**Project Book — for GenAI & ML**

Production-ready patterns for ML and LLM/GenAI on Azure Databricks

Version: 1.1 | Date: 2025-09-10

# 1. Executive Summary

This Project Book defines a production-grade template and runbook for delivering machine learning (ML) and generative AI (GenAI) applications using Databricks Asset Bundles (DAB). It separates concerns between ML (data science, modeling, evaluation) and MLOps (platform, deployment, governance, observability), and codifies the repository structure, bundle configuration (databricks.yml), CI/CD, security, and operations.

# 2. Scope & Audience

* Audience: ML Engineers, Data Scientists, Platform/MLOps Engineers, Security & Governance, Product Owners.
* Scope: Applies to both traditional ML (forecasting, classification, etc.) and GenAI (RAG, agents, copilots).
* Cloud/Platform: Azure Databricks with Unity Catalog, MLflow, Mosaic AI, Entra ID, Serverless Compute.

# 3. Architecture Overview

3.1 Four-layer Reference (from Project Charter)

* Data Foundation (GraphRAG): Delta Lake + Knowledge Graph (Spark GraphX/GraphFrames) for long-term memory.
* Tooling & Integration (Model Context Protocol, MCP): standardized, secure tool connectors.
* Intelligence & Reasoning (Multi-Agent System): OpenAI Assistants/Agents orchestration with LangChain.
* Foundational LLM (Hosted in North America): choice of Llama, GPT, Claude; governed access via Mosaic AI.

3.2 Confirmed Stack (from Project Charter)

* Cloud: Microsoft Azure; Platform: Databricks (Serverless where possible).
* Data & AI Platform: Mosaic AI; Governance: Unity Catalog; Tracking: MLflow.
* AuthN/Z: Microsoft Entra ID (SSO, OAuth/M2M).
* Agent Framework: OpenAI Assistants, LangChain; UI: Gradio (or app frameworks).

# 4. ML vs MLOps — Separation of Concerns

4.1 ML (Product/Use-case scope)

* Define business problem & KPIs; curate datasets; feature engineering; model training & evaluation.
* Prompt/agent design, RAG pipelines, tool selection (MCP), offline & online evaluation.
* Register models & app artifacts in MLflow / Unity Catalog; author notebooks & src modules; tests.

4.2 MLOps (Platform/Operations scope)

* Bundle configuration (databricks.yml), environments (dev/stage/prod) and deployment modes.
* Provision & manage compute (serverless where possible), permissions, and secret management.
* Observability (MLflow Tracing), model/app serving, cost/usage tracking via Mosaic AI Gateway.
* CI/CD automation, policy checks, data/AI governance via Unity Catalog.

# 5. Databricks Asset Bundles (DAB) Overview

5.1 Why Bundles

* Declarative, versioned deployments for jobs, pipelines, models, serving endpoints, and related resources.
* Environment-specific targets (dev/stage/prod) with validation and deployment modes.
* Repeatable automation in CI/CD with bundle validate/deploy/run commands.

5.2 Key Files & Concepts

* databricks.yml at repo root defines bundle name, targets, artifacts, variables, workspace settings.
* resources section defines jobs, pipelines, models, connections, volumes, dashboards, apps.
* targets allow environment-specific overrides (compute, permissions, schedules, variables).

# 6. Repository & Project Folder Structure

Use the following structure for any ML or GenAI project. The /bundle folder contains DAB configuration; ML code is modularized under /src; notebooks are for exploration; pipelines and tests are first-class citizens.

|  |
| --- |
| / (repo root) /  ├── app/  │ ├── app.py  │ ├── settings.py  │ └── prompts/  │ ├── system/  │ │ └── agent\_system.md  │ └── rag/  │ └── retrieval\_prompt.md  ├── src/  │ ├── config/  │ │ ├── settings.py  │ │ ├── secrets\_provider.py  │ │ ├── registry.py  │ │ └── \_\_init\_\_.py  │ ├── logging/  │ │ ├── logger.py  │ │ ├── mlflow\_logging.py  │ │ ├── tracing.py  │ │ └── \_\_init\_\_.py  │ ├── data/  │ │ ├── loaders/  │ │ ├── processing/  │ │ ├── schemas/  │ │ └── graph/  │ ├── evaluation/  │ │ ├── metrics/  │ │ ├── evaluators/  │ │ └── reports/  │ ├── ai/  │ │ ├── training/  │ │ ├── inference/  │ │ ├── embeddings/  │ │ ├── retrieval/  │ │ ├── prompts/  │ │ ├── agents/  │ │ ├── generation/  │ │ └── registry/  │ ├── pipelines/  │ │ ├── training\_pipeline.py  │ │ ├── validation\_pipeline.py  │ │ ├── testing\_pipeline.py  │ │ └── inference\_pipeline.py  │ ├── serving/  │ │ ├── endpoints.py  │ │ └── validators.py  │ └── utils/  │ ├── io.py  │ ├── decorators.py  │ └── constants.py  ├── bundle/  │ ├── databricks.yml  │ └── resources/  │ ├── jobs.yml  │ ├── pipelines.yml  │ ├── connections.yml  │ ├── serving\_endpoints.yml  │ └── permissions.yml  ├── docs/  │ ├── index.md  │ └── architecture.md  ├── .github/  │ └── workflows/  │ └── databricks\_deploy.yml  ├── requirements.txt  ├── README.md  └── LICENSE |

# 7. Governance & Security

* Unity Catalog: single source of truth for data, models, volumes; assign privileges to groups; avoid user-level grants.
* Use service principals for automation; manage identities in Entra ID; prefer OAuth tokens over PATs.
* Store secrets in Unity Catalog connections or secret scopes; never commit secrets.
* Adopt serverless compute for least-privilege data plane and simpler ops; lock down production endpoints.

# 8. Observability & Monitoring

* MLflow: experiment tracking, model registry (aliases like @champion/@challenger), lineage to runs.
* Mosaic AI Gateway: usage tracking, payload logging (Delta tables), guardrails, rate limits, fallbacks.
* MLflow Tracing: capture prompts, tool calls, latencies; wire into dashboards and alerts.
* Quality monitors: schedule evaluation jobs; track costs/latency & business KPIs.

# 9. Testing & Quality Gates

* Unit tests for src modules; integration tests for end-to-end pipelines; security tests for auth and inputs.
* Bundle validation in CI; policy checks (lint YAML, prohibited settings).
* Offline eval datasets; automated acceptance thresholds (retrieval precision/recall, hallucination rate).
* Blue/green model rollout using model registry aliases; automatic rollback on SLO breaches.

# 10. Runbooks & Operational Procedures

Common Databricks Asset Bundle commands:

databricks bundle validate -t <target>  
databricks bundle deploy -t <target>  
databricks bundle run <resource-name> -t <target>  
databricks bundle destroy -t <target>

# 11. Appendices

A. Sample RAG application structure (adapted)

See notebooks and src layout above; align with your current RAG structure to ensure parity with model tracking and retrieval components.

B. MCP Tooling Layer

* Expose enterprise tools via MCP servers; register tools in agents/tools\_mcp.py; enforce auth via Unity Catalog connections.
* Log tool I/O via MLflow Tracing for auditability and evaluation.