating cluster policies (based on use\_case patterns) and assigning the usage of allowed cluster policies to the respective groups. Compute policies are not required as part of terraform templates as per discussion with SC. |
| FRs/NFRs Addressed | FR-03 |
| Options | **Option-1:**  Create groups per env, bu/sub-domain, persona and data access  In such a scenario the patterns would be:  ***default:***<bu>-<env>-<pii/gen>-<persona>-<rw/ro>  ***special-scenarios:***<bu>-<env>-<use-case>-<pii/gen>-<persona>-<rw/ro>    **Option-2:**  Create separate groups for data and compute governance  For data governance, restrict groups by env, BU/Sub-domain and Data access  For compute governance restrict groups by env, BU/Sub-domain and Persona  In such a scenario, the data governance group patterns would be:  ***default:***SUZ1-Users-AIFactory-<bu/sub\_domain>-<env>-<data\_maturity>-<app\_name>-<ro/rw>  ***special-scenarios:***SUZ1-Users-AIFactory-<bu/sub\_domain>-<env>-<use\_case>\_<app\_name>-<pii>    Compute governance group patterns would be:  ***default:*** SUZ1-Users-AIFactory-<env>-<bu>-<persona>  ***special-scenarios:*** SUZ1-Users-AIFactory-<env>-<bu/usecase>-<persona>  ** |
| Justification | **Pros/Cons of Option-1:**  ➕Single group structure for both data and compute governance.  ➖Mixing data and compute governance in one group could lead to complications and might need more number of groups to be created.  ➖Operational overhead might increase to manage more number of groups.  **Pros/Cons of Option-2:**  ➕Separate groups for data and compute governance offer simplicity and lesser number of groups need to be created.  ➕Lesser operational overhead. |
| Decision | Option 2 is recommended by Databricks as it can help manage compute and storage permission separately. |

| AD-05 | How shall the Unity catalog assets be deployed via CI/CD pipelines? |
| --- | --- |
| Category | Azure Databricks Platform - CI/CD & Unity Catalog |
| Description | Unity Catalog assets such as catalogs, schemas, and grants need to be deployed across environments (e.g., sandbox, dev, staging, prod) in a consistent, secure, and auditable way. The deployment process should support Infrastructure as Code (IaC), promote traceability via version control, and align with GitOps ADO and SC DevOps best practices via governed templates. |
| FRs/NFRs Addressed | FR-12 |
| Options | **Option - 1**:  Deploy Unity Catalog assets using [Terraform and Databricks Provider](#_wmth9phj25vn)   * Define UC assets (databricks\_catalog, databricks\_schema, databricks\_grants, etc.). as YAML config * Store configurations in an ADO repo and manage environment-specific values via variables or workspaces. * CI/CD pipelines run terraform init, plan, and apply to deploy to target environments using SC governed templates.   **Option - 2**: Manual deployment using ClickOps   * Unity Catalog assets (catalogs, schemas, tables, views, etc.) are created and modified manually via the Databricks workspace UI. |
| Justification | **Pros/Cons of Option-1:**  ➕Fully automated, repeatable, and scalable across environments.  ➕Version-controlled — all changes are reviewed via Git workflows.  ➕Enables auditability, traceability, and compliance alignment.  ➖Requires initial setup of Terraform, repo structure, and governed templates..  **Pros/Cons of Option-2:**  ➕Quick and easy for one-time setups, demos, or sandbox experimentation.  ➖No version control — changes cannot be tracked or rolled back easily.  ➖ High manual effort, prone to human error and misconfiguration.  ➖ Not reproducible across environments. |
| Decision | Option 1 is preferred by SC and Databricks. |

| AD-06 | How shall Table Definitions be deployed as part of the CI/CD process? |
| --- | --- |
| Category | Azure Databricks Platform - CI/CD & Unity Catalog |
| Description | Table definitions in Databricks need to be version-controlled and deployed consistently across environments (Sandbox, Dev, Staging, Prod). The approach should support automation, minimize manual intervention, and enable traceability, auditability, and rollback via Infrastructure as Code. The options have also been detailed out in the `[Automation and CI/CD / Table Deployment](#_7vfekz3gqzss)` section below.   | **NOTE:** It’s **not mandatory** in a databricks environment to deploy tables using DDL Scripts. Tables can and are most commonly created/modified as part of **PySpark jobs** where if a table doesn’t exist while writing as part of a scheduled job/workflow, it gets created **automatically** as part of the write step. Also schema evolution in most cases is automatically handled when the spark property *`mergeSchema`* is set to *`True`* during the write step. The Pyspark Jobs in turn are managed and deployed using the CI/CD process. | | --- | |
| FRs/NFRs Addressed | FR-12 |
| Options | **Option - 1**:  Use [Databricks Asset Bundles (DAB)](#_gikwsenlqz4t)   * Deploy Tables using DDL Scripts as part of code deployment using Databricks Asset Bundles. * In this scenario, DDL scripts/Notebooks are deployed along with databricks workflows (using DAB commands) and after deploying to a particular environment (i.e. prod), a databricks workflow is executed using a service account which has access to create/modify tables.   *-- initial ddl*  ***create table if not exists <table\_name>***  ***(***  ***id string***  ***name string***  ***dept\_id string***  ***load\_date string***  ***)***  ***partitioned by (load\_date)***  ***;***    --modifications  ***alter table <table\_name>***  ***add column location string after dept\_id;***     * The Databricks Asset Bundles can further be wrapped and run as part of an Azure Devops Pipeline for automation based on a ddl\_flag, i.e. deployment/modification of tables is needed.   **Option - 2**:  Use Terraform with Databricks Provider:   * Use terraform [databricks\_sql\_table](https://registry.terraform.io/providers/databricks/databricks/latest/docs/resources/sql_table) resource to deploy tables. * In this scenario, developers would have to create table definitions as part of the terraform config, and the terraform template is deployed as part of the Azure Devops pipeline along with other unity catalog assets. * However in this scenario, a Databricks SQL Warehouse specifically to be used for table deployment would also have to be configured in the workspace. |
| Justification | **Pros/Cons of Option-1:**  ➕Part of the development/deployment workflow: Table definitions when required can be created/modified by developers and deployed to the higher environments as part of the code deployment via CICD  ➕Flexibility: In case of complex scenarios where datatypes or schemas change, the code can be modified by developers to ensure any schema correction if required.  ➕Schema Evolution: It is a natural consequence of the flexibility offered via deployment scripts or notebooks.  ➖Too flexible and reliant on DDL scripts being created correctly for complex scenarios.  ➖Requires adoption of the Databricks Asset Bundle structure, which might present a bit of a learning curve.  **Pros/Cons of Option-2:**  ➕Stronger control on the table deployment process as all tables will be deployed via Terraform  ➖**Schema evolution is not supported** out of the box and in case of complex scenarios, we have to rely on special scripts or manual intervention.  ➖Not part of the development workflow, Table definitions have to be added manually or programmatically to the terraform config. |
| Decision | Option-1 is recommended by Databricks as it provides more flexibility and a lack of schema evolution [ref](https://registry.terraform.io/providers/databricks/databricks/latest/docs/resources/sql_table) in Option-2. |

| AD-07 | How should developers access prod data in the sandbox workspace |
| --- | --- |
| Category | Data Access |
| Description | Data Scientists often need access to real data in sandbox environments to perform exploratory analysis, prototyping, or experiment design. However, direct access to production data raises data governance and compliance concerns. This decision focuses on how to enable controlled access to production-like data while preserving security, privacy, and isolation.  For more details on the below options refer to the section [Data Access](#_x0aok3z5rqya) section below. |
| FRs/NFRs Addressed | FR-04, FR-07 |
| Options | **Option - 1**:  Provide [read-only access](#_x0aok3z5rqya) to masked or anonymized production data   * Masked or anonymized views/tables are created in the production catalog. * Developers in the sandbox workspace are granted read-only access to those specific views/tables via Unity Catalog.   **Option - 2**:  Replicate a subset of production data (masked) into a separate sandbox catalog   * A scheduled job (e.g., daily/weekly) replicates a masked subset of production data into a dedicated sandbox catalog/schema. * No direct access to the live production data or tables. |
| Justification | **Pros/Cons of Option-1:**  ➕No data duplication — Data Scientists always query the latest production state.  ➕Centralized governance: Access is auditable via Unity Catalog logs.  ➕Lightweight and simple to maintain (no data sync jobs).  ➖Accessing production data directly (even if masked) may raise concerns for sensitive datasets.  ➖May affect performance of production workloads during heavy sandbox usage.  **Pros/Cons of Option-2:**  ➕Clear isolation between production and sandbox — sandbox queries never touch prod tables.  ➕Enables transformation or sampling during replication (e.g., synthetic generation, downsampling).  ➖Requires a sync pipeline to keep data fresh and masked properly.  ➖Potential lag between prod and sandbox data availability.  ➖Slightly higher storage and compute costs due to data duplication. |
| Decision | Option 1 is recommended by Databricks as it avoids data replication. |

| AD-08 | How should data be published for sharing across BUs? |
| --- | --- |
| Category | Data Sharing |
| Description | In a multi-BU setup, some data assets (e.g., schema, tables) need to be made available across business units for reuse and collaboration. Consideration should be to design a scalable, governed approach for cross-BU data sharing that ensures consistency, minimizes duplication, and respects access control boundaries. These details are mentioned in the [Data Sharing section.](#_alal18ntg20q) |
| FRs/NFRs Addressed | FR-02 |
| Options | **Option - 1** :  Use Delta Sharing and publish to catalog at the BU level:   * Each BU exposes its datasets (e.g., gold or feature tables) through a Delta Sharing `[share](https://learn.microsoft.com/en-us/azure/databricks/delta-sharing/#share)`. * A single Catalog will be created per BU (e.g., cib\_published, wrb\_published) * Other BUs are added as recipients to specific shares based on need-to-know basis. * Sharing is done at table or schema level, not entire catalog   **Option - 2**:  All BUs publish into a single catalog in the central BU.   * All BUs share data assets (tables, schemas) to a single delta sharing `share` * All data assets are published into a single `centralized\_published` catalog. |
| Justification | **Pros/Cons of Option-1:**  ➕Fine-grained access control — specific datasets can be exposed only to consuming teams/workspace that need them.  ➕Encourages data product ownership by teams, aligning with Data Mesh principles. Easier to set up lineage, cost attribution, and audit at the team level.  ➖Increases the number of shares and catalogs to manage.  ➖Teams need clarity on ownership and support expectations for shared data products.  **Pros/Cons of Option-2:**  ➕Easier to manage — fewer shares and catalogs to govern.  ➕Reduces overhead in catalog structure and permission management.  ➖Coarse-grained control — hard to expose different datasets to different teams without creating too many ACL exceptions.  ➖Risk of overexposing data within the org, especially when sensitive datasets span multiple BUs. |
| Decision | Option 1 and 2 are shared with SC, decision pending |

| AD-09 | Who will own the process for publishing data from production to sandbox/dev env? |
| --- | --- |
| Category | Data Sharing |
| Description | In a multi-environment setup, certain datasets (e.g., masked production data) are required in sandbox or development environments for experimentation and model development. A decision is needed on who should own and govern the data publishing workflow between environments — individual BUs or a central platform team. |
| FRs/NFRs Addressed | FR-04 |
| Options | **Option - 1** :  Ownership by BU/Sub-domain Admins   * Each BU or sub-domain owns the process of publishing data from production to lower environments (sandbox/dev). * Responsibility includes masking PII, auditing access, and ensuring environment segregation.   **Option - 2**:  Ownership by Central Platform Team   * A centralized team owns and operates a shared service or framework to publish data from production to sandbox/dev. * Standardized policies and automation applied for masking, logging, and audit. |
| Justification | **Pros/Cons of Option-1:**  ➕ Promotes domain-level ownership and agility.  ➕ BU teams can prioritize their own data publishing workflows.  ➕ Aligns with Data Mesh principles where teams are accountable for their own data lifecycle.  ➖ Requires each BU to develop and maintain their own secure publishing pipelines.  ➖ May lead to inconsistent data masking and governance standards across BUs.  **Pros/Cons of Option-2:**  ➕ Ensures uniform governance, masking, and audit practices.  ➕ Easier to scale and maintain a single governed framework.  ➕ Reduces duplication of effort across BUs.  ➖ Central bottleneck if not properly resourced.  ➖ May reduce flexibility for BU teams needing faster or custom data publishing workflows. |
| Decision | Decision pending |

| AD-10 | Who will own the process for creating the delta shares and managing the access? |
| --- | --- |
| Category | Data Sharing |
| Description | Delta Sharing allows structured sharing of data assets (e.g., tables, schemas) across Databricks workspaces or with external consumers. A clear decision is needed on who will own the process of creating shares and managing access controls — whether it should reside with each BU or be governed centrally to ensure standardization |
| FRs/NFRs Addressed | FR-02 |
| Options | **Option - 1** :  Ownership by BU:   * Each Business Unit (BU) is responsible for creating their own Delta shares and managing access for consumers (internal or external). * BU teams define what data is shared, to whom, and under what conditions.   **Option - 2**:  Ownership by Central Team   * A central platform or governance team owns all Delta share creation and access management. * BUs request share creation and access changes via defined workflows using change config details. |
| Justification | **Pros/Cons of Option-1:**  ➕ Encourages data ownership and autonomy at the domain level.  ➕ Faster execution and flexibility tailored to BU-specific needs.  ➕ Aligns with Data Mesh principles.  ➖ Inconsistent security or governance standards across BUs.  ➖ Risk of overexposing sensitive data if access is not centrally validated.  ➖ Requires strong controls and training in every BU.  **Pros/Cons of Option-2:**  ➕ Centralized enforcement of data governance, masking, and access standards.  ➕ Easier to audit and manage cross-BU data flows at scale.  ➕ Reduces misconfiguration and duplication across teams.  ➖ Potential bottlenecks if central team is under-resourced.  ➖ May limit agility and experimentation in BUs. |
| Decision | Decision pending |

