# Terraform Modules Design — AWS Data Platform

Created: September 15, 2025

## Executive summary

This document defines a set of reusable Terraform modules for automating the infrastructure and data platform components listed: Redshift, S3, AWS DataZone, SageMaker, Athena, AFT (account provisioning patterns), Guardrails, and CloudWatch. Each module is intended to be:

- Single-purpose: do one job well (create a bucket, create a Redshift cluster/namespace, create a SageMaker endpoint).  
- Composable: modules can be composed by higher-level stacks (environment, team, data-pipeline).  
- Secure by-default: sensible defaults for encryption, logging, and least-privilege IAM.  
- Well-documented: clear variables, outputs and examples.

## High-level architecture & module strategy

Low-level modules ("primitive") live in modules/ and implement a single AWS resource or small group of tightly related resources. Higher-level modules ("platform" or "service stack") live in modules/platform/<service> and compose primitives into opinionated patterns (e.g., Redshift with subnet groups, security groups, parameter groups, IAM roles, and monitoring). Design principles include inputs validation, sensitive variables handling, and composability.

## Folder layout & module registry

repo-root/  
├─ modules/  
│ ├─ s3/  
│ ├─ redshift/  
│ ├─ datazone/  
│ ├─ sagemaker/  
│ ├─ athena/  
│ ├─ aft/  
│ ├─ guardrail/  
│ └─ cloudwatch/  
├─ envs/  
├─ ci/ (pipelines, linters)  
└─ docs/

## General module conventions (inputs, outputs, tagging, IAM)

Inputs: small, explicit. Outputs: resource IDs/ARNS, connection strings, monitoring log group names. Tagging: every resource must include tags: Environment, Project, Owner, CostCenter. Naming: ${var.prefix}-${var.environment}-${var.name}. IAM: narrow roles; allow existing\_role\_arn.

## Module designs - Redshift

Purpose: Provision Redshift clusters or Redshift Serverless namespaces and workgroups. Responsibilities: create cluster or serverless, subnet group, security group, parameter groups, IAM role for COPY/UNLOAD, automated snapshots, CloudWatch exports and alarms. Inputs: name, environment, deployment\_mode, node\_type, number\_of\_nodes, vpc\_id, subnet\_ids, kms\_key\_arn, snapshot\_retention\_days, s3\_access\_role\_arn. Notes: use private subnets, publicly\_accessible = false.

## Module designs - S3

Purpose: Create S3 buckets with secure defaults and lifecycle rules. Responsibilities: create bucket with versioning, encryption (SSE-KMS), block public access, lifecycle rules, optional CRR, access points. Inputs: name, environment, kms\_key\_arn, versioning\_enabled, lifecycle\_rules, replication. Outputs: bucket\_arn, bucket\_id, bucket\_domain\_name. Lifecycle default: 30 days -> IA, 90 days -> Glacier, 365 days -> expire.

## Module designs - AWS DataZone

Purpose: Automate DataZone domain, environment, and user-role bindings. Responsibilities: create DataZone domain/environment/project skeleton (if provider supports), configure data product catalogs and default permissions, set up integration with Lake Formation and Glue catalogs. Inputs: domain\_name, environment\_name, catalog\_integration, admins. Notes: DataZone resources and APIs evolve; provide toggle to skip resource creation.

## Module designs - SageMaker

Purpose: Provision SageMaker resources: execution role, model, endpoint configuration, endpoint, studio domains/notebook instances. Responsibilities: create execution role with scoped access, model definition, endpoint config and endpoint, optionally Studio domain and user profiles. Inputs: model\_name, instance\_type, initial\_instance\_count, s3\_model\_artifact, ecr\_image\_uri, kms\_key\_arn, vpc\_config, enable\_autoscaling. Notes: prefer private VPC endpoints for sensitive data.

## Module designs - Athena

Purpose: Create Athena workgroups, named queries, and configure result bucket. Responsibilities: create workgroup with encryption and results location, named queries and prepared statements, grant Glue Catalog permissions. Inputs: workgroup\_name, result\_bucket, enforce\_workgroup\_settings. Notes: separate result buckets per environment.

## Module designs - AFT

Purpose: Support automation with AWS Account Factory for Terraform (AFT) or chosen account vending system. Responsibilities: provision AFT-related resources or call AFT modules to create accounts, manage templates for account baselines. Inputs: account\_config, organizational\_units, control\_tower\_parameters. Notes: keep AFT operations in separate pipeline.

## Module designs - Guardrail

Purpose: Implement guardrails: AWS Config rules, SCPs, IAM Permission Boundaries. Responsibilities: create/update AWS Config rules, aggregators, remediation actions, create SCPs for accounts/OUs. Inputs: scps, config\_rules, remediation\_actions. Notes: test SCPs in preprod.

## Module designs - CloudWatch

Purpose: Create log groups, metric filters, dashboards and alarms. Responsibilities: create log groups with retention and KMS encryption, metric filters and alarms for critical metrics, create dashboards per environment. Inputs: dashboards, log\_groups, metric\_filters. Notes: integrate with SNS and incident tools.

## Cross-cutting concerns

Networking & VPC design: modules should accept vpc\_id and subnet\_ids. IAM: create\_role toggles and existing\_role\_arn. Logging & monitoring: enable resource-level logging and centralize logs. Encryption & secrets: use KMS CMKs and Secrets Manager. Cost controls & tagging: integrate with Budgets and tag resources.

## Testing, validation & CI/CD (Azure DevOps example)

Automated checks: terraform fmt, tflint, terraform validate, terraform plan, use kitchen-terraform or terratest, security checks with checkov and tfsec. Example pipeline snippet provided in the original document.

## Release & versioning policy

Semantic versioning MAJOR.MINOR.PATCH, breaking changes bump MAJOR, maintain CHANGELOG.md and publish to private registry.

## Appendix: example module interface & usage snippets

Includes example s3 module variables and example redshift module usage snippet.

## Next steps / Recommendations

1. Review these module interfaces and adapt naming/tags to your organization standards.  
2. Implement one module at a time and publish to the private registry.  
3. Add automated tests and deploy first to dev environment, then iterate.