Secure, Reproducible Databricks on AWS – Production Architecture Guide

This guide explains how to set up a production-ready, secure, and reproducible Databricks workspace on AWS using Terraform and modular IaC. It is tailored for deployment in Australia (Sydney or Melbourne regions) but can be adapted to other regions.

# 1) Landing zone & network (private-by-default)

Why: Isolate compute, avoid public IPs, and keep all traffic private within AWS.

Do this:  
- Create a VPC with only private subnets (2+ AZs), NAT for egress.  
- Add VPC endpoints: S3 (gateway) and interface endpoints for STS/KMS/Glue/CloudWatch Logs.  
- Turn on VPC Flow Logs to CloudWatch.

# 2) Storage & encryption

Why: Reproducible, governed storage with strong encryption and auditability.

Do this:  
- Buckets for DBFS root, lakehouse data, and logs.  
- SSE-KMS with key rotation.  
- Enforce TLS-only bucket policies.  
- S3 access logging.

# 3) Least-privilege IAM

Why: Minimize blast radius and scope credentials to purpose.

Do this:  
- Cross-account role for Databricks control plane.  
- Cluster instance profile role with only needed S3/KMS perms.

# 4) Workspace creation (secure cluster connectivity)

Why: Consistent workspace setup with audit logging.

Do this:  
- Create workspace via Databricks account-level APIs.  
- Enable Unity Catalog (metastore + assignment).  
- Configure audit log delivery to S3.

# 5) Workspace hardening & governance

Why: Enforce good defaults that protect data and control costs.

Do this:  
- Cluster policy enforcing Unity Catalog, auto-termination.  
- Groups for admins/engineering/science.  
- UC objects (catalog + bronze/silver/gold schemas).  
- External locations bound to S3 + grants.  
- Secret scopes.  
- Starter SQL Warehouse.

# 6) CI/CD & environments

Why: Consistent, reviewable changes; easy promotion.

Do this:  
- Keep env vars in environments/<env>/terraform.tfvars.  
- Use GitHub Actions OIDC for Terraform plan/apply.

# 7) Region-ready presets (Australia)

Use either Sydney (ap-southeast-2) or Melbourne (ap-southeast-4).

Environments are prefilled with buckets + workspace names.  
Control plane AWS account ID = 414351767826 (commercial).  
Update account\_id to your Databricks E2 Account ID.

# 8) Step-by-step runbook

Stage 01 – Account layer:  
$env:AWS\_PROFILE="default"  
$env:DATABRICKS\_ACCOUNT\_ID="<your-e2-account-id>"  
$env:DATABRICKS\_TOKEN="<account-level PAT>"  
$env:DATABRICKS\_ACCOUNT\_HOST="https://accounts.cloud.databricks.com"  
.\scripts\ps\01-bootstrap.ps1  
.\scripts\ps\10-apply-account.ps1 -Environment au-sydney -Action apply

Auto-fill Stage 02 variables:  
.\scripts\ps\15-fill-workspace-tfvars.ps1 -Environment au-sydney  
$env:DATABRICKS\_TOKEN="<workspace-scoped token>"

Stage 02 – Workspace layer:  
.\scripts\ps\20-apply-workspace.ps1 -Environment au-sydney -Action apply