# Scope validation for Implementation of Assets

| S. No |  | Comments |
| --- | --- | --- |
| 1. | UC assets will be deployed via terraform using Azure ADO with Governed templates to one of the scdbw-scaifacto-we-dev/scdbw-sc-aifact-sea-dev workspaces.  UC assets to be included:[scb\_terraform\_uc\_deployment](https://docs.google.com/spreadsheets/d/1Zrqq-B4EqJBwmFGSG02YvISFtCeJ8xYQh9KmyTO3xbU/edit?gid=0#gid=0) |  |
| 2. | Additional Azure storage containers and managed identities required for the BU catalog/schema setup will be created via Azure portal using click ops under the existing storage account. | Storage accounts and containers will be provided and they will be mapped to the Unity catalog. Mapping is included. |
| 3. | Existing metastore deployed in the region will be used. |  |
| 4. | Asset Bundles design and template scope for 30th June. | General template in case tables are not deployed via terraform and asset bundles are used. Asset bundles will not be included if tables are deployed via Terraform. |
| 5. | Governed template for Terraform CICD | Use existing governed templates. |
| 6. | Governed template for DAB | Will the same Terraform governed be used? Need to move the centralized template to the central repo? **[In evaluation]** |
| 7. | Dynamic masking implementation not in scope |  |

# 

# Naming Conventions

| ***Type*** | ***Pattern*** | ***Example*** |
| --- | --- | --- |
| Catalog | <sub-domain>\_<env> | cib\_cash\_sandbox,  cib\_cash\_dev |
| Schema | <medallion\_layer>\_<app\_name> | <username>\_schema, bronze\_<app1>,  silver\_<app1>,  gold\_<app1> |
| Table | <subject\_area>\_<name> | lsr\_monitoring\_agg |
| Storage Account / Container | <env><bu><data-maturity>/<usecase> | procib/cibcashsandbox, procib/cibcashdev |
| Workspace | <env>\_<bu>\_<id> | dev\_cib\_01 |
| Azure AD /Databricks Account Group | <bu>\_<env>\_<app>\_<data-maturity> | cib\_dev\_app1,  connectivity\_prod\_home\_customer360\_gold\_ro |
| Non Azure AD Databricks Account Group | <bu>\_<env>\_<app>\_<data-maturity>\_NAD | cib\_dev\_app1\_NAD conn\_prod\_home\_partners\_NAD |
| Compute | <bu>\_<env>\_<compute-type>\_<id> | cib\_dev\_allpur\_01 |

# Tagging

Tags will be applied to databricks workflows and compute clusters based on Environment, Team, Project, Owner, CostCenter. These tag attributes will be captured by the system table for granular tracking of billing and are visible in audit logs. Along with these tags, follow SC tagging strategy for all Azure objects created for Unity Catalog.

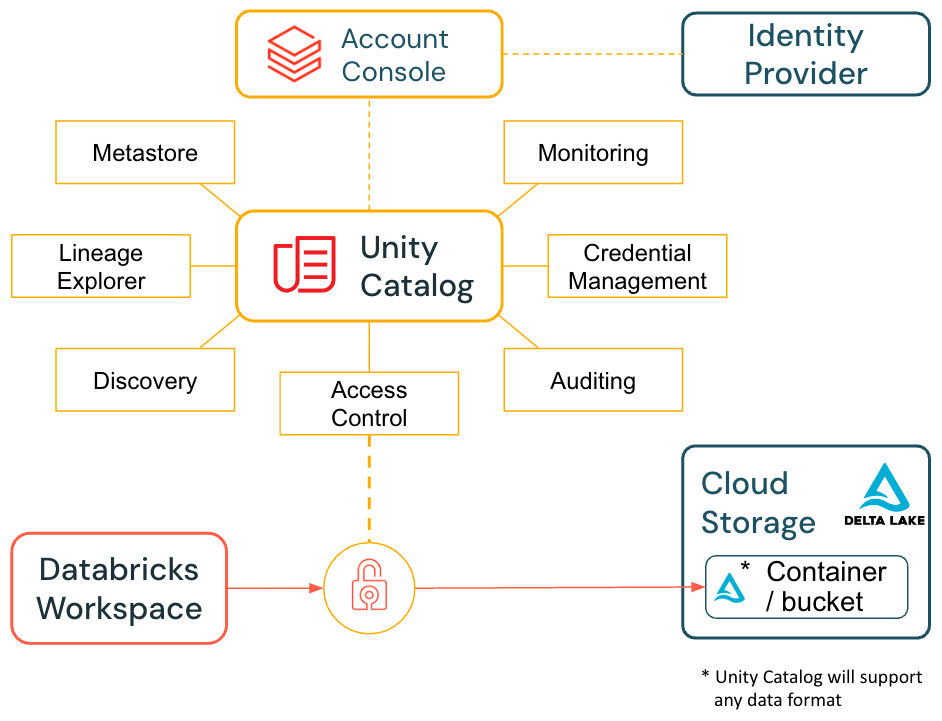
* We can use system level tags as well as the custom tags at the time of resource creation [link](https://docs.azure.cn/en-us/databricks/admin/account-settings/usage-detail-tags).
* These tags will then be used to group data to find out the cost incurred.
* Alerts can be set on top of these queried data for cost management.
* These tags can be enforced using Azure policy on Azure and Databricks [cluster policies](https://learn.microsoft.com/en-us/azure/databricks/admin/clusters/policy-definition) on Databricks.

| **Name** | **Value** | **Description** | **Type** |
| --- | --- | --- | --- |
| Project Code | ml-pipeline, datalake | Project the resource is used for | Custom tags |
| Cost Centre | cc-12345 | Financial tracking cost center | Custom tags |
| App Name | app1 | Resource application name | Custom tags |
| Team | team cash, team cc | Team name | Custom tags |
| Use Case | Use case name | Data Pipeline | Custom tags |
| Environment | dev, stg, prd | Environment resource belongs to | Custom tags |
| Usage Type | compute, storage | Type of workload | Custom tags |

# UC design Layout

Data isolation should primarily happen by business units and environments. In practice this means onecatalog per BU per environment (e.g. risk\_dev, risk\_prod, compliance\_dev, compliance\_prod) for risk assessment and compliance business units.

Within each catalog we use schemas (databases) for medallion layers (Bronze/Silver/Gold). This lets us manage Bronze/Silver data in the same catalog rather than creating separate catalogs for each layer. A ML specific schema (e.g. ml\_assets) can be used to hold reusable ML assets (feature tables, dimension data, registered models).



# Guiding principals

* Users can only gain access to data based on specified access rules.
* Data is physically separated in storage. Each BU will share their own Azure Storage account.
* User specific schemas in the sandbox catalog should only be used by data scientists for their DS/ML experimentation.
* Data scientists must not be allowed to make any changes to the bronze, silver, or gold schemas in any catalog:
* Data scientists may have read access to the masked silver, gold, and ml\_assets schemas in the production catalogs.
* Data scientists may have write access only to the ml\_assets schema in the sandbox/dev catalog and to their own individual schemas.
* Only the service account should be granted write access to all other catalogs and schemas.
* A central feature store catalog will be used for assets that need to be accessed across other business units or environments.
* Permissions and controls will be enforced at three levels: Unity Catalog permissions, CI/CD pipeline checks, and monitoring via system tables with alerting for policy violations.

# Business Units

BUs include (but are not limited to): CIB, WRB, ICS, GF, CDO. New BUs can be added in the future. Each BU will have their own Databricks workspace. Depending on the Azure Subscription limits, new azure subscriptions can be leveraged to create resources for the Business Units depending on [resource limitations](https://learn.microsoft.com/en-us/azure/databricks/resources/limits). Each BU will have its own storage account for storage isolation as per SC policy.

## Workspaces

Three workspaces will be deployed for Testing & Acceptance (T&A) or LLM Serving across environments: Development, Staging, and Production. Each Business Unit will receive three workspaces: Development, Staging, and Production. A sandbox workspace will be provided to access mask production data.

| ***ID*** | ***Item*** | ***Reasoning/Comments*** |
| --- | --- | --- |
| WS-01 | Separate Databricks workspaces for each BU. | This is to completely separate compute costs per BU. While the same can be achieved, by cluster tagging policies, there are other azure components like private endpoints which are shared for a single workspace. |
| WS-02 | One Sandbox, one Dev, one Staging and one Prod workspace to be created per BU. | The sandbox workspaces are for Data Scientist EDA and Experimentation. The dev workspace will have no data. Staging workspace will contain synthesized data. |
| WS-03 | T&A or LLM Serving dedicated workspaces | 3 workspaces i.e. Dev, Staging, Prod for T&A and LLM serving use case |
| WS-04 | Dedicated workspace for quarantine to validate the new open source LLM models. |  |

## Sub Domains

Sub Domain is a combination of BU and Teams within each BU i.e. CIB BU has CASH, CC, FM teams. Each sub-domain will have their own set of catalogs.

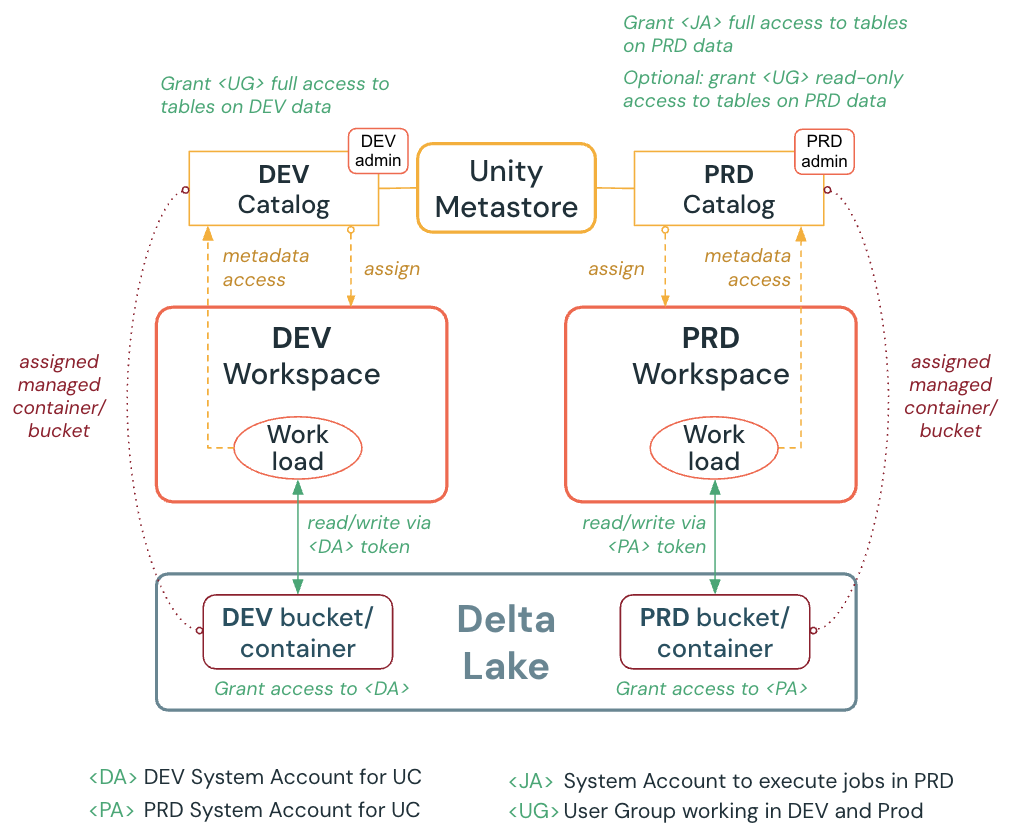
## Apps

Each sub domain can deploy multiple applications which will have their own medallion architecture. However, medallion architecture can be optionally controlled by a flag based on the use case.

## Catalogs

Catalogs will be segregated based on:

* **Environment** i.e. sandbox, dev, staging, prod
* Combination of **Business units** i.e.. CIB, WRB, ICS, GF, CDO etc. and **Teams** i.e. CIB has different Teams including but not limited to: CASH, CC, FM, etc.
* Each catalog will have its own Blob container assigned under the storage account accessed using Databricks Storage credential supported by Azure Managed Identity.



