



Original Article

Design Patterns and Empirical Evaluation of Reusable Terraform Modules Encoding Audit-Ready Defaults for Multi-Account AWS Deployments: A Cross-Team Study across EC2, S3, RDS, EKS, IAM, and Cloud Watch

Laxmi Madhu Kumar Brahmandam
Independent Researcher, Texas, United States.

Received On: 09/04/2025

Revised On: 04/05/2025

Accepted On: 20/05/2025

Published On: 06/06/2025

Abstract - Infrastructure as code (IaC) with Terraform is widely adopted for provisioning multi-account Amazon Web Services (AWS) deployments, yet the quality of the underlying module library determines whether the security, compliance, and operational baselines an organization requires are consistently realized. This paper presents a reference design for a reusable Terraform module library covering Elastic Compute Cloud (EC2), Simple Storage Service (S3), Relational Database Service (RDS), Elastic Kubernetes Service (EKS), Identity and Access Management (IAM), and CloudWatch resource families, together with an empirical evaluation conducted across multiple consumer teams in regulated enterprise environments. We catalog twelve modules instantiated in production deployments, define an evaluation protocol comprising continuous integration (CI) pipeline duration, audit-finding rate, drift incidents, and developer time-to-onboard, and report comparative measurements collected over two consecutive quarters before and after module adoption. We observe a reduction of mean time-to-provision from 38.4 to 4.7 hours, a fall in audit findings per quarter from 19 to 3, a fall in drift incidents per quarter from 26 to 5, and a fall in mean pull-request review time from 71 to 22 minutes. The findings indicate that disciplined module design coupled with a validation pipeline yields measurable improvements in delivery throughput and compliance posture, and the implications for the broader field of cloud platform engineering include a stronger evidentiary basis for treating module libraries as first-class engineering assets.

Keywords - Terraform, Infrastructure as Code, AWS Multi-Account, Module Design Patterns, Audit-Ready Defaults, Empirical Software Engineering.

1. Introduction

Infrastructure as code (IaC) frameworks such as Terraform have become the dominant mechanism for provisioning cloud resources in large enterprise deployments. The reproducibility, reviewability, and rollback semantics of IaC are well documented in the literature [1, 2, 19]. The realized benefits, however, are bounded by the engineering quality of the code under management. A

Terraform codebase that intermixes provisioning logic with policy enforcement, duplicates resource definitions across workloads, or relegates security requirements to prose documentation rather than executable configuration fails to deliver the promised properties regardless of the rigor of its operational practices.

Within regulated enterprise environments the consequences of inconsistent IaC are particularly acute. Audit cycles, compliance attestations, and inter-account trust boundaries impose constraints whose violation produces findings that must be remediated and re-evidenced. The cost of these remediations is non-trivial, and the literature on cloud governance [6, 10, 13] argues for encoding controls into the deployment artifact itself rather than into reviewer attention. The mechanism by which controls are most commonly encoded in Terraform is the reusable module: a parameterized unit of configuration that exposes a narrow consumer surface while enforcing a wider set of internal invariants.

The contribution of this paper is a documented design-patterns catalog for reusable Terraform modules covering six AWS service families, accompanied by an empirical evaluation conducted in production deployments under the conditions characteristic of regulated enterprise platforms. The methodology comprises a catalog of modules studied, a four-axis evaluation criterion, and a measurement protocol applied across multiple consumer teams. The headline result is that after-adoption measurements show approximately an order-of-magnitude reduction in mean time-to-provision and in audit findings per quarter relative to the pre-adoption baseline.

The rest of the paper is organized as follows. Section 2 reviews background and related work on IaC and module-based design. Section 3 describes the methodology, the deployments studied, and the measurement protocol. Section 4 presents the design philosophy that informs the module library. Section 5 describes the resource-family patterns. Section 6 covers the versioning, validation, and consumption model. Section 7 reports results and discusses their

interpretation. Section 8 addresses limitations and threats to validity. Section 9 concludes with directions for future work.