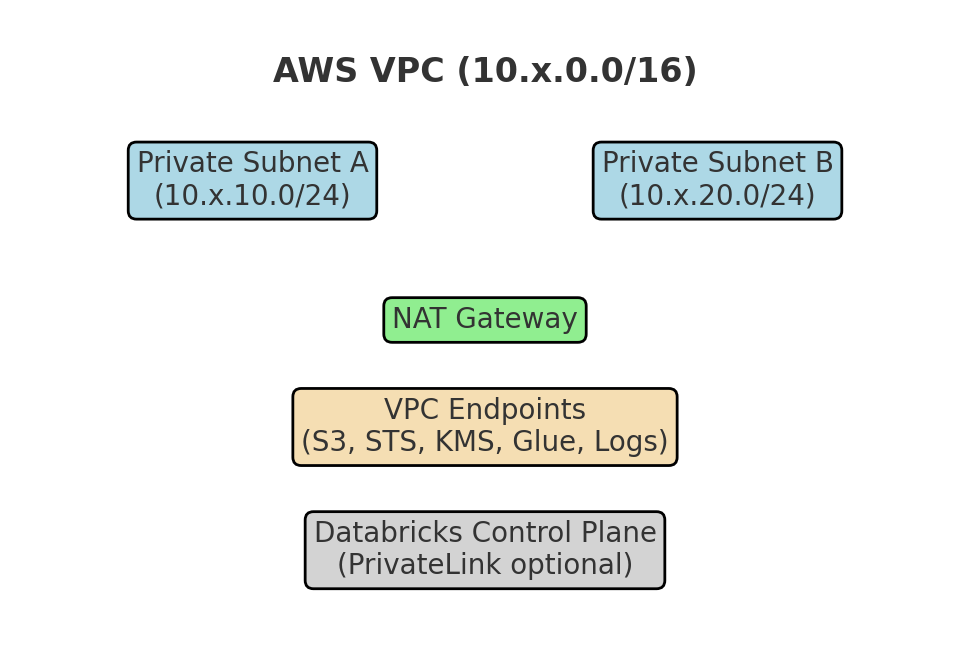
Put secrets:  
databricks secrets put --scope platform-secrets --key jdbc\_password  
databricks secrets put --scope platform-secrets --key api\_token

# 9) Operational guardrails

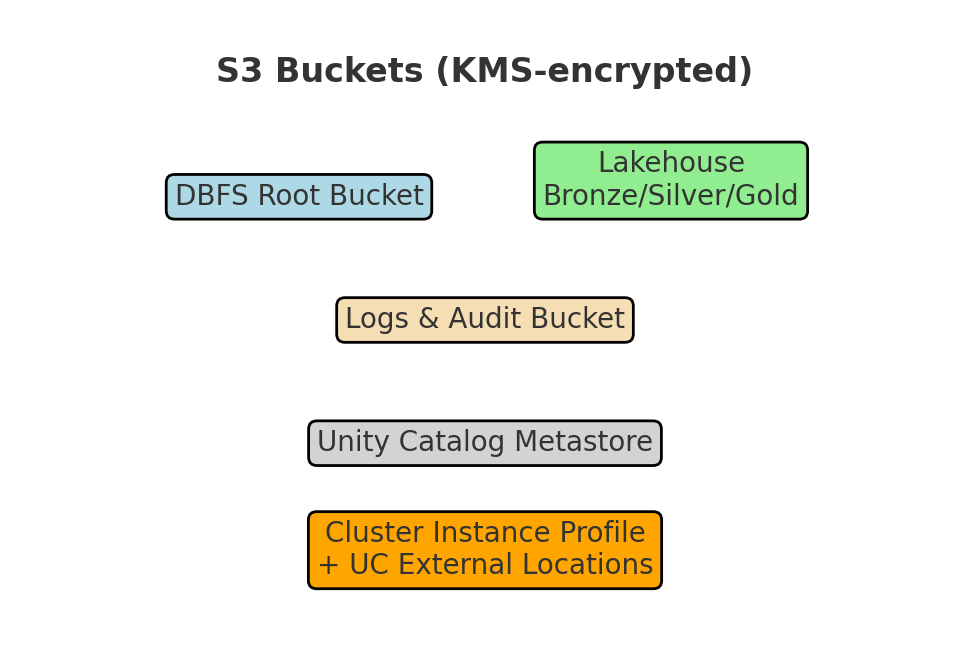
Added modules for:  
- Budgets + alerts.  
- S3 lifecycle rules.  
- PrivateLink endpoints.  
- Stricter cluster policy.  
- Default tags (env, owner, cost-center).

Architecture Diagrams

# Network Architecture



# Data & Governance Architecture



End-to-End Data Workflow

Data flows through Bronze (raw), Silver (curated), and Gold (BI-ready). Unity Catalog enforces governance and BI tools connect via SQL Warehouses.



Operational Runbook Checklist

This section provides a concise operator-focused runbook for deploying and validating a Databricks workspace on AWS in production. It is divided into three phases.

# Pre-Deployment

- Confirm AWS account is enrolled in Control Tower/landing zone.  
- Ensure AWS budgets/alerts are configured.  
- Validate that Databricks E2 account exists and you have an account-level PAT.  
- Verify required DNS, CIDR ranges, and region selection (Sydney or Melbourne).  
- Confirm Terraform backend (S3 + DynamoDB) is set up for state management.  
- Validate IAM permissions for bootstrap user/role.

