# Azure Data Mesh Engineering Standards

## 1. Overview

This High-Level Design (HLD) and Standards document provides the blueprint for designing, developing, deploying, and operating Azure-based data products in a data mesh architecture using Synapse, PySpark, Event Hub, and Azure Functions. It sets out consistent engineering practices, naming conventions, and CI/CD models to ensure quality, scalability, and maintainability across environments.  
  
Detailed Scope:  
• Defines the technical architecture, coding standards, and best practices for PySpark development in Synapse, following schema-first, idempotent, and parameterized design principles.  
• Establishes uniform naming conventions for code elements, data assets, pipelines, and storage paths to improve readability, discoverability, and governance.  
• Provides guidelines for creating and managing a shared common library (wheel) containing reusable utilities such as date/time functions, audit column handling, Delta Lake I/O helpers, and Azure Table audit writers.  
• Details the recommended repository layout, separating Synapse artifacts, PySpark code, configuration overlays, schemas, tests, and infrastructure as code.  
• Defines the CI/CD process in Azure DevOps across Dev, Test, and Prod subscriptions, incorporating automated build, validation, environment-specific deployment, and gated promotions.  
• Outlines a branching strategy that supports feature development, controlled releases, regression testing, and stable production deployments.  
• Specifies standards for Event Hub naming, partition management, capture-to-ADLS configuration, schema versioning, and DLQ handling, prioritizing Managed Identity for secure access.  
• Details Azure Functions design patterns for event-driven ingestion, including trigger configuration, environment settings, DLQ handling, audit integration, and deployment practices.  
• Establishes monitoring and observability standards using Azure Monitor, Application Insights, Log Analytics, and Audit Tables to track performance, SLA compliance, and cost efficiency.  
• Covers security and compliance controls including RBAC, private endpoints, Key Vault integration, and zero-secrets-in-code policy.  
• Includes testing strategy definitions (unit, contract, integration) and operational runbooks to ensure robust, scalable, and auditable data product operations in Dev, Test, and Prod.

## 2. Designing – Engineering View

This section focuses on engineering implementation design principles and patterns, complementing the enterprise architecture owned by the Architecture Team. For complete system architecture diagrams and enterprise design decisions, refer to the official Enterprise Architecture documentation.  
  
Design Principles:  
• Data Mesh-aligned — Each data product is independently developed, deployed, and owned, with common platform standards.  
• Separation of Concerns — Isolate Synapse artifacts, business logic, configuration, schema definitions, and infrastructure-as-code.  
• Environment Parity — Consistent architecture across Dev, Test, and Prod, deployed to separate subscriptions.  
• Cloud-native Security — Managed Identity-first for all service connections, private endpoints in non-dev environments, and Key Vault for secrets.  
• Observability Built-in — Audit logging, data quality metrics, and pipeline telemetry by default.  
  
Engineering Architecture Components:  
1. Synapse Analytics — Parameterized pipelines and PySpark notebooks.  
2. PySpark Processing Layer — schema-first transformations using shared common\_lib.  
3. Event Hub — ingestion and capture to ADLS.  
4. Azure Functions — event-driven ingestion and DLQ handling.  
5. Delta Lake Storage Zones — raw/bronze/silver/gold/recon/audit.  
6. Azure DevOps CI/CD — build, validate, deploy, promote.  
7. Infrastructure-as-Code — Bicep/ARM templates.  
8. Monitoring & Alerting — Application Insights, Log Analytics.

## 3. Quick Glance

|  |  |  |
| --- | --- | --- |
| Area | Abstract | Details |
| Coding Standards | Schema-first PySpark in Synapse; modular, idempotent, no secrets in code. | See Section 4.1 |
| Naming Conventions | Functions=verb\_noun, DataFrames end \_df, columns=snake\_case. | See Section 4.2 |
| Common Library | Shared wheel for utilities; versioned, tested. | See Section 4.3 |
| Repo Layout | Separate Synapse artifacts, PySpark code, configs, schemas, tests. | See Section 4.4 |
| CI/CD Model | Azure DevOps multi-subscription, gated Dev→Test→Prod. | See Section 4.5 |
| Branching Strategy | feature/\* to Dev, release/\* to Test, main to Test, release/\* to main to Prod. | See Section 4.6 |
| Event Hub Standards | Namespace per product/env, UTC timestamps, Capture to ADLS. | See Section 4.7 |
| Azure Functions Standards | Event-driven ingestion, MI-first, KV refs, DLQ path. | See Section 4.8 |
| Monitoring & SLOs | Audit Table + App Insights + Log Analytics. | See Section 4.9 |
| Security & RBAC | MSI + RBAC for storage/KV/EH, private endpoints. | See Section 4.10 |