## 

## 

## 

## Schemas

* **Medallion layer schemas:** Create one schema per medallion layer—bronze, silver, gold for each App within a BU catalog, except the sandbox catalog.
* **ML asset schemas**: Create one schema for ML assets (including models, functions, and feature tables) within each catalog, except the sandbox catalog.
* **Individual data scientist schemas:** Within the sandbox catalog, create a dedicated schema for each data scientist to support DS/ML experimentation.

| ***ID*** | ***Item*** | ***Reasoning/Comments*** |
| --- | --- | --- |
| UC-01 | Only one metastore to be created per region. | This is a restriction on the databricks platform. |
| UC-02 | Catalogs to be created per Sub-domain. | There is a limit of 1000 catalogs for a single metastore. |
| UC-03 | The Sandbox, Dev, Staging and Prod catalog shall be restricted to an environment. | This is a requirement from the SC. |
| UC-04 | Schemas to be created per App/Usecase | The schemas in a catalog can follow the medallion architecture, and be created as thus: i.e. bronze\_<app1>, silver\_<app2> etc. |

### 

### Catalog-Schema setup for each sub-domain

| **BU** | **Team** | **Environment** | **Workspace** | **Catalog** | **App Name** | **Follow\_medallion flag?** | **Schema** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **<sub-domain>** | | **<env>** | **<domain>-<region>-<env>** | **<sub-domain>\_<env>** |  | **True/False** | **<data\_maturity/medallion\_layer>\_<app\_name>** |
| CIB | CASH | Sandbox | cib-we-dev | cib\_cash\_sandbox | - | - | <username>\_schema |
| Dev | cib-we-dev | cib\_cash\_dev | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| Staging | cib-we-stg | cib\_cash\_stg | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| Prod | cib-we-prd | cib\_cash\_prd | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| CC | Sandbox | cib-we-dev | cib\_cc\_sandbox | - | - | <username>\_schema |
| Dev | cib-we-dev | cib\_cc\_dev | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| Staging | cib-we-stg | cib\_cc\_stg | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| Prod | cib-we-prd | cib\_cc\_prd |  |  |  |
| WRB | PB | Sandbox | wrb-pb-dev | wrb\_pb\_sandbox | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |
| Dev | wrb-pb-dev | wrb\_pb\_dev | similar pattern | similar pattern | similar pattern |
| Staging | wrb-pb-dev | wrb\_pb\_stg | similar pattern | similar pattern | similar pattern |
| Prod | wrb-pb-dev | wrb\_pb\_prd | app1 | True | bronze\_<app1>  silver\_<app1>  gold\_<app1>  ml\_assets\_<app1> |
| False | <app1>  ml\_assets\_<app1> |
| app2 | True | bronze\_<app2>  silver\_<app2>  gold\_<app2>  ml\_assets\_<app2> |
| False | <app2>  ml\_assets\_<app2> |

### 

# 

# 

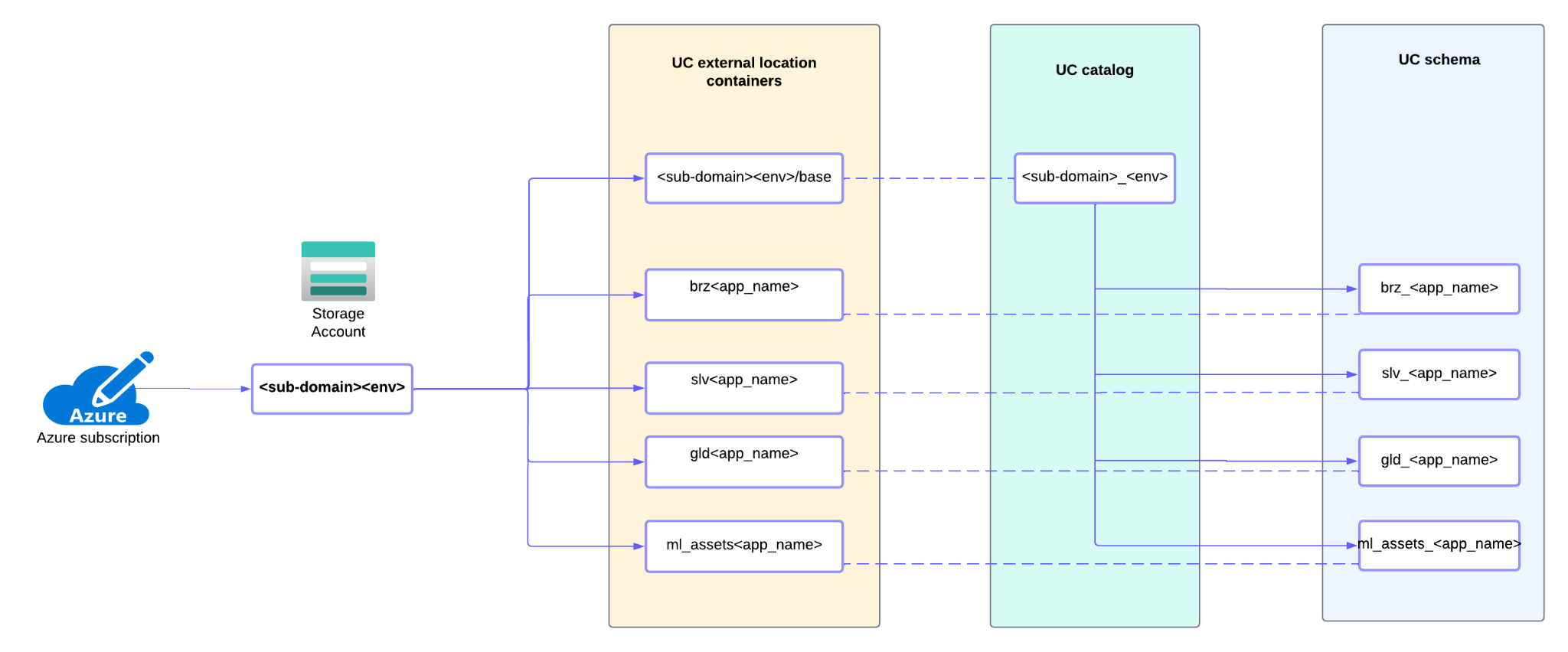
# 

# 

## Storage Layout

The storage account layout refers to the underlying ADLS gen2 storage accounts and containers to be created in the Azure cloud and to be registered as unity catalog artefacts in Databricks. The following section outlines some baseline recommendations for the storage accounts and containers to be created and registered as UC catalogs and schemas based on the discussion with SC.

| ***Type*** | ***Name*** | ***Storage Account/Container*** |
| --- | --- | --- |
| Catalog | <sub-domain>\_<env> | <sub-domain><env>/base |
| Schema | <medallion\_layer>\_<app\_name> | <sub-domain><env>/<medallion\_layer><app\_name> |



In the above section the patterns outlined, where the underlying storage account/container has separate storage account per BU with container denoting the layer for sub-domain catalogs.

**Note:** Consider the number of requests for accessing the data to avoid throttling. Do not leverage a single storage account/bucket for the tables if you have storage-intensive workloads. For example, Azure Storage Accounts supports [40,000 requests per second for the southeast Asia and West Europe region](https://learn.microsoft.com/en-us/azure/storage/common/scalability-targets-standard-account). This can cause workload throttling and slowdown. In this case, recommendation is to strip storage across different storage accounts since using different containers in the same storage account would still incur the same limit. Depending on the file size read by spark engine -: lets says - 4 MB file size (40,000 requests/sec × 4 MB) would translate to a read capacity of 160 GB/s, 40 MB file size (40,000 requests/sec × 40 MB) would translate to a read capacity of 1.6 TB/s

and 128 MB file size (40,000 requests/sec × 128 MB) would translate to a read capacity of 5.2 TB/s.

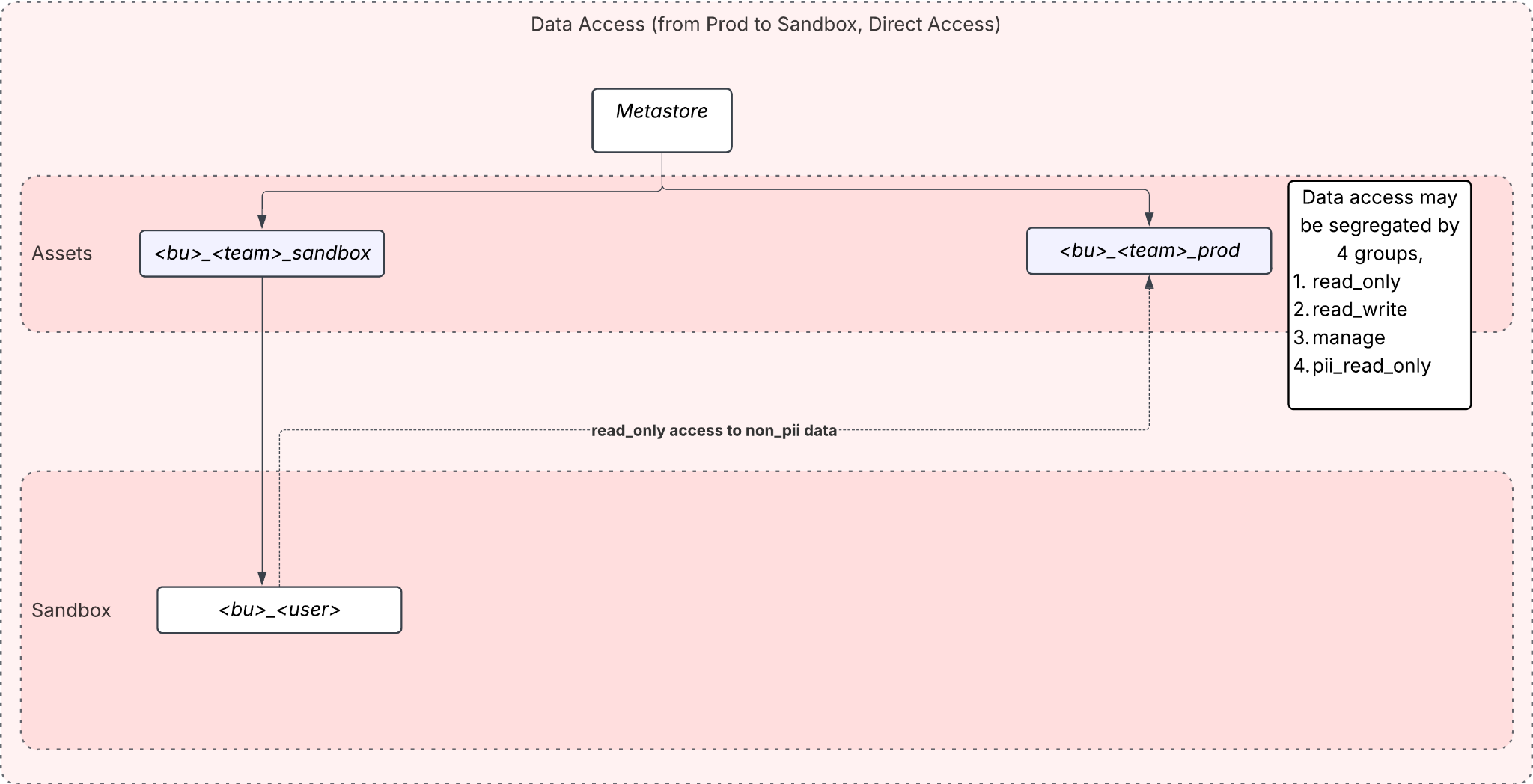
# 

# Data Access

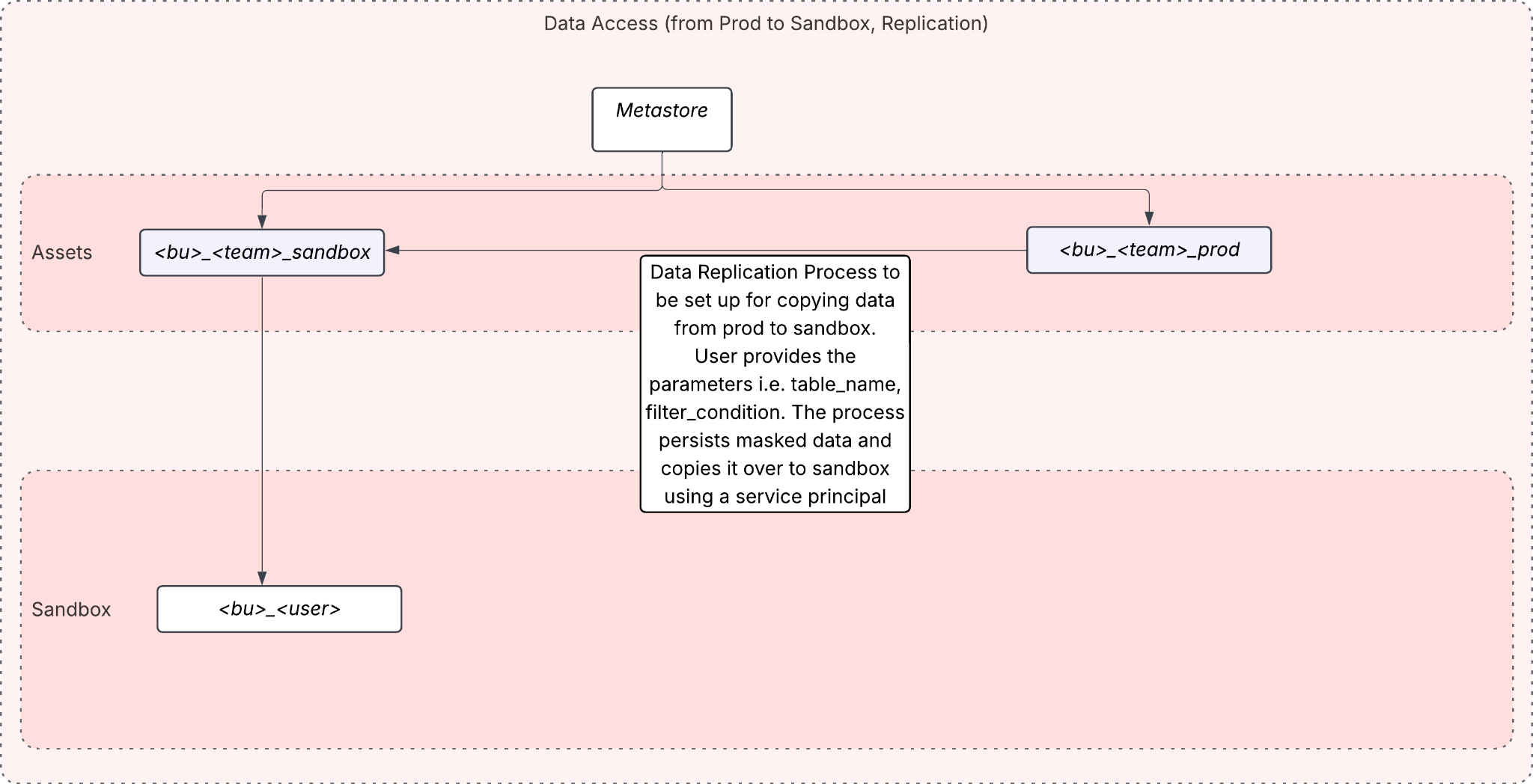
This section provides the guidelines for accessing production data by data scientists and engineers in the sandbox environment.