# Deployment

Stage 01 – Account Layer:  
1. Export AWS and Databricks environment variables.  
2. Run bootstrap script to set up providers and state.  
3. Apply account-level Terraform (cross-account role, workspace defn, storage configs).  
  
Stage 02 – Workspace Layer:  
1. Auto-fill workspace.tfvars with generated IDs.  
2. Apply workspace Terraform (UC metastore, groups, policies, secret scopes).  
3. Create initial SQL Warehouse.  
4. Rotate workspace token and store securely.

# Post-Deployment Validation

- Validate network: confirm no public IPs; test cluster egress to S3 via VPC endpoint.  
- Check audit logs flowing into S3/CloudWatch.  
- Confirm UC metastore is assigned and permissions enforced.  
- Validate cluster policies (auto-termination, UC enforcement).  
- Run sample job writing to Bronze/Silver/Gold to confirm pipeline works.  
- Test BI tool (e.g., Power BI, Tableau, Databricks SQL) connection to Gold tables.  
- Check budgets/alerts fire correctly when thresholds are met.  
- Document workspace URL, admin group members, and escalation contacts.

Terraform: Providers & Variables (Account Layer)

Defines AWS and Databricks account-level providers (for workspace creation), plus core variables you'll reuse across environments.

terraform {  
 required\_version = ">= 1.5.0"  
 required\_providers {  
 aws = { source = "hashicorp/aws", version = "~> 5.52" }  
 databricks = { source = "databricks/databricks", version = "~> 1.51.0" }  
 }  
}  
  
provider "aws" {  
 region = var.aws\_region  
 profile = var.aws\_profile  
 default\_tags { tags = var.default\_tags }  
}  
  
provider "databricks" {  
 alias = "account"  
 host = coalesce(var.databricks\_account\_host, "https://accounts.cloud.databricks.com")  
 account\_id = var.account\_id  
}  
  
variable "aws\_profile" { type = string }  
variable "aws\_region" { type = string }  
variable "account\_id" { type = string } # Databricks E2 Account ID  
variable "databricks\_account\_host" { type = string, default = null }  
variable "default\_tags" { type = map(string), default = {} }

Terraform: Networking (VPC, Subnets, NAT, Endpoints, Flow Logs)

Private subnets only with NAT for egress; gateway endpoint for S3, optional interface endpoints, and VPC Flow Logs to CloudWatch.

resource "aws\_vpc" "this" {  
 cidr\_block = var.vpc\_cidr  
 enable\_dns\_support = true  
 enable\_dns\_hostnames = true  
 tags = { Name = "${var.workspace\_name}-vpc" }  
}  
  
resource "aws\_subnet" "private" {  
 for\_each = { for idx, cidr in var.private\_subnet\_cidrs : idx => { cidr = cidr, az = var.azs[idx] } }  
 vpc\_id = aws\_vpc.this.id  
 cidr\_block = each.value.cidr  
 availability\_zone = each.value.az  
 map\_public\_ip\_on\_launch = false  
 tags = { Name = "${var.workspace\_name}-private-${each.value.az}" }  
}  
  
resource "aws\_eip" "nat" { vpc = true }  
resource "aws\_nat\_gateway" "nat" {  
 allocation\_id = aws\_eip.nat.id  
 subnet\_id = values(aws\_subnet.private)[0].id  
}  
  
resource "aws\_route\_table" "private" { vpc\_id = aws\_vpc.this.id }  
resource "aws\_route" "default" {  
 route\_table\_id = aws\_route\_table.private.id  
 destination\_cidr\_block = "0.0.0.0/0"  
 nat\_gateway\_id = aws\_nat\_gateway.nat.id  
}  
resource "aws\_route\_table\_association" "a" {  
 for\_each = aws\_subnet.private  
 subnet\_id = each.value.id  
 route\_table\_id = aws\_route\_table.private.id  
}  
  