# Detailed Glance – Full Standards

## 4.1 Detailed Coding Standards

* Project & Code Org: Thin main() → reusable functions; package as wheel; notebooks orchestrate, not implement.
* Config: JSON/YAML env overlays; no literals or secrets; config validated at load.
* Spark Session: AQE, DPP on; shuffle partitions tuned to data size; UTC timezone.
* DataFrames: Schema-first; built-in Spark funcs; avoid RDD unless required.
* I/O: Delta default; zone structure follows landing/historical/harvest/publish.
* Performance: Join clean keys; handle skew; avoid unnecessary cache.
* Idempotency: Safe re-runs; watermarks/high-water marks; Delta MERGE with keys.
* Observability: Structured JSON logs; row counts; DQ metrics per zone.

## 4.2 Detailed Naming Conventions

* Functions: verb\_noun() (e.g., load\_landing\_sales()).
* Variables: snake\_case (e.g., is\_incremental\_load).
* DataFrames: \_df suffix; prefix with zone if relevant (e.g., landing\_sales\_df).
* Columns: snake\_case; audit cols \_ingest\_ts, \_run\_id, \_src\_system.
* Tables: <domain>\_<product>\_<layer>\_<entity> (e.g., retail\_fdp\_harvest\_sales).
* Paths: <zone>/<product>/<entity>/year=YYYY/month=MM/day=DD/.
* Files: <entity>\_<yyyymmdd>\_<run\_id>.parquet.

## 4.3 Detailed Common Library

* Modules:
* - timeutils.py: parse\_date(), to\_utc(), iso\_utc()
* - audit.py: add\_audit\_columns(), AuditTableClient (job start/metrics/end)
* - io.py: read\_delta(), write\_delta(), merge\_delta(), compact\_delta()
* - ids.py: new\_run\_id(), attach\_correlation()
* Principles: Versioned (SEMVER), tested, MI-first, no env-specific logic inside.

## 4.4 Detailed Repo Layout

* Zones in ADLS:
* - landing/ → Raw ingestion/Event Hub capture
* - historical/ → Immutable, append-only full history
* - harvest/ → Curated for internal domain consumption
* - publish/ → Consumable outputs for external consumers
* Repo Folders:
* /synapse/ # ARM-exported artifacts
* /code/ # PySpark jobs
* /config/dev|test|prod/
* /schemas/ # StructType JSON / DDL
* /tests/ # pytest
* /functions/ # Azure Functions
* /infra/ # Bicep/ARM

## 4.5 Detailed CI/CD Model

* Pipeline Flow:
* - Landing → Validate schema/DQ → write to Historical
* - Historical → Transform → write to Harvest
* - Harvest → Apply business rules → write to Publish
* Environments: Dev, Test, Prod in separate subscriptions.
* Azure DevOps: Build (lint, test, package), Deploy (infra → code → Synapse → Functions).
* Promotion: Dev → Test (manual gate) → Prod (tagged release + regression).

## 4.6 Detailed Branching Strategy

* feature/\* → Deploy to Dev.
* release/\* → Deploy to Test.
* main → Regression + Prod.
* Tagged releases: vX.Y.Z → Prod deployment.

## 4.7 Detailed Event Hub Standards

* Namespace: eh-<domain>-<product>-<env>.
* Hub: eh-<entity>-<purpose>.
* Capture: To Landing zone in UTC partitioned folders.
* Consumer Groups: One per consuming app.
* Security: MI preferred; SAS keys short-lived.
* Schema Versioning: Envelope fields + schema\_version in payload.

## 4.8 Detailed Azure Functions Standards

* Purpose: Ingest events to Landing; validate, enrich, and route to DLQ if invalid.
* Config: Env vars + KV refs; MI for ADLS/Event Hub access.
* DLQ: ADLS path: dlq/<product>/<entity>/...
* Audit: Write job start/metrics/end to Azure Table.

## 4.9 Detailed Monitoring & SLOs

* Audit Table: job status, metrics per zone.
* App Insights: Functions telemetry.
* Log Analytics: Synapse Spark stage times, shuffle metrics.
* Dashboards: SLA compliance; DQ pass/fail rates; cost per run.

## 4.10 Detailed Security & RBAC

* Identity: MSI for Synapse, Functions, Pipelines.
* RBAC Roles: Storage Blob Data Contributor, KV Secrets User, EH Data Sender.
* Networking: Private endpoints for Prod/Test; IP restrictions in Dev.
* Secrets: No secrets in code; all via Key Vault.