# Databricks Platform Architecture Design Principles

This document provides an extensive overview of the design and implementation strategies for Databricks on Azure and AWS. It emphasizes automation, resilience, cost governance (FinOps), modularity, and compliance with data security and regulatory standards.

## 1) Design for Automation & Favor Managed Services

Goals: reproducible infrastructure, consistent deployments, minimal undifferentiated operations.

### Infrastructure as Code (IaC) & Configuration

Leverage Terraform for managing Databricks and cloud infrastructure. Define workspaces, metastores, Unity Catalog permissions, clusters, pools, and SQL warehouses as code. Use Databricks Terraform provider for workspace-level resources, and azurerm/aws providers for the underlying cloud infrastructure. Implement a configuration-as-code model (YAML/JSON) that defines environment-specific parameters such as paths, tables, and SLAs.

### CI/CD

Use a Git-based CI/CD workflow to automate deployment and testing. Incorporate pre-commit hooks for linting (black, ruff, sqlfluff), unit and data testing (pytest, dbx, Great Expectations), and package validation. Deployment pipelines (GitHub Actions/Azure DevOps) should promote code through environments using Terraform apply or Databricks CLI.

### Prefer Managed Services

Favor managed services such as Serverless SQL Warehouses for BI workloads, Delta Live Tables (DLT) for pipeline orchestration and data quality, and Lakehouse Federation for cross-source queries. Use partner-managed ingestion tools (Fivetran, Azure Data Factory Managed Copy) wherever possible. For custom ingestion, utilize Databricks Auto Loader.

### Automation Guardrails

Implement policy-as-code for clusters, SQL warehouses, and workspace entitlements. Detect drift using Terraform plan checks in CI, and block deployments when discrepancies exceed thresholds. Standardize environments using cluster instance pools with pinned runtimes (e.g., Photon).

## 2) Plan for Failure (Resilience & Disaster Recovery)

Plan proactively for service continuity through redundancy, multi-AZ deployment, idempotent data processing, and automated recovery. Define reliability SLOs and DR targets per workload tier.

### Targets

Define SLOs for key metrics such as ingestion lag, job success rate, query latency, and model serving response time. Establish RTO/RPO thresholds (e.g., Tier-1 Gold RTO ≤ 60 min, RPO ≤ 15 min).

### Resilience Patterns

Use multi-AZ compute to ensure workload availability across availability zones. Configure autoscaling with a minimum node threshold to maintain baseline capacity for critical jobs. Ensure pipelines are idempotent through DLT checkpoints and deterministic Delta MERGE logic. Enable retries with exponential backoff for Glue, Workflows, or DLT Jobs. Apply VACUUM and OPTIMIZE policies that comply with retention and legal hold requirements.

### Expanded: Multi-AZ Compute Architecture

Multi-AZ (Availability Zone) compute in Databricks ensures workload resiliency by distributing cluster nodes across multiple data center zones within a single region. In AWS, this involves deploying Databricks worker nodes across multiple AZs in an EC2 Auto Scaling group tied to subnets in distinct zones. Azure Databricks achieves the same through zonal deployment and node distribution across availability zones within the Virtual Network. Databricks control plane automatically handles cluster scheduling to distribute workloads evenly, reducing the risk of localized zone failure. For mission-critical workloads, use a cluster configuration with 'availability=ZONE\_REDUNDANT' to enforce node spread. Autoscaling policies should always maintain a minimum number of nodes (>0) in different zones to avoid total workload downtime.

### Disaster Recovery (DR)

Enable storage replication (ADLS RA-GRS or S3 CRR) for critical datasets and metadata. Maintain Unity Catalog backups and replicate metastore configuration using Terraform scripts. Define failover playbooks that include re-pointing SQL warehouses, restarting DLT pipelines, and replaying Auto Loader events.

### Chaos & Validation

Conduct quarterly game days to test system resilience. Simulate cluster termination, API rate throttling, or secret revocation events to validate alerting, retry, and failover mechanisms.

## 3) Reversible Decision-Making

Design for flexibility and rollback by prioritizing open formats (Delta Lake, Parquet), decoupling compute and storage, and adopting blue/green deployment patterns. Document decisions through Architecture Decision Records (ADRs) capturing rationale and rollback procedures.

## 4) Embrace FinOps

Apply financial accountability and transparency through tagging, dashboards, and automation. Track DBU and storage usage per workspace, implement budgets and anomaly detection, and automate cost control policies. Leverage Photon runtimes, Serverless SQL for bursty workloads, and autoscaling clusters to optimize cost-performance balance.

## 5) Build a Loosely Coupled System

Create modular, event-driven data pipelines that communicate through well-defined Delta Lake interfaces. Use Unity Catalog as the data contract for schema and access governance. Adopt domain-based DLT pipelines and restrict cross-domain access to shared curated layers.

## 6) Prioritize Security (User & Regulatory Data)

Integrate SSO with SCIM provisioning for identity governance. Enforce least privilege access controls in Unity Catalog, employ row and column masking, and segregate roles by function. Ensure full encryption at rest and in transit using customer-managed keys (CMK/KMS). Route traffic through private endpoints and restrict public workspace access. Centralize audit logging in Log Analytics (Azure) or CloudWatch (AWS), and integrate with a SIEM for proactive threat detection.

## C) Cloud-Specific Choices & Templates

### Azure Databricks

Use a Hub-and-Spoke model with Secure Cluster Connectivity and VNet injection. Configure private endpoints to Databricks control plane, ADLS, and Key Vault. Store data in ADLS Gen2 with RA-GRS replication for DR. Enable monitoring via Azure Monitor and Log Analytics. Enforce policies via Azure Policy and Databricks cluster policies.

### AWS Databricks

Deploy within a VPC with PrivateLink connections to the Databricks control plane, S3, STS, Secrets Manager, and KMS. Configure IAM federation and SCIM provisioning. Enable GuardDuty, Security Hub, and CloudWatch for observability. Use Terraform to enforce guardrails through AWS Config and Service Control Policies (SCPs). Unity Catalog remains the primary governance layer, with optional integration into AWS Lake Formation for shared datasets.