This can be achieved in one of two ways outlined below:

## Option-1: Direct Access



## Option-2: Data Replication



| **Options** | **Description** |
| --- | --- |
| 1- Direct Access | In this scenario, we propose the following:   1. Create the following data access groups for governance of prod data    1. **<data\_domain>\_ro\_pii**: users who are added to this group will be able to read all data including pii/sensitive data    2. **<data\_domain>\_ro**: users added to this group will only be able to read non-sensitive data    3. **<data\_domain>\_rw**: No users to be added to this group other than `service accounts` which are used to insert/modify the dataset as part of scheduled job runs.  | **Note**: *<data\_domain>* here refers to <bu>-<team>-<env>-<data\_maturity>-<app\_name>, i.e. `cib-cash-prod-bronze-app1` | | --- |  1. Allow access to the `<bu>-<team>-prod` catalog in the *sandbox* environment. 2. Add necessary data\_scientists/model\_engineer personas to <data\_domain>\_ro group, such that they may only have access to masked prod data in sandbox. |
| 2- Data Replication | In this scenario, we propose the following:   1. Scheduled jobs replicate masked subset of production data into a dedicated sandbox catalog/schema. |

| **Note**: We propose the *<data\_domain> groups* in the option-1 be created regardless of the data access pattern chosen. |
| --- |

# 

# Data Sharing

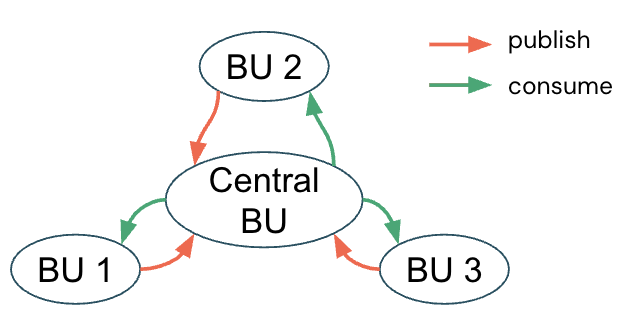
This section provides the guidelines for sharing data assets across business units and metastores.

The unified approach proposed for sharing data is using the `[Delta Sharing](https://learn.microsoft.com/en-us/azure/databricks/delta-sharing/)` feature which is provided out of the box as part of Databricks Platform.

## Overview

The data sharing approach outlined below follows the *centralized publishing* model, where

1. A central BU serves as a Hub
2. BUs are isolated from each other, e.g. by network access restrictions
3. The central BU provides
   1. central data storage and the central metastore to publish (data, metadata & ACLs) datasets for all BUs and usually
   2. central data governance and applies quality assurance on all published data
   3. platform operations (secure and compliant blueprints, infrastructure setups, …)
4. Often the central BU publishes own data (via a central data engineering team)



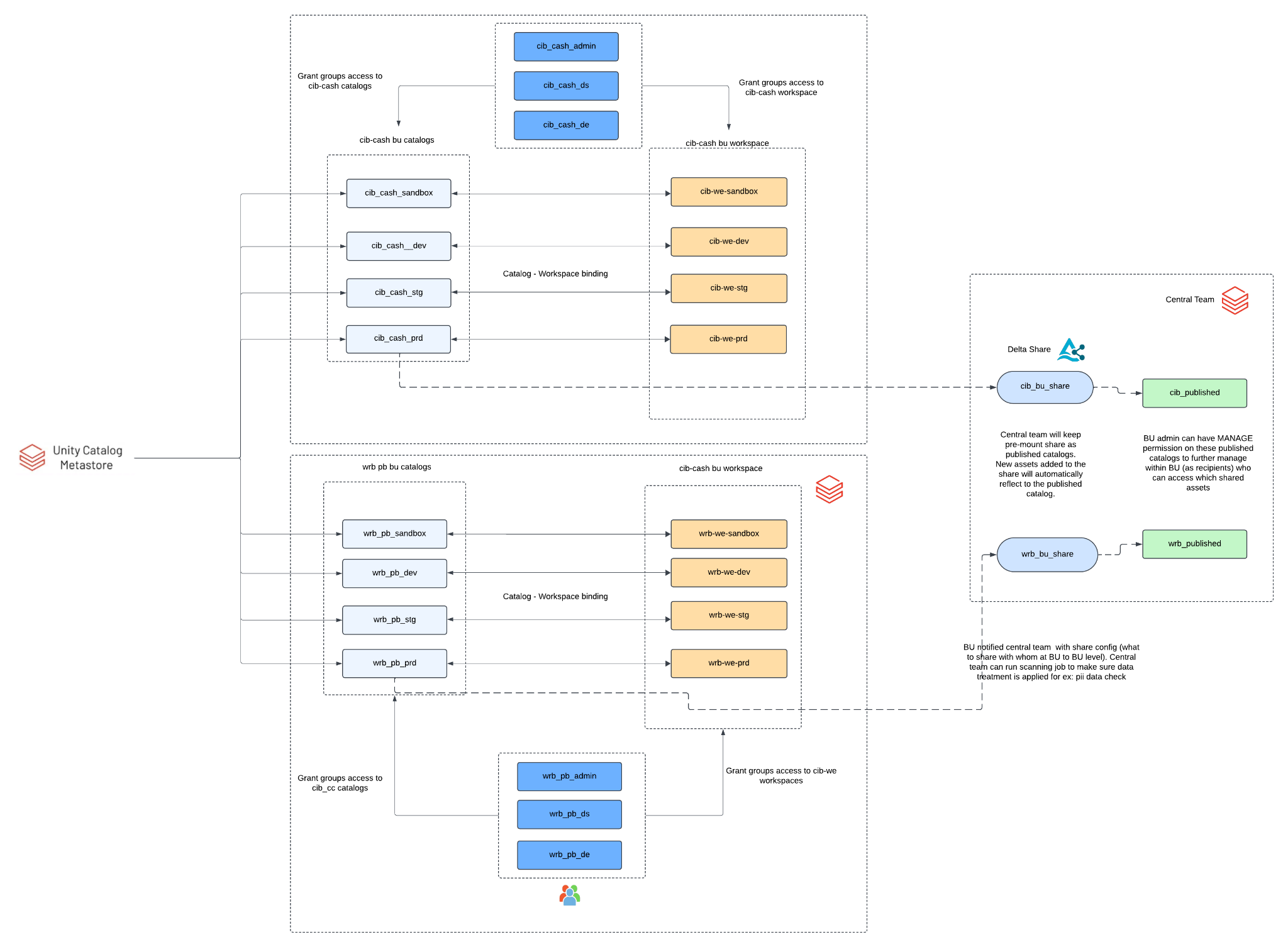
In the below section, we primarily discuss the catalogs to be created in the central BU as part of the delta sharing process.

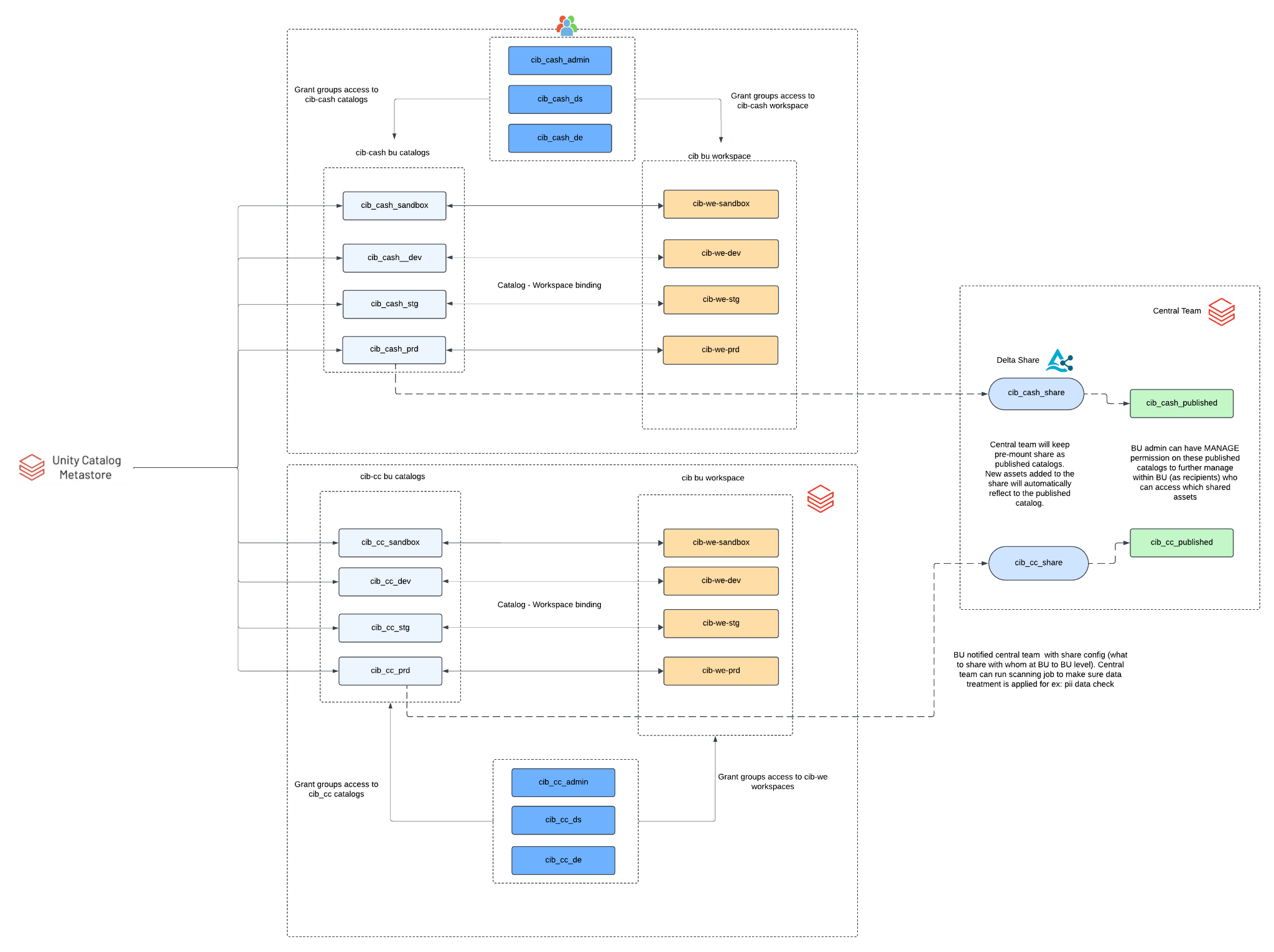
* Option-1: Multiple published catalogs from each BU
* Option-2: Single published catalog from all BUs

## Option-1: Multiple published catalogs from each BU

* Each BU will have a dedicated Delta sharing `[share](https://learn.microsoft.com/en-us/azure/databricks/delta-sharing/#share)` where they can add the assets they want to share.
* Each BU will have a dedicated published catalog.
* The sharing process must go through a CICD pipeline and must be validated by the CDO before approving or rejecting the sharing request.

| **BU / Shared** | **Team** | **Delta Share** | **Catalog** |
| --- | --- | --- | --- |
| **CIB** | CASH | cib\_bu\_share | cib\_published |
| CC | cib\_bu\_share | cib\_published |
| FM | cib\_bu\_share | cib\_published |
| **WRB** | PB | wrb\_bu\_share | wrb\_published |

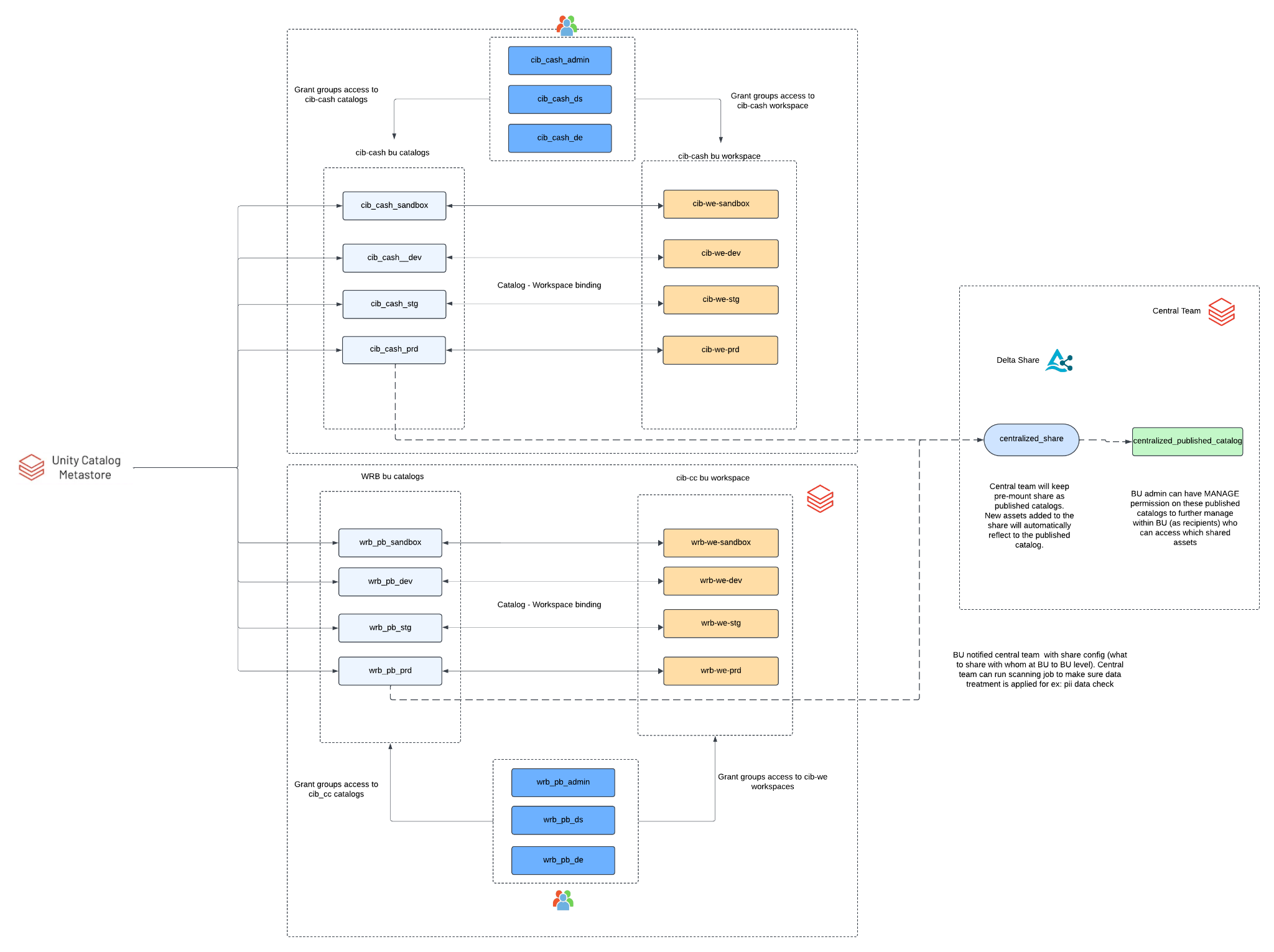




## Option 2: Single published catalog from all BUs

* All BUs to share data assets in the same delta sharing `[share](https://learn.microsoft.com/en-us/azure/databricks/delta-sharing/#share)`
* All BUs publish to a single centralized catalog..
* The sharing process must go through a CICD pipeline and must be validated by the CDO before approving or rejecting the sharing request.