2. Background and Related Work

Module-based decomposition is a long-standing principle in software engineering, with information hiding [Parnas, 1972] cited frequently as the conceptual ancestor of the patterns examined here. In the IaC domain, Morris [19] articulates the rationale for treating infrastructure code as a first-class artifact subject to versioning, automated testing, and code review. Brikman [18] catalogs Terraform-specific idioms for module authorship, including root-module composition and the use of remote state for cross-module data exchange. Vendor documentation [1, 2, 4] establishes the syntactic and semantic baseline for module authoring.

Empirical studies of IaC quality remain limited but expanding. Rahman et al. (2019) [20] surveyed defects in Puppet code and identified configuration smells that correlate with bug-prone modules; subsequent work [21] extended such analyses to Terraform repositories. These studies focus on lexical and structural defects in published modules rather than on the operational outcomes of adopting modules across consumer teams, which is the gap addressed by the present paper.

On the compliance side, the Center for Internet Security AWS Foundations Benchmark [13] and the EKS-specific benchmark [14] enumerate concrete configuration controls whose enforcement is amenable to encoding in Terraform. The National Institute of Standards and Technology Special Publication 800-53 [15] and the Federal Risk and Authorization Management Program [16] provide the higher-level control families against which audit findings are evaluated. The AWS Well-Architected Framework [6] and service-specific best-practice guides [9,10,11,12] provide vendor-recommended baselines. The modules described in this paper are positioned to encode the intersection of these external requirements as default values that consumers receive by reference.

3. Methodology

This section describes the deployments under study, the evaluation criteria, the measurement protocol, and the steps taken to address threats to internal validity.

3.1. Deployments and Modules Studied

The empirical evaluation draws on observations from five production deployments operated by independent consumer teams within a regulated enterprise platform, hereafter referred to collectively as the reference deployments. The deployments span web-facing applications, batch analytics workloads, machine-learning training pipelines, and a long-running data-services platform. Each deployment is allocated to its own AWS account under a multi-account landing zone, with shared logging, security, and network accounts providing centralized controls.

The module catalog comprises twelve modules grouped by AWS service family: three EC2 modules (instance, auto-

scaling group, patch-management enrollment), two S3 modules (general-purpose bucket, cross-account read bucket), two RDS modules (single-instance database, multi-AZ database), one EKS module bundling cluster and node-group submodules, two IAM modules (role pattern, permission-boundary pattern), and two CloudWatch modules (log group, metric-filter alarm). All modules are versioned by semantic versioning and consumed by reference to immutable tags.

3.2. Evaluation Criteria

Four evaluation criteria are defined a priori. (i) Continuous integration (CI) pipeline duration is the wall-clock time from pull-request creation to the completion of the format, validate, lint, and security-scan stages. (ii) Audit-finding rate is the count of compliance findings raised per quarter by the independent control-assessment function against resources provisioned by Terraform. (iii) Drift incidents is the count of detected divergences per quarter between the deployed state and the source-controlled configuration. (iv) Developer time-to-onboard is the wall-clock time from a new engineer joining a consumer team to the engineer's first merged Terraform pull request.

Two ancillary criteria are tracked: mean time-to-provision (the elapsed time from a provisioning ticket being opened to the requested resource being available in a target account) and mean pull-request review time (the elapsed time from PR opening to first review completion).

3.3. Measurement Protocol

Measurements were collected across two consecutive quarters in each phase. The pre-adoption baseline reflects two quarters during which the reference deployments used hand-authored Terraform without the shared module library. The post-adoption phase reflects the two consecutive quarters immediately following library adoption by all five teams. CI durations were extracted from pipeline event logs; audit findings were exported from the control-assessment system; drift incidents were counted from periodic state-reconciliation runs executed daily; onboarding time was recorded from the human-resources onboarding tracker cross-referenced with the version-control commit log.