# Endpoints  
resource "aws\_vpc\_endpoint" "s3" {  
 vpc\_id = aws\_vpc.this.id  
 service\_name = "com.amazonaws.${var.aws\_region}.s3"  
 vpc\_endpoint\_type = "Gateway"  
 route\_table\_ids = [aws\_route\_table.private.id]  
}  
  
resource "aws\_security\_group" "endpoints" {  
 name = "${var.workspace\_name}-endpoints-sg"  
 vpc\_id = aws\_vpc.this.id  
 egress { from\_port = 0, to\_port = 0, protocol = "-1", cidr\_blocks = ["0.0.0.0/0"] }  
}  
  
resource "aws\_vpc\_endpoint" "interfaces" {  
 for\_each = toset(["kms", "sts", "glue", "logs"])  
 vpc\_id = aws\_vpc.this.id  
 service\_name = "com.amazonaws.${var.aws\_region}.${each.key}"  
 vpc\_endpoint\_type = "Interface"  
 security\_group\_ids = [aws\_security\_group.endpoints.id]  
 subnet\_ids = [for s in aws\_subnet.private : s.id]  
 private\_dns\_enabled = true  
}  
  
# Flow Logs  
resource "aws\_cloudwatch\_log\_group" "vpc\_flow" {  
 name = "/databricks/${var.workspace\_name}/vpc-flow"  
 retention\_in\_days = 30  
}  
resource "aws\_iam\_role" "vpc\_flow" {  
 name = "${var.workspace\_name}-vpc-flow-logs"  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow", Principal = { Service = "vpc-flow-logs.amazonaws.com" }, Action = "sts:AssumeRole"  
 }]  
 })  
}  
resource "aws\_iam\_role\_policy" "vpc\_flow" {  
 role = aws\_iam\_role.vpc\_flow.id  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow",  
 Action = ["logs:CreateLogStream","logs:PutLogEvents","logs:DescribeLogGroups","logs:DescribeLogStreams"],  
 Resource = aws\_cloudwatch\_log\_group.vpc\_flow.arn  
 }]  
 })  
}  
resource "aws\_flow\_log" "this" {  
 vpc\_id = aws\_vpc.this.id  
 log\_destination\_type = "cloud-watch-logs"  
 log\_group\_name = aws\_cloudwatch\_log\_group.vpc\_flow.name  
 iam\_role\_arn = aws\_iam\_role.vpc\_flow.arn  
 traffic\_type = "ALL"  
}

Terraform: KMS + S3 (Root/Data/Logs) with TLS-only and Access Logging

Buckets for DBFS root, lakehouse data, and audit logs. All encrypted with KMS, TLS-only policies, and access logs sent to the logs bucket.

resource "aws\_kms\_key" "s3" {  
 description = "S3 encryption key"  
 enable\_key\_rotation = true  
 deletion\_window\_in\_days = 10  
}  
resource "aws\_kms\_alias" "s3" { name = "alias/${var.workspace\_name}-s3"; target\_key\_id = aws\_kms\_key.s3.key\_id }  
  
resource "aws\_s3\_bucket" "root" { bucket = var.root\_bucket\_name }  
resource "aws\_s3\_bucket\_versioning" "root" { bucket = aws\_s3\_bucket.root.id  
 versioning\_configuration { status = "Enabled" } }  
resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "root" {  
 bucket = aws\_s3\_bucket.root.id  
 rule { apply\_server\_side\_encryption\_by\_default { sse\_algorithm = "aws:kms" kms\_master\_key\_id = aws\_kms\_key.s3.arn } }  
}  
resource "aws\_s3\_bucket\_public\_access\_block" "root" {  
 bucket = aws\_s3\_bucket.root.id  
 block\_public\_acls = true; block\_public\_policy = true; ignore\_public\_acls = true; restrict\_public\_buckets = true  
}  
  
resource "aws\_s3\_bucket" "data" { bucket = var.data\_bucket\_name }  
resource "aws\_s3\_bucket\_versioning" "data" { bucket = aws\_s3\_bucket.data.id  
 versioning\_configuration { status = "Enabled" } }  
resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "data" {  
 bucket = aws\_s3\_bucket.data.id  
 rule { apply\_server\_side\_encryption\_by\_default { sse\_algorithm = "aws:kms" kms\_master\_key\_id = aws\_kms\_key.s3.arn } }  
}  
resource "aws\_s3\_bucket\_public\_access\_block" "data" {  
 bucket = aws\_s3\_bucket.data.id  
 block\_public\_acls = true; block\_public\_policy = true; ignore\_public\_acls = true; restrict\_public\_buckets = true  
}  
  
resource "aws\_s3\_bucket" "logs" { bucket = var.logs\_bucket\_name }  
resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "logs" {  
 bucket = aws\_s3\_bucket.logs.id  
 rule { apply\_server\_side\_encryption\_by\_default { sse\_algorithm = "aws:kms" kms\_master\_key\_id = aws\_kms\_key.s3.arn } }  
}  
resource "aws\_s3\_bucket\_public\_access\_block" "logs" {  
 bucket = aws\_s3\_bucket.logs.id  
 block\_public\_acls = true; block\_public\_policy = true; ignore\_public\_acls = true; restrict\_public\_buckets = true  
}  
  