| **BU / Shared** | **Team** | **Delta Share** | **Catalog** |
| --- | --- | --- | --- |
| **CIB** | CASH | centralized\_share | centralized\_published\_catalog |
|
|
| **WRB** | PB | centralized\_share | centralized\_published\_catalog |



**Note -** Pros and Cons of **[Option 1]** and **[Option 2]** are mentioned in the [Architectural Decision section](#_1pbaicn7b9gy)

# 

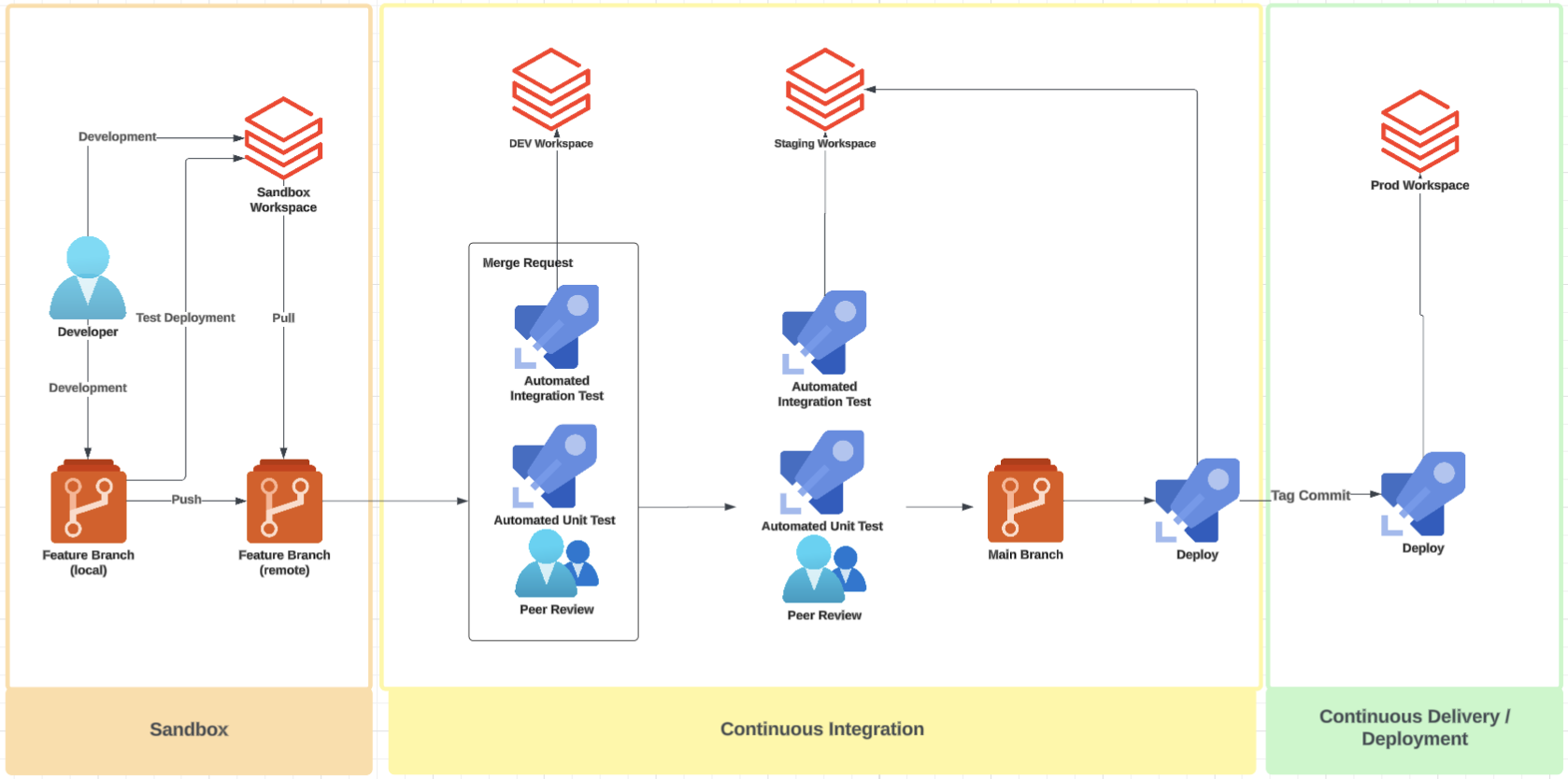
# Automation and CICD

## Overview

* All artefacts that influence the behaviour of the system are versioned (code, configurations,, models) → Easy rollback on error and traceability
* Code is deployed to production through an automated process steps → Fast, repeatable deployment and easy rollback
* All code changes go through an quality control process
* Merge requests with automated tests and manual code review
* Production, testing and development environments are isolated and access controlled
* Production environments are monitored to ensure correct behaviour
* Code is modularized to make it easy to understand, test and minimise repetition
* **Unity Catalog Assets** like catalogs, schemas, grants, connections, credentials will be defined in Terraform code in an ADO repository.
* **Code Assets and Table Metadata** like Notebooks, jobs, pipelines, feature table definitions, models etc. will be deployed using Databricks Asset bundles.

## High Level Flow

The overall CI/CD flow is represented in the below diagram.



* The developer works in a feature branch.
* Code is deployed manually or via automation to the Sandbox workspace.
* The developer iterates, and pushes changes to the remote ADO Git repository.
* An iterative merge/approval process in followed for dev and staging environments
  + CI pipeline triggers automated Unit and Integration Tests in the dev and staging environments.
  + The assets are deployed to the next environment on peer review and approval.
* A pull request (PR) is created by the Developer to integrate changes into the main branch.
* Pull request will require approval from `CODE OWNERS` which may be provided post peer review.
* On successful validation and approval, changes are merged into the main branch.

## Deployment of Unity Catalog assets

This section outlines the CI/CD process using which unity catalog assets will be deployed.

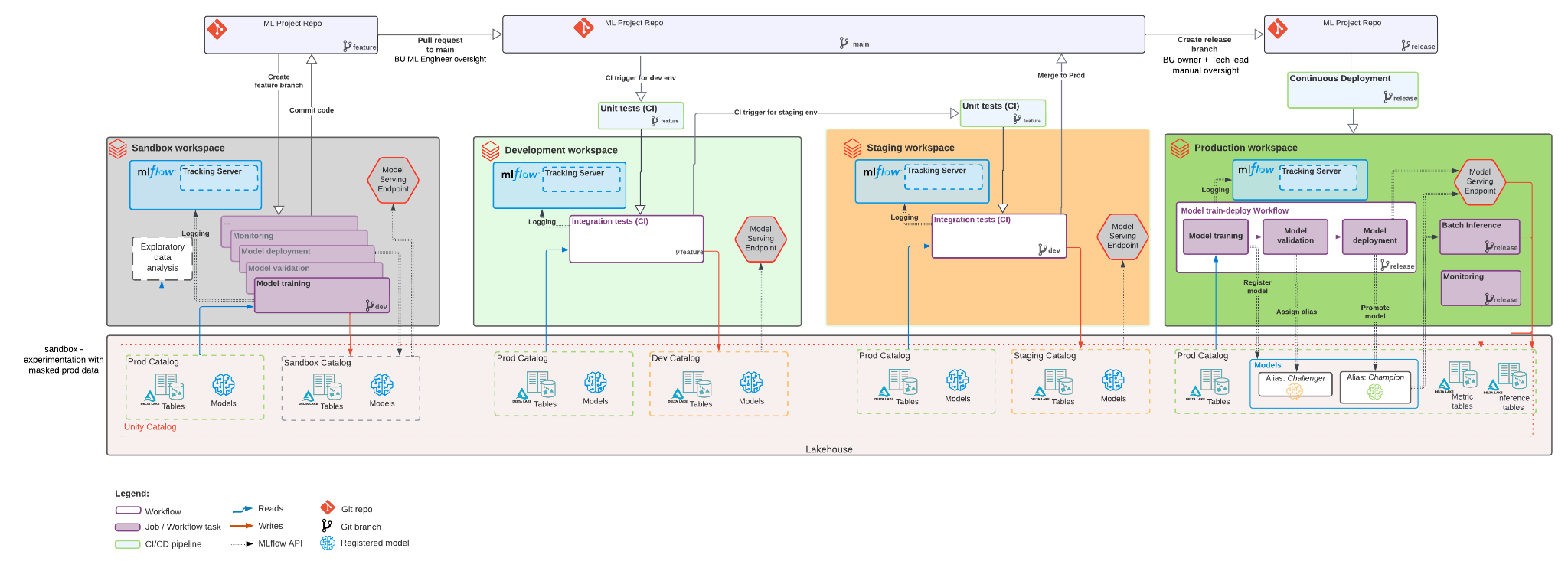
For details on the Unity Catalog object model, please refer to the documentation [here](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/#-the-unity-catalog-object-model).



* UC Objects will be configured using Terraform ([databricks\_terraform\_provider](https://registry.terraform.io/providers/databricks/databricks/latest/docs)) and deployed using Azure Devops(ADO) pipeline. This is the current pattern that is followed in SCB.
* As part of this pattern, the ADO Pipeline will include the standard governed templates, which has been pre-approved by the SCB teams.
* The pattern of resources being defined through YAML configs will be followed in this approach.
* The ADO pipeline will follow the standard approval process currently in place at SCB, i.e. approval required before merge to main branch, approval before deployment to environments etc.
* The ADO Pipeline will provide the Env, region and workspace parameters to the Terraform Module to identify the `Environment` and `Workspace` for which resources need to be deployed.
* The terraform module will contain configuration at the following level:
  + BU name
  + Team name
  + App/Use case name
  + Environment name
  + Medallion Flags
  + Custom schema names.
  + Table Definition

## Deployment of Code assets

* Each Business Unit will have their own ADO repository that can deploy tables, models, features using **Databricks asset bundles.**
* The asset bundles will deploy resources to the BU specific workspaces based on the target configuration for each environment and workflow.
* CI/CD pipelines will use environment variables for catalog, schema, and table names, and should include checks to prevent unauthorized writes to restricted schemas.
* The CICD pipeline should validate the catalog, schema names and does not let the user write unintended schema like bronze, silver and gold based on the environment(controlled by UC permissions).
* The CICD pipelines should include unit tests, integration testing in dev/stg workspace before deployment to Production workspace where manual approval from BU/sub-domain lead is required
* Pipelines in production can be executed as Databricks Workflows Jobs on schedule (using Cron syntax) or manually (one-off).



## Databricks Asset Bundles

Databricks Asset Bundles or DABs are a collection of Databricks artifacts 1 (e.g. jobs, ML models, DLT pipelines, and clusters) and assets 2 (e.g. Python files, notebooks, SQL queries, and dashboards). **Databricks Assets Bundles are an infrastructure-as-code (IaC)** approach to managing your Databricks projects. This section highlights the important aspects of using Databricks asset bundles. For a complete reference, please refer to the documentation [here](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/bundles/). These DABs are represented in a configuration file and can be co-versioned in the same repository as the assets and artifacts referenced in the bundle.

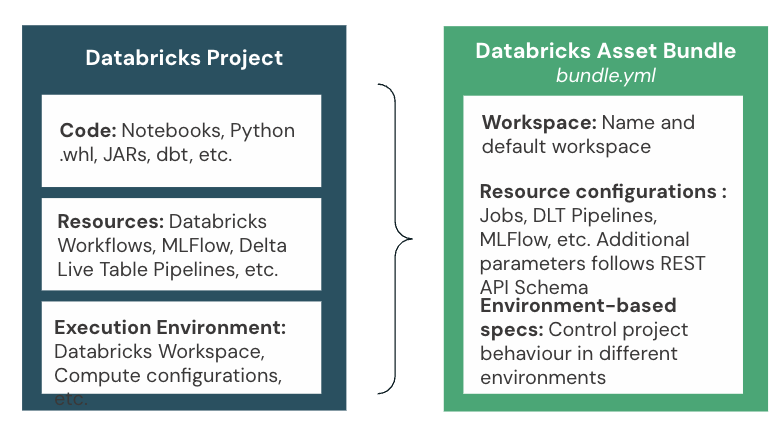
Using the databricks CLI these bundles can be materialized across multiple workspaces like dev / staging and production enabling customers to integrate these into their automation and CI/CD processes.

* YAML files that specify the artifacts, resources, and configurations of a Databricks project.
* The databricks CLI has functions to validate, deploy and run Databricks Asset Bundles using bundle.yml files
* Bundles are useful during development and CI/CD processes.