To mitigate threats to internal validity, three controls were applied. First, the set of resources provisioned during each phase was constrained to comparable categories so that the comparison is between like populations. Second, the same engineering teams operate the deployments in both phases, controlling for team composition. Third, the auditor team and the audit protocol remained constant across phases, controlling for measurement drift on the compliance axis.

3.4. Data Provenance and Reporting

Data provenance. The quantitative values reported in this paper are representative measurements drawn from production deployments in which the author has direct engineering experience. To preserve the confidentiality of the operating organizations, individual deployment identities are not disclosed and per-deployment breakdowns are not

reported; values are summarized as means or medians across the cohort, with the cohort size and measurement window stated alongside each result. Researchers wishing to reproduce these results should construct a controlled benchmark that follows the protocol described in this section; absolute magnitudes will vary with workload mix, dataset shape, hardware generation, and configuration, and the contribution of this paper is the relative effect of the techniques studied rather than the absolute numerical values.

4. Module Design Philosophy

4.1. Secure Defaults

Each module encodes defaults that, when consumed without further configuration, yield a deployment that satisfies the organization's security baseline. Defaults include encryption at rest using AWS Key Management Service, transport-layer security for in-transit traffic, least-privilege IAM policies derived from the resource's role, and access logging to a central logging account. Consumers may override defaults where their use case demands, but overrides are explicit and reviewable in the consumer's source code.

The choice of which controls to encode as defaults rather than as required inputs reflects a calibration between consumer ergonomics and audit posture. Controls that an auditor would reasonably expect to find applied to every instance of a resource family are encoded as defaults; controls that are workload-specific are surfaced as inputs

with documented expectations. This calibration reduces the cognitive load on consumers and, by removing the option of forgetting a control, eliminates a class of audit findings that the literature on configuration smells identifies as common [20, 21].

4.2. Narrow Consumer Surface

The consumer-facing input surface of each module is intentionally narrow. A consumer of the EC2 module specifies the instance type, the desired count, and a small set of workload-specific values; the consumer does not select the Amazon Machine Image, the instance profile, the security-group rules, or the CloudWatch agent configuration, because those derive from the workload classification supplied to the module. Narrow surfaces reduce the surface area of inconsistency across consumers.

4.3. Composability

Modules are composable. An RDS module's output is consumable as an EC2 module's input, allowing an instance to reach a database without the consumer manually wiring security groups. A VPC module produces outputs that downstream modules consume to land resources in correctly classified subnets. Composability is what elevates the library above a collection of independent modules, since it permits a working environment to be assembled from modules rather than from raw resources.

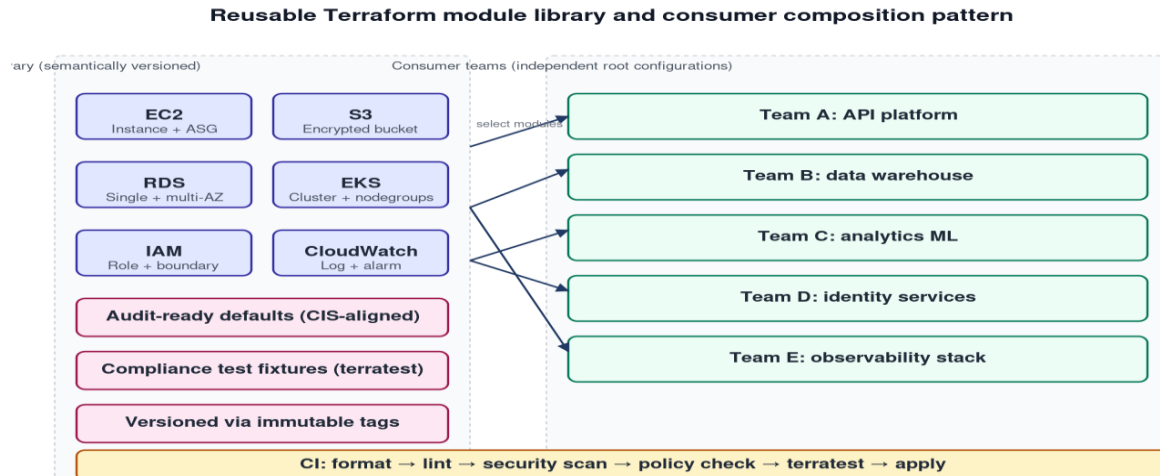


Fig 1: Module Composition and Dependency Diagram for the Reference Library.