# TLS-only policy  
data "aws\_iam\_policy\_document" "tls\_only" {  
 statement {  
 sid = "DenyInsecureTransport"  
 effect = "Deny"  
 actions = ["s3:\*"]  
 principals { type = "\*", identifiers = ["\*"] }  
 resources = [  
 aws\_s3\_bucket.root.arn, "${aws\_s3\_bucket.root.arn}/\*",  
 aws\_s3\_bucket.data.arn, "${aws\_s3\_bucket.data.arn}/\*",  
 aws\_s3\_bucket.logs.arn, "${aws\_s3\_bucket.logs.arn}/\*",  
 ]  
 condition { test = "Bool", variable = "aws:SecureTransport", values = ["false"] }  
 }  
}  
resource "aws\_s3\_bucket\_policy" "tls\_root" { bucket = aws\_s3\_bucket.root.id policy = data.aws\_iam\_policy\_document.tls\_only.json }  
resource "aws\_s3\_bucket\_policy" "tls\_data" { bucket = aws\_s3\_bucket.data.id policy = data.aws\_iam\_policy\_document.tls\_only.json }  
resource "aws\_s3\_bucket\_policy" "tls\_logs" { bucket = aws\_s3\_bucket.logs.id policy = data.aws\_iam\_policy\_document.tls\_only.json }  
  
# Access logging to logs bucket  
resource "aws\_s3\_bucket\_logging" "root\_to\_logs" {  
 bucket = aws\_s3\_bucket.root.id  
 target\_bucket = aws\_s3\_bucket.logs.id  
 target\_prefix = "s3-access/root/"  
}  
resource "aws\_s3\_bucket\_logging" "data\_to\_logs" {  
 bucket = aws\_s3\_bucket.data.id  
 target\_bucket = aws\_s3\_bucket.logs.id  
 target\_prefix = "s3-access/data/"  
}

Terraform: IAM (Databricks Cross-Account + Cluster Instance Profile)

Cross-account role for Databricks control plane and an EC2 instance profile role for workspace clusters with least-privilege access to your buckets and KMS key.

# Cross-account role (databricks\_aws\_account\_id must match your region)  
resource "aws\_iam\_role" "databricks\_cross" {  
 name = "${var.workspace\_name}-dbcx"  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow",  
 Principal = { AWS = "arn:aws:iam::${var.databricks\_aws\_account\_id}:root" },  
 Action = "sts:AssumeRole",  
 Condition = { StringEquals = { "sts:ExternalId" = var.account\_id } }  
 }]  
 })  
}  
resource "aws\_iam\_policy" "dbcx\_permissions" {  
 name = "${var.workspace\_name}-dbcx-permissions"  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [  
 { Effect = "Allow", Action = ["s3:\*"], Resource = [  
 aws\_s3\_bucket.root.arn, "${aws\_s3\_bucket.root.arn}/\*"  
 ]},  
 { Effect = "Allow", Action = ["logs:\*"], Resource = "\*" },  
 { Effect = "Allow", Action = ["kms:\*","sts:AssumeRole"], Resource = "\*" }  
 ]  
 })  
}  
resource "aws\_iam\_role\_policy\_attachment" "dbcx\_attach" {  
 role = aws\_iam\_role.databricks\_cross.name  
 policy\_arn = aws\_iam\_policy.dbcx\_permissions.arn  
}  
  