#### High-level view of a development and CI/CD pipeline with bundles:

# 

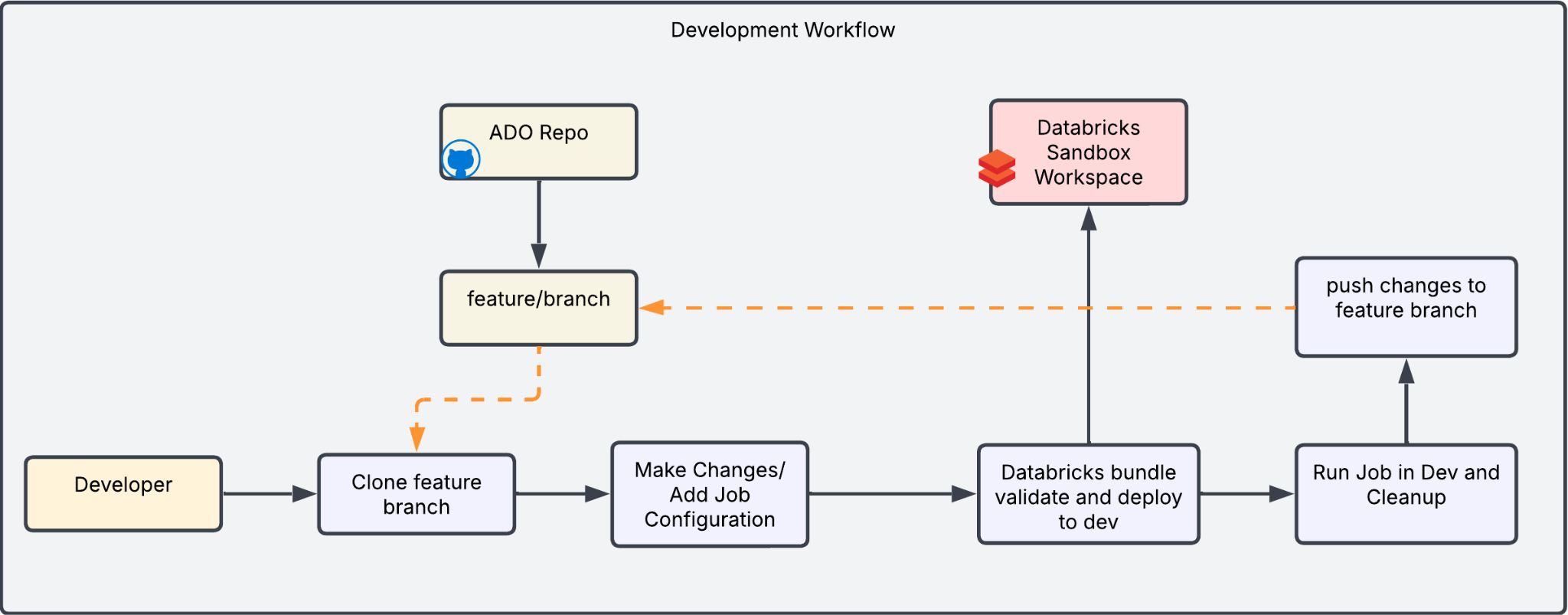
#### Components and Configuration





#### CICD with DABs

Change from Github repo to Azure repo

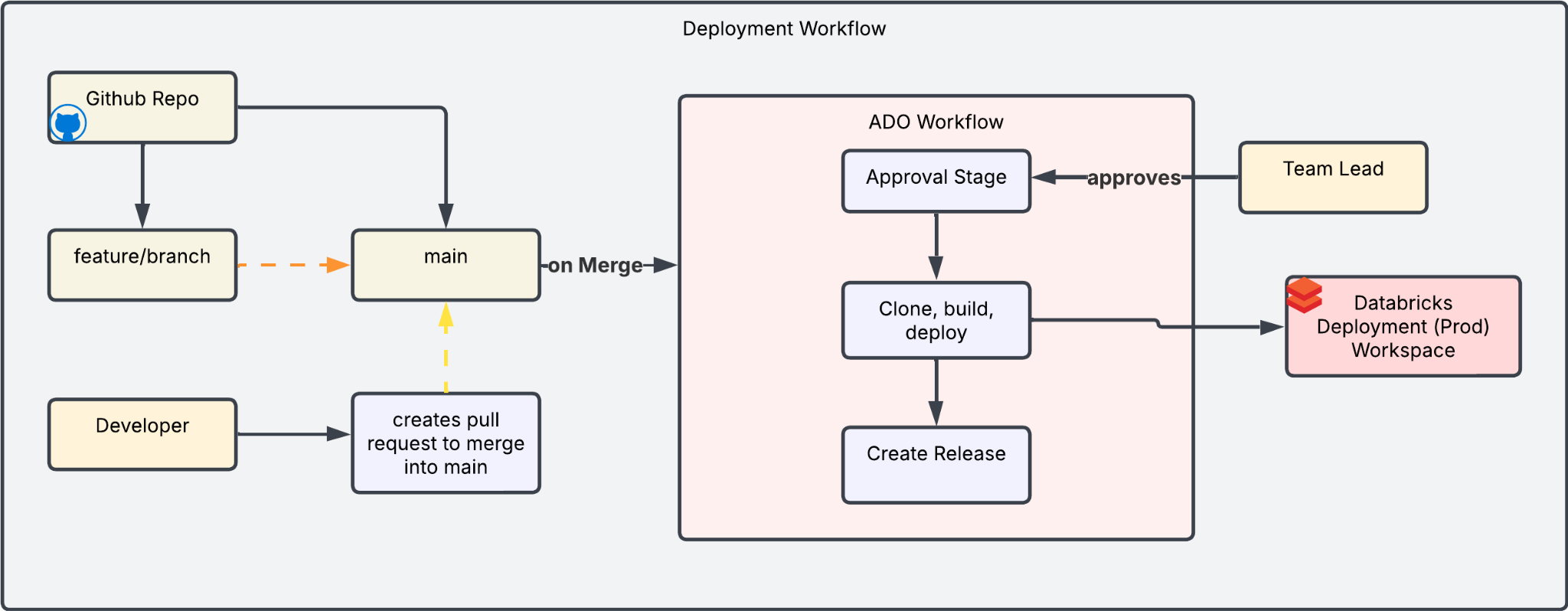


* **Developer** prepares the code from the AzureRepo
  + First, the developer **clones the feature branch** from **ADO Repository** to get the source code.
  + Then, they **make changes or add job configuration** based on the feature they are working on.
* **Validate the bundle in Sandbox/Dev**
  + Run $ databricks bundle validate to check if the databricks.yml file is correct.
  + This step helps catch problems early, like missing libraries or wrong settings.
* **Deploy the bundle to Sandbox/Dev**
  + Run $ databricks bundle deploy to deploy the bundle.
* **Run the job in Dev and clean up**
  + Run $ databricks bundle run to test the job in Dev.
  + After testing, run $ databricks bundle destroy to remove the notebook and job, so the workspace stays clean.
* **Push changes to the feature branch**
  + Once everything works, the developer pushes the changes to the feature branch.
  + A pull request (PR) is created and reviewed by teammates or stakeholders before merging.



* The developer prepares the code and tests/validates in a sandbox workspace.
* The developer pulls the existing bundle from the ADO Repo feature branchto get the source code.
* The developer adds existing jobs to the bundle using following command: $databricks bundle generate job --existing-job-id <job-id-from-ui>
* **Validate bundle**
  + Run $ databricks bundle validate to check if the databricks.yml file is correct.
  + This step helps catch problems early, like missing libraries or wrong settings.
* **Push changes to the feature branch**
  + Once everything works, the developer pushes the changes to the feature branch.

Deployment Workflow:



* **Creates pull request to merge into main branch**
  + The developer pushes changes to the feature branch.
  + A pull request (PR) is created and reviewed by teammates or code owners before merging.
  + After peer review, merge the PR to the main branch.
* Merge and trigger ADO pipeline
  + Once the PR is approved, the changes are merged to the main branch.
  + This will automatically trigger an Azure DevOps (ADO) workflow
* Approval Stage
  + Before deploying to production, it needs the team BU/lead’s approval.
* Clone, build, deploy
  + ADO will **clone the code**, then run $ databricks bundle validate and $ databricks bundle deploy again to deploy the latest files.
* Create Release
  + After successful deployment, a release is created.
* **~~Deploy the bundle to Dev~~**
  + ~~Run $ databricks bundle deploy to deploy the bundle.~~
* **~~Run the job in Dev and clean up~~**
  + ~~Run $ databricks bundle run to test the job in Dev.~~
  + ~~After testing, run $ databricks bundle destroy to remove the notebook and job, so the workspace stays clean.~~

—---

* **Developer** prepares the code/bundle from a **Git Repo**
  + First, the developer **clones the feature branch** from **Git Repository** to get the source code.
  + Then, they **make changes or add job configuration** based on the feature they are working on.
* **~~Validate the bundle in Dev~~**
  + ~~Run $ databricks bundle validate to check if the databricks.yml file is correct.~~
  + ~~This step helps catch problems early, like missing libraries or wrong settings.~~
* **~~Deploy the bundle to Dev~~**
  + ~~Run $ databricks bundle deploy to deploy the bundle.~~
* **~~Run the job in Dev and clean up~~**
  + ~~Run $ databricks bundle run to test the job in Dev.~~
  + ~~After testing, run $ databricks bundle destroy to remove the notebook and job, so the workspace stays clean.~~

##### Bundle commands across environments -



Deploy Databricks Asset Bundles across environments: Dev, Staging, and Production.

1. Dev Environment
   * In the Dev stage, developers test the bundle by themselves.
   * They can run and validate it locally or inside their Databricks workspace.
   * This is for early testing and development.
2. Staging Environment
   * When the code is merged into the **main branch**, it automatically triggers the CI/CD pipeline to deploy to staging.
   * The pipeline runs on a **CI/CD server**, using a **Service Principal** for secure access.
   * This staging environment is very close to production, so we use it to test before going live.
3. Production Environment
   * If everything works well in staging, we create the release branch.
   * This triggers the **production pipeline**, which again runs on the CI/CD server and uses a Service Principal.
   * It will then deploy the bundle to the production workspace.

##### DABs Project Structure

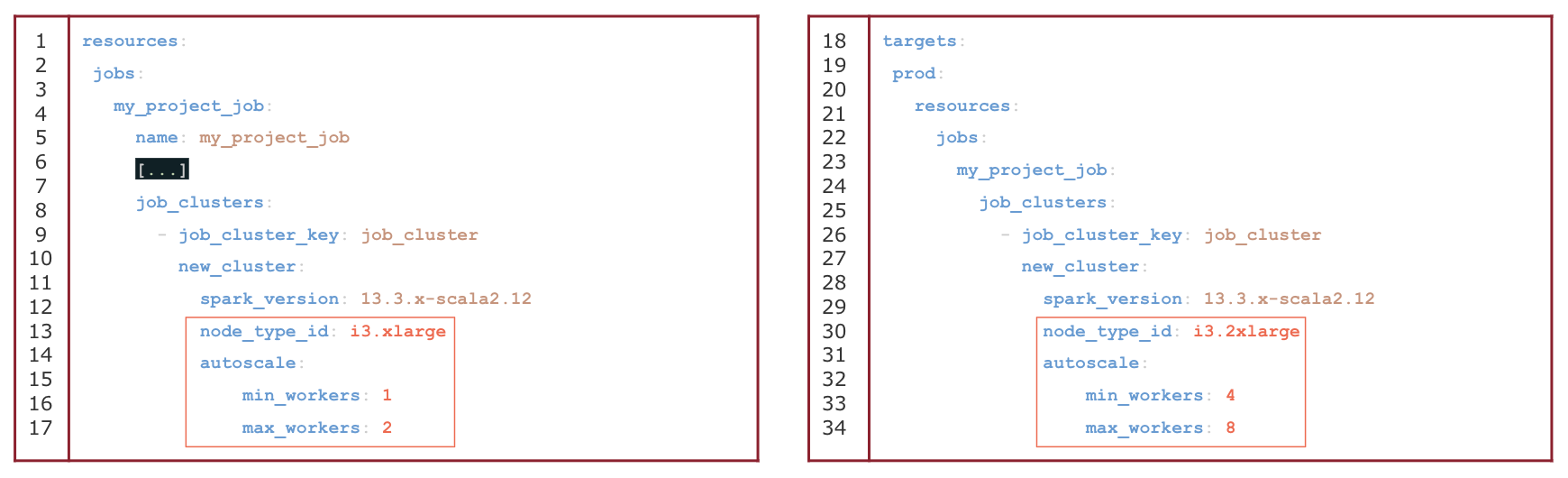
|  |  |
| --- | --- |

##### Configure stages in Databricks Asset Bundles

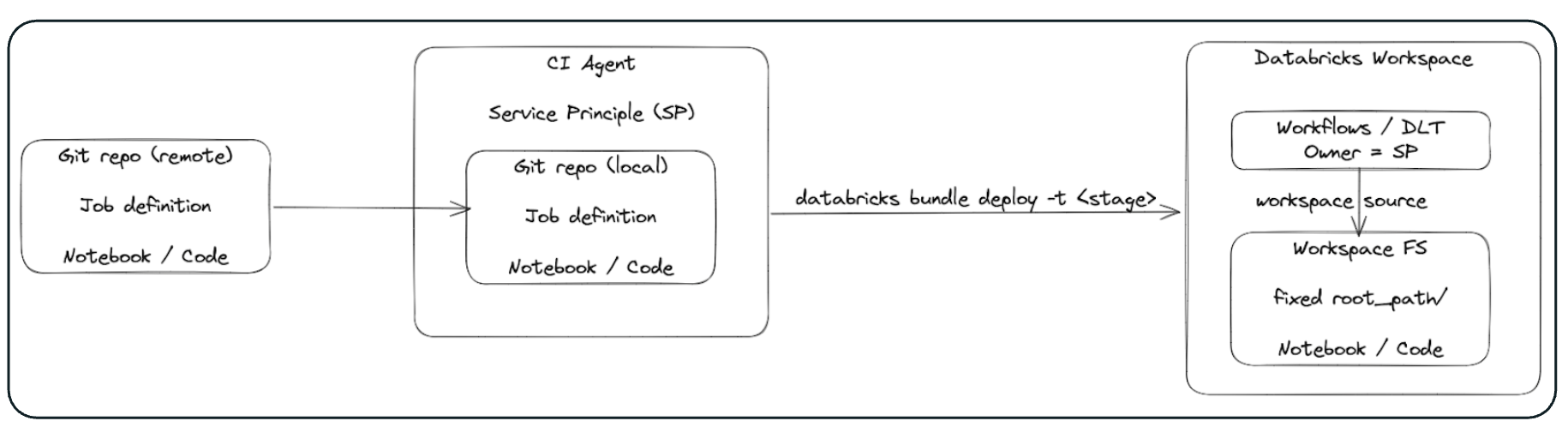
* Defines stage specific configurations
  + Overwrite job specific configurations such as cluster size in dev and prod
  + Target workspace
  + Mode: development or production
* Development Mode:
  + Pauses all schedules and triggers on deployed jobs
  + Attaches jobs to all purpose cluster if compute id is specified