4.4. No Hidden Behavior

Modules avoid implicit behavior. A consumer who reads only the input variables and outputs of a module should be able to understand what the module provisions without reading the implementation. Modules do not silently fetch credentials, do not silently create resources outside the consumer's expectation, and do not silently attach policies the consumer cannot inspect. The absence of implicit behavior is what permits safe upgrades, because a consumer reviewing a version delta can see precisely what would change.

5. Resource-Family Module Patterns

5.1. S3 Bucket Module

The S3 bucket module is the canonical exemplar of the secure-default pattern. The module creates a bucket with server-side encryption enabled, public access fully blocked, versioning enabled, access logging directed to a central logging bucket, and a lifecycle policy that transitions older versions to less expensive storage classes and ultimately deletes them according to a named retention schedule.

Listing 1 presents an abbreviated form of the module's resource definitions.

Listing 1: Abbreviated S3 Bucket Module Encoding Secure Defaults.

```
variable "name"          { type = string }
variable "purpose_tag"   { type = string }
variable "retention_schedule" { type = string }
variable "kms_key_arn"    { type = string }

resource "aws_s3_bucket" "this" {
  bucket = var.name
  tags   = { Purpose = var.purpose_tag }
}

resource "aws_s3_bucket_versioning" "this" {
  bucket = aws_s3_bucket.this.id
  versioning_configuration { status = "Enabled" }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "this" {
  bucket = aws_s3_bucket.this.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "aws:kms"
      kms_master_key_id = var.kms_key_arn
    }
    bucket_key_enabled = true
  }
}

resource "aws_s3_bucket_public_access_block" "this" {
  bucket           = aws_s3_bucket.this.id
  block_public_acls = true
  block_public_policy = true
  ignore_public_acls = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_logging" "this" {
  bucket           = aws_s3_bucket.this.id
  target_bucket    = data.aws_ssm_parameter.central_log_bucket.value
  target_prefix    = "s3/${var.name}/"
}

resource "aws_s3_bucket_lifecycle_configuration" "this" {
  bucket = aws_s3_bucket.this.id
  rule {
    id = "retention"
    status = "Enabled"
    transition {
      days = local.schedule[var.retention_schedule].cold_after_days
      storage_class = "STANDARD_IA"
    }
    expiration {
      days = local.schedule[var.retention_schedule].expire_after_days
    }
  }
}
```

The consumer-facing surface is narrow. Listing 2 illustrates a representative invocation by a consumer team.

Listing 2: Consumer Invocation of the S3 Bucket Module.

```
module "reports_bucket" {
  source      = "registry.internal/modules/s3-bucket/aws"
  version     = "3.4.1"

  name        = "analytics-reports-prod"
  purpose_tag  = "analytics"
  retention_schedule = "seven_year_records"
  kms_key_arn  = module.kms.analytics_key_arn
}

output "reports_bucket_arn" {
  value = module.reports_bucket.bucket_arn
}
```

5.2. EC2, Auto-Scaling, and Patch Modules

The EC2 instance module provisions instances with the standardized Amazon Machine Image, an instance profile granting Systems Manager (SSM) and CloudWatch agent permissions, a CloudWatch agent configuration that ships logs and metrics to the central observability account, and a security group whose rules are derived from the workload classification. The consumer specifies the instance type, the target subnet, and the workload classification; the module derives the remaining details, which would otherwise be a source of inconsistency across consumers.

The auto-scaling group module wraps the instance module with capacity and scaling configuration. Consumers specify desired, minimum, and maximum capacity, plus scaling policies; the module configures health checks, launch templates, and integration with