# Instance profile for clusters  
resource "aws\_iam\_role" "cluster\_data" {  
 name = "${var.workspace\_name}-cluster-data"  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow", Principal = { Service = "ec2.amazonaws.com" }, Action = "sts:AssumeRole"  
 }]  
 })  
}  
resource "aws\_iam\_policy" "cluster\_data\_access" {  
 name = "${var.workspace\_name}-cluster-data-access"  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [  
 { Effect = "Allow", Action = ["s3:ListBucket"], Resource = [  
 aws\_s3\_bucket.data.arn, aws\_s3\_bucket.root.arn  
 ]},  
 { Effect = "Allow", Action = ["s3:GetObject","s3:PutObject","s3:DeleteObject","s3:ListBucketMultipartUploads","s3:AbortMultipartUpload"],  
 Resource = ["${aws\_s3\_bucket.data.arn}/\*","${aws\_s3\_bucket.root.arn}/\*"] },  
 { Effect = "Allow", Action = ["kms:Encrypt","kms:Decrypt","kms:ReEncrypt\*","kms:GenerateDataKey\*","kms:DescribeKey"],  
 Resource = [aws\_kms\_key.s3.arn] }  
 ]  
 })  
}  
resource "aws\_iam\_role\_policy\_attachment" "cluster\_data\_attach" {  
 role = aws\_iam\_role.cluster\_data.name  
 policy\_arn = aws\_iam\_policy.cluster\_data\_access.arn  
}  
resource "aws\_iam\_instance\_profile" "cluster\_profile" {  
 name = "${var.workspace\_name}-cluster-profile"  
 role = aws\_iam\_role.cluster\_data.name  
}

Terraform: Databricks Workspace (Account-Level)

Create the workspace via the account-level Databricks provider, referencing credentials, storage, and network.

resource "databricks\_mws\_credentials" "creds" {  
 provider = databricks.account  
 account\_id = var.account\_id  
 credentials\_name = "${var.workspace\_name}-creds"  
 role\_arn = aws\_iam\_role.databricks\_cross.arn  
}  
  
resource "databricks\_mws\_storage\_configurations" "storage" {  
 provider = databricks.account  
 account\_id = var.account\_id  
 storage\_configuration\_name = "${var.workspace\_name}-storage"  
 bucket\_name = aws\_s3\_bucket.root.bucket  
}  
  
resource "databricks\_mws\_networks" "network" {  
 provider = databricks.account  
 account\_id = var.account\_id  
 network\_name = "${var.workspace\_name}-network"  
 vpc\_id = aws\_vpc.this.id  
 subnet\_ids = [for s in aws\_subnet.private : s.id]  
 security\_group\_ids = [aws\_security\_group.endpoints.id]  
}  
  
resource "databricks\_mws\_workspaces" "ws" {  
 provider = databricks.account  
 account\_id = var.account\_id  
 workspace\_name = var.workspace\_name  
 deployment\_name = var.workspace\_name  
 aws\_region = var.aws\_region  
 credentials\_id = databricks\_mws\_credentials.creds.credentials\_id  
 storage\_configuration\_id = databricks\_mws\_storage\_configurations.storage.storage\_configuration\_id  
 network\_id = databricks\_mws\_networks.network.network\_id  
}

Terraform: Unity Catalog Metastore + Assignment

Create a metastore with a storage root in your logs bucket and assign it to the workspace.

resource "databricks\_metastore" "uc" {  
 provider = databricks.account  
 name = "${var.workspace\_name}-metastore"  
 storage\_root = "s3://${aws\_s3\_bucket.logs.bucket}/uc-metastore"  
 force\_destroy = false  
}  
  
resource "databricks\_metastore\_assignment" "uc\_ws" {  
 provider = databricks.account  
 metastore\_id = databricks\_metastore.uc.id  
 workspace\_id = databricks\_mws\_workspaces.ws.workspace\_id  
 default\_catalog\_name = "main"  
}

Terraform: Databricks Audit Log Delivery

Account-level log delivery configuration that sends JSON audit logs to your logs bucket.

resource "databricks\_mws\_log\_delivery" "audit\_logs" {  
 provider = databricks.account  
 account\_id = var.account\_id  
 config\_name = "${var.workspace\_name}-audit-logs"  
 log\_type = "AUDIT\_LOGS"  
 output\_format = "JSON"  
 delivery\_path\_prefix = "audit-logs"  
 workspace\_ids\_filter = [databricks\_mws\_workspaces.ws.workspace\_id]  
 s3\_bucket = aws\_s3\_bucket.logs.bucket  
 s3\_region = var.aws\_region  
 s3\_prefix = "databricks"  
}

Terraform: Workspace Hardening (Groups, UC Objects, External Locations, Grants)

Define workspace groups, a team catalog with bronze/silver/gold schemas, external locations to S3, and grants that give least-privilege access.

provider "databricks" {} # workspace-level (uses DATABRICKS\_HOST + DATABRICKS\_TOKEN)  
  
resource "databricks\_group" "admins" { display\_name = "ws-admins" }  
resource "databricks\_group" "data\_eng" { display\_name = "data-eng" }  
resource "databricks\_group" "data\_science" { display\_name = "data-science" }  
  
resource "databricks\_cluster\_policy" "secure\_default" {  
 name = "secure-cluster-policy"  
 definition = file("${path.module}/policies/secure.json")  
}  
  
resource "databricks\_catalog" "team" { name = "lakehouse" }  
resource "databricks\_schema" "bronze" { catalog\_name = databricks\_catalog.team.name, name = "bronze" }  
resource "databricks\_schema" "silver" { catalog\_name = databricks\_catalog.team.name, name = "silver" }  
resource "databricks\_schema" "gold" { catalog\_name = databricks\_catalog.team.name, name = "gold" }  
  
resource "databricks\_storage\_credential" "lake\_cred" {  
 name = "lake-credential"  
 aws\_iam\_role { role\_arn = aws\_iam\_instance\_profile.cluster\_profile.arn }  
}  
  
resource "databricks\_external\_location" "bronze" {  
 name = "ext-bronze"  
 url = "s3://${var.data\_bucket\_name}/bronze"  
 credential\_name = databricks\_storage\_credential.lake\_cred.name  
}  
resource "databricks\_external\_location" "silver" {  
 name = "ext-silver"  
 url = "s3://${var.data\_bucket\_name}/silver"  
 credential\_name = databricks\_storage\_credential.lake\_cred.name  
}  
resource "databricks\_external\_location" "gold" {  
 name = "ext-gold"  
 url = "s3://${var.data\_bucket\_name}/gold"  
 credential\_name = databricks\_storage\_credential.lake\_cred.name  
}  
  
resource "databricks\_grants" "catalog" {  
 catalog = databricks\_catalog.team.name  
 grant { principal = databricks\_group.data\_eng.display\_name privileges = ["USE\_CATALOG"] }  
 grant { principal = databricks\_group.data\_science.display\_name privileges = ["USE\_CATALOG"] }  
 grant { principal = databricks\_group.admins.display\_name privileges = ["ALL\_PRIVILEGES"] }  
}

Terraform: SQL Warehouse + Secret Scope

Provision a small, UC-enabled SQL Warehouse for BI and create a secret scope for credentials (values added via CLI).

resource "databricks\_sql\_warehouse" "standard" {  
 name = "analytics-standard"  
 cluster\_size = "2X-Small"  
 max\_num\_clusters = 1  
 auto\_stop\_mins = 15  
 spot\_instance\_policy = "COST\_OPTIMIZED"  
 enable\_photon = true  
 warehouse\_type = "PRO"  
}  
  
resource "databricks\_secret\_scope" "platform" { name = "platform-secrets" }  
# Put secrets at runtime using CLI:  
# databricks secrets put --scope platform-secrets --key jdbc\_password  
# databricks secrets put --scope platform-secrets --key api\_token

Terraform Modules: Budgets & Alerts, S3 Lifecycle, PrivateLink

Plug-and-play guardrail modules. Wire them in your account layer depending on your needs.

module "budgets" {  
 source = "../modules/budgets-alerts"  
 name = "${var.workspace\_name}-${var.aws\_region}-monthly"  
 limit\_amount = 500  
 emails = ["alerts@example.com"]  
}  
  
module "logs\_bucket\_lifecycle" {  
 source = "../modules/s3-lifecycle"  
 bucket\_id = aws\_s3\_bucket.logs.id  
 current\_transition\_days = 30  
 noncurrent\_transition\_days = 60  
 expiration\_days = 365  
}  
  
module "privatelink" {  
 source = "../modules/privatelink-generic"  
 vpc\_id = aws\_vpc.this.id  
 subnet\_ids = [for s in aws\_subnet.private : s.id]  
 security\_group\_id = aws\_security\_group.endpoints.id  
 service\_names = [] # paste region-specific service names for Databricks PrivateLink  
}