##### Overwrite job specific definitions in DABs

* Other job specific definitions stay the same between stages e.g. task definition, schedule etc.
* In this example no of autoscaling workers and node type gets increased in prod



##### Deployment Pattern in DABs



* Within a workspace workflows are separated by owner (user) and code (Workspace FS / Repos) by user directory
  + Each user should create their own connection to the git repo in their user directory to separate code development in sandbox
  + Using dev tools (e.g. CLI) users can make test deployments from local env to their user dir in the sandbox workspace
* Run fast unit tests before slower integration tests
* Dev/Staging is used to:
  + (1) run integration tests triggered from CI pipeline during merge request
  + (2) do a pre-production deployment (pipelines run on a regular schedule) to monitor pipelines over some time
  + Separate (1) and (2) with different service accounts.
* Prod and Staging should be read only. Prod can sometimes even be restricted to privileged users only (e.g. when data is sensitive).
* The Prod environment must only have access to Prod data (e.g. via workspace restricted catalog, access control for service account).
* If prod data is not sensitive, the sandbox/dev environment can have (read only) access to all data (e.g. prod, dev).
* If prod data is sensitive, the sandbox environment can have (read only) access to the masked data.

## Table Deployment

In this section, we outline the options for deploying tables as part of the CICD process.

The options are outlined below:

* Option-1: Using Databricks Asset Bundles
* Option-2: Using Terraform ([databricks\_sql\_table](https://registry.terraform.io/providers/databricks/databricks/latest/docs/resources/sql_table) resource)

| **Note**: For both options, deployment across environments will follow the CI/CD process as defined in Section [`Automation and CICD/ High Level Flow`](#_iplv4mzbarn6) |
| --- |

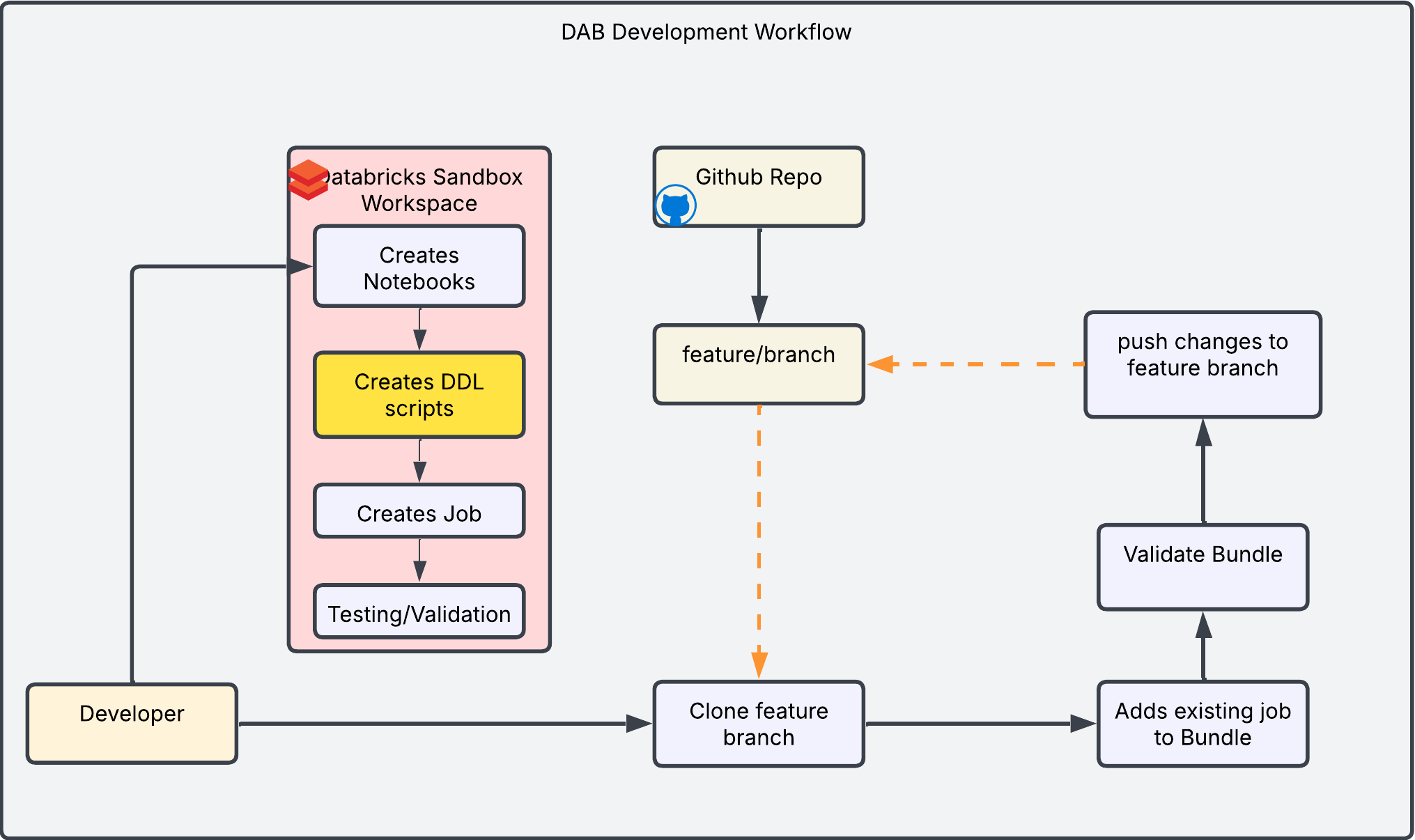
### Option-1: Using Databricks Asset Bundles

The table definitions will be deployed alongside code, using DDL scripts.

In this scenario, the developer creates the code assets (i.e. notebooks) and table definitions (DDL Scripts) and the databricks workflows.

All of these are integrated into the Databricks Asset Bundle configuration and deployed to the feature branch in Azure Devops Repo.

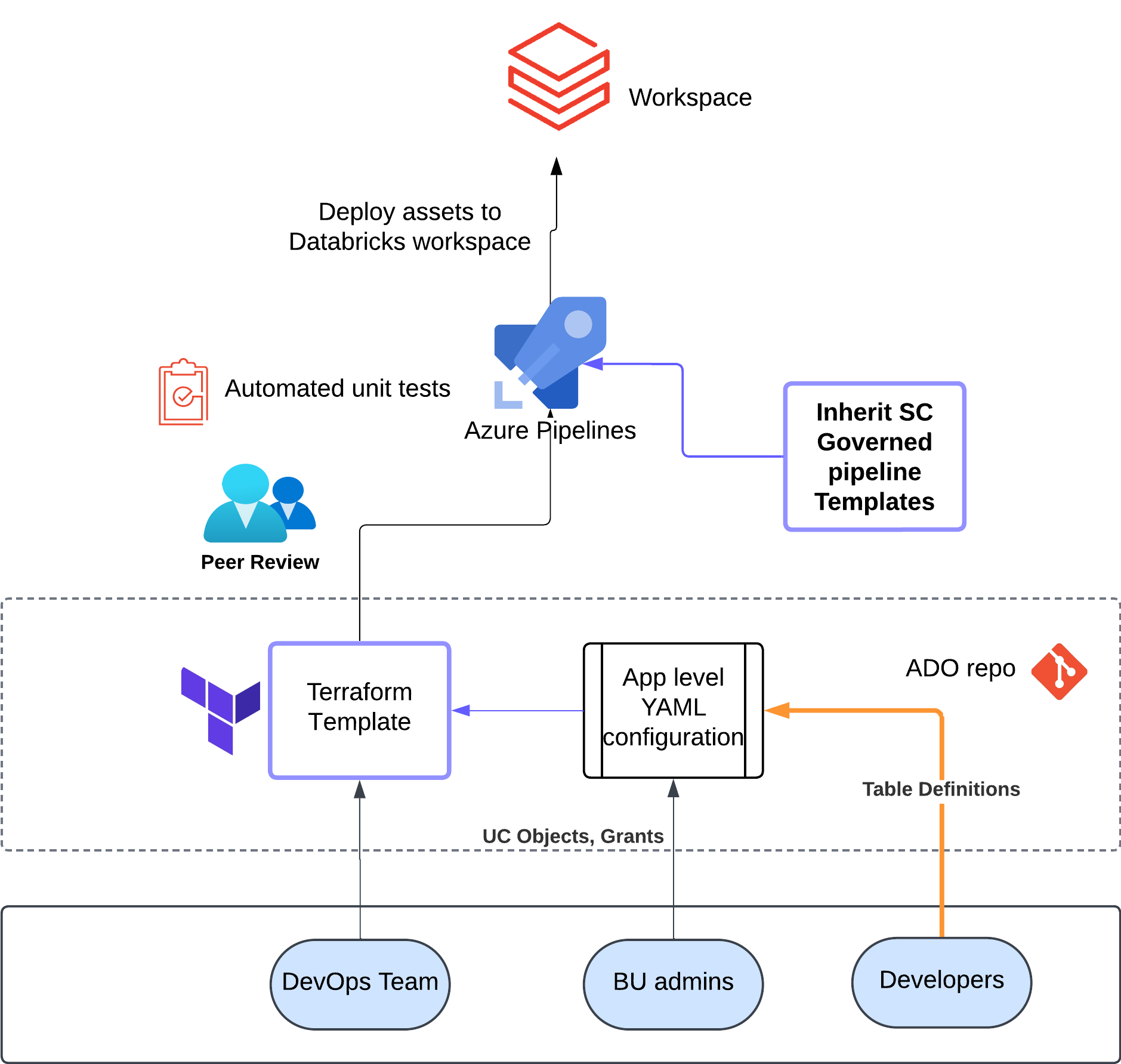
Deployment to development, staging and production environments will follow the standard CI/CD process as defined in the [Automation and CICD/ High Level Flow](#_iplv4mzbarn6) section.



### Option-2: Using Terraform (databricks\_sql\_table resource)

In this option, the following steps will be followed:

* Developers add table definition in the terraform YAML config
* Tables get deployed alongside unity catalog assets and the overall deployment across environments will follow the standard CI/CD process as defined in the  [Automation and CICD/ High Level Flow](#_iplv4mzbarn6) section.



## Persona Permissions

We define personas (and corresponding groups) with least-privilege roles in each environment at BU/Sub-domain level. Following personas will be merged with SC’s existing personas (workspace admin, workspace user, CDO) and workspace permissions to define fine grained access.

* **Data Scientists:** Use Databricks for ML, GenAI, building and training machine learning models, features and running experiments in sandbox/dev environments.
* **Data Engineers:** Manage data processing and transformation. They need scalable data processing, work in multiple programming languages, and design and build integrations with various data sources and storage systems.
* **AI Model Engineers:** Similar to data scientists (And DevOps) but with a stronger focus on operationalizing machine learning models, these engineers use things like the MLflow feature for the lifecycle management of machine learning models.

| ***Persona*** | ***BU*** | ***Comments*** |
| --- | --- | --- |
| bu\_data\_engineer | Per BU | Refers to the BU specific team of data engineers who work in conjunction with the platform data engineers. They work across the Spark (Python/Scala), SQL tech stack.  *[Data and Compute access is restricted per bu/use case.]*. |
| bu\_data\_scientist | Per BU | Refers to a team of data scientists working for specific BUs. They work in conjunction with bu\_data\_engineers for data processing optimization and are responsible for handling Model and Pipeline development in sandbox env. The tech stack to be accessed is Spark(Python, Scala), SQL, MLFlow, Python ML Libraries, and R.  *[Data and Compute access is restricted per bu/use case.]* |
| bu\_ai/model\_engineer | Per BUs | Refers to the engineers/team that handles the deployment of artefacts from DEV to PROD using a CI/CD process. |
| workspace\_admin | ALL BUs | Refers to the identified group of administrators who are responsible for managing the platform operations and overseeing the approval process for Data and Compute governance in unity catalog. |
| workspace\_users | ALL BUs | Basic browse permission to workspace metadata |
| workspace\_operators | ALL BUs | Persona in between workspace admin and bu admin |
| security | ALL BUs | Access to audit logs for security purposes. |
| cdo | ALL BUs | Refers to the Central Team which requires critical business operational information for overseeing and managing various organizational aspects and governance. They usually have permissions to browse and access lineage. |

### Permissions Consideration related to SC environment

Prevent data scientists from accidentally using the bronze, silver, or gold schemas. Databricks recommended the following measures to enforce this requirement:

* The Terraform templates for data governance must restrict both data scientists and the service account to write access only on the ml\_assets schemas. This will be implemented as a part of UC implementation.
* The CI/CD pipeline should validate variable values to ensure that users are not referencing the bronze, silver, or gold schemas.
  + For example, verify that the schema\_fe\_name variable is correctly set to ml\_assets. This is part of asset deployment(models, tables) included in the UC design but the implementation is done in a separate workflow not part of UC setup.
* Leverage Databricks system tables to audit user activity across the Workspace and Unity Catalog, and to detect suspicious actions. These system tables can be used to build monitoring queries and trigger alerts to notify the admin team of any suspicious activity such as:
  + A data scientist creating a table in the bronze, silver, or gold catalogs.
  + A data scientist writing data to an incorrect catalog

| **Environment** | **Medallion Layer Exist?** | **Write Permission** | **Reads Permissions** |
| --- | --- | --- | --- |
| **Sandbox(User)** | No | User schema only | Central Feature Store Production medallion(silver/gold) schemas Production ml\_assets schema |
| **Dev(service account)** | Yes | Dev ml\_asset schema | Dev medallion(silver/gold) schemas Central Feature Store  Production ml\_assets schema |
| **Staging(service account)** | Yes | Staging ml\_asset schema | Staging medallion(silver/gold) schemas Central Feature Store  Production ml\_assets schema |
| **Prod(service account)** | Yes | Prod ml\_asset schema | Prod medallion(silver/gold) schemas Central Feature Store  Production ml\_assets schema |

# Security and Compliance

## Production Data Masking

* Implement schema-level Access Control Lists (ACLs) using Unity Catalog to manage permissions for various user groups. This includes granular row and column-level access restrictions.
* Data masking appearance is customizable, offering options to substitute values with NULL, asterisks (\*\*\*\*), or more intricate SQL-based [mask](https://docs.databricks.com/aws/en/sql/language-manual/functions/mask) functions.
* Refer to the [01-Row-Column-access-control](https://notebooks.databricks.com/demos/uc-01-acl/) notebook for detailed examples and implementation guidance.
* Masking use case: Column level masking.
* Subset of data: Row level filter (give all rows where ID > 10)

## System Table

[System Tables](https://learn.microsoft.com/en-us/azure/databricks/admin/system-tables/) offer essential health and audit data for the Databricks environment, accessible through dedicated system tables. These can be activated per schema using either an [API](https://docs.databricks.com/api/azure/workspace/systemschemas/enable) or [Terraform](https://registry.terraform.io/providers/databricks/databricks/latest/docs/resources/system_schema). Detailed information on available tables and their functionalities is provided in the documentation. Security personnel will have access to the System table for auditing purposes.

| ***Type*** | ***System Table Name*** | ***Comments*** |
| --- | --- | --- |
| Billable Usage | *system.billing.usage* | Billable usage across the account. |
| Platform audit | *system.access.audit* | Includes audit events across all workspaces for an account per region. Refer to this [list](https://learn.microsoft.com/en-us/azure/databricks/admin/account-settings/audit-logs) for available audit events. |
| Query History | *system.query.history* | Includes all queries run on SQL warehouses along with serverless compute for notebooks and jobs. |
| Jobs | *system.lakeflow.jobs* | Tracks all jobs created in the account. |
| Clusters | *system.compute.clusters* | Contains a full history of compute configurations over time for any cluster. |
| Table Lineage | *system.access.table\_lineage* | Contains read or write events for unity catalog managed table or path. |

| *System tables are enabled at the schema level. Only the billing schema is enabled by default. For steps to enable/disable other necessary schemas, please refer to the documentation* [*here*](https://learn.microsoft.com/en-us/azure/databricks/admin/system-tables/#enable)*.* |
| --- |

#### Information Schema

System tables also provide a base *information\_schema* which returns information about the objects across all catalogs within the Unity Catalog metastore. For a complete reference to information schema and ER diagram, please refer to the databricks documentation [here](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-information-schema#entity-relationship-diagram-of-the-information-schema). Some important references are described below.

| ***Info Type*** | ***Table Name*** | ***Sample Query*** |
| --- | --- | --- |
| *Tables* | *system.information\_schema.tables* | select \* from system.information\_schema.tables where catalog = 'my\_catalog' and table\_type = 'MANAGED' and table\_schema = 'my\_schema'   |
| *External Locations* | *system.information\_schema.external\_locations* | select \* from system.information\_schema.external\_locations where url like '%my\_container@my\_storage\_account%'   |
| *Column Masks* | *system.information\_schema.column\_masks* | -- functions used as column masks  select mask\_catalog,  mask\_schema,  mask\_name,  count(1) as usage\_count  from system.information\_schema.column\_masks  group by 1,2,3  order by 1,2,3   |
| *Row Filters* | *system.information\_schema.row\_filters* | -- functions used as row filters  select filter\_catalog,  filter\_schema,  filter\_name,  count(1) as usage\_count  from system.information\_schema.row\_filters  group by 1,2,3  order by 1,2,3   |

# Reference Links

| [Isolation of environments on the Databricks Platform](https://community.databricks.com/t5/technical-blog/isolation-of-environments-on-the-databricks-data-intelligence/ba-p/56737) |
| --- |
| [Define environment isolation strategy](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-architecture/operational-excellence/best-practices#define-environment-isolation-strategy) |
| [Unity catalog best practices](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/best-practices#organize-data) |
| [Azure Databricks Resource Limits (Search: Unity Catalog)](https://learn.microsoft.com/en-us/azure/databricks/resources/limits) |

# Limits

| Metastore | One per account per region | Fixed |
| --- | --- | --- |
| Catalog | 1000 per Metastore | Not Fixed |
| Schema | 10000 per Catalog | Not Fixed |
| Workspaces | NA |  |
| Users | 10000 per account | Not Fixed (250k on Demand) |
| Groups | 5000 per account | Not Fixed (50k on Demand) |
| Job Tasks (concurrent) | 2000 per workspace | Fixed |
| Clusters | [NA, Available IPs] |  |
| Users (concurrent execution contexts) per Cluster | 145 per cluster | Fixed |
| External Locations | 10000 per Metastore | Not Fixed |
| Storage and Service Credential1 | 200 per Metastore | Not Fixed |

[Azure VM Limits](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-subscription-service-limits#azure-virtual-machines-limits---azure-resource-manager) 25000 VMs per subscription per region

—-------------------------------------

—-------------------------------------

### Trimmed part -

### Stages and Lifecycle

#### EDA & Experimentation (Sandbox)

* Data scientists will leverage notebooks for exploratory data analysis, feature engineering, and developing numerous models.
* Sandbox will allow individual data scientists to experiment in isolated environments using masked production data.
* Codebase will be version controlled, with feature/main/release branching & PR strategy following MLOps CI/CD.
* Data Scientists or Developers can use VSCode to remotely connect to Databricks clusters in order to write and execute code across Notebooks and Jobs.
* Data Scientists or Developers can use Databricks notebooks in the browser to execute codes and Jobs interactively.
* Databricks Runtime for Machine Learning enables data scientists to use ML frameworks and MLflow experiments to manage and monitor their experimental work.
* Once EDA is completed, Data scientists engage in machine learning model development and training, leveraging engineered features, conducting training experiments, fine-tuning hyperparameters, and rigorously comparing model performance.
* Experiment tracking is achieved using MLflow, which logs parameters, metrics, and artifacts. Upon identifying a satisfactory model, they can commit training notebooks and configuration files to the ADO repository.
* Experimentation notebooks can be decomposed into modularized components, facilitating the creation of workflows to execute each component.
* Unit tests and integration tests for the ML code can be added in this phase.
* Data Scientists will create a feature branch, commit their changes, and raise a pull request to the main branch.

#### Dev & Staging

* Once the pull request is raised to the main branch, CI unit and integration tests will run on dev and staging environments sequentially.
* The result of these tests will be reviewed by ML Engineers.
* Ensuring the successful execution of unit tests and Databricks Workflows testing including integration and performance falls under the ML engineer's responsibility.
* Assuming all validations succeed, the model version can be assigned the "Challenger" alias within Unity Catalog.
* The Pull Request can be merged to the main branch if the tests are passed successfully.

#### Production

* For deployment to production, cut a new release branch to promote ML code changes to the production environment.
* The release branch deployments will be reviewed and approved by the BU admin.
* Once approved, databricks workflows should run as required for each specific use case.
* Production workflow should include training, evaluation, validation deployment, inference and monitoring.

The table below summarizes an example of how environment variables should be defined in the CI/CD lifecycle, using the demand forecasting use case as a reference:

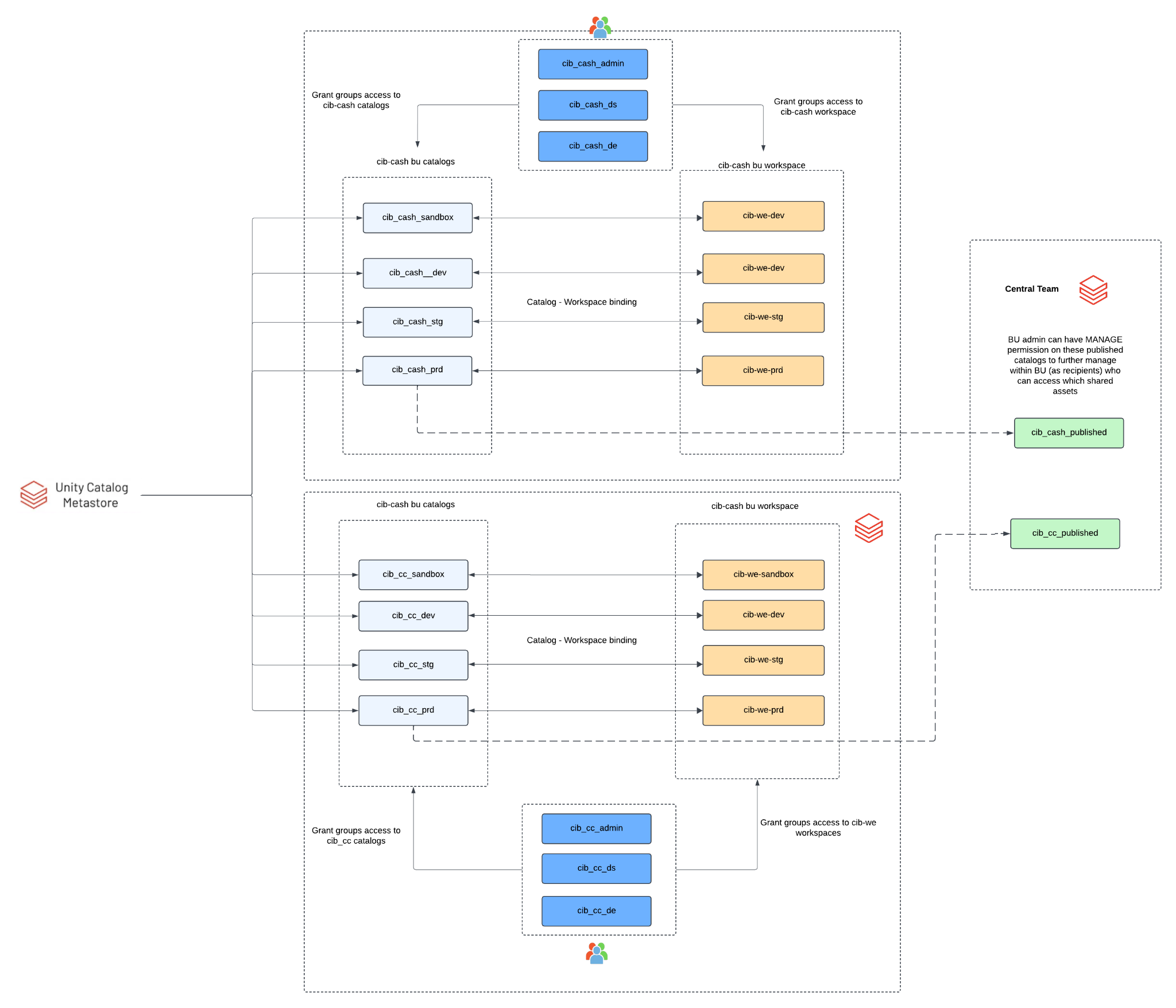
**Feature engineering workflow or Model Training workflow:**

| **Workflow** | **Feature engineering (FE)** | | | |
| --- | --- | --- | --- | --- |
| **Env** | Sandbox | Dev | Staging | Production |
| **Remarks** | Experimentation using Sandbox catalog and feature branch. | Promotion to the Main branch and using the DEV catalog. | Promotion to the Main and using the Staging catalog. | Promotion to Production and using the Production catalog. |
| **Catalog** | **catalog\_fe\_name:**  cib\_cash\_sandbox | **catalog\_fe\_name:**  cib\_cash\_dev | **catalog\_fe\_name:**  cib\_cash\_stg | **catalog\_fe\_name:**  cib\_cash\_prd |
| **Schema** | **schema\_fe\_name:**  <user\_name>\_schema | **schema\_fe\_name:**  ml\_assets | **schema\_fe\_name:**  ml\_assets | **schema\_fe\_name:**  ml\_assets |
| **Table** | **table\_fe\_name:**  demand\_forecasting\_features | **table\_fe\_name:**  demand\_forecasting\_features | **table\_fe\_name:**  demand\_forecasting\_features | **table\_fe\_name:** demand\_forecasting\_features |

## 

## Within Metastore

* Every Metastore will incorporate a central feature store, which contains segregation at schema level for each application (BU + Team + Application combination), thereby ensuring shared asset isolation.
* The central feature store will be used to share feature tables across different business units, applications, and environments within the same Unity Catalog metastore.
* The sharing process must go through a CICD pipeline and must be validated by the CDO before approving or rejecting the deployment to the Central feature store.



| **BU + Team + Application** | **Catalog** | **Schema(Sub\_domain level)** |
| --- | --- | --- |
| CIB + CC + App1 | ai\_factory\_feature\_store\_<region> | cib\_cash\_app1 |
| CIB + CC + App2 | ai\_factory\_feature\_store\_<region> | cib\_cash\_app2 |
| CIB + CC + App1 | ai\_factory\_feature\_store\_<region> | cib\_cc\_app1 |
| CIB + CC + App2 | ai\_factory\_feature\_store\_<region> | cib\_cc\_app2 |
| WRB + PB + App1 | ai\_factory\_feature\_store\_<region> | wrb\_pb\_app1 |
| WRB + PB + App2 | ai\_factory\_feature\_store\_<region> | wrb\_pb\_app2 |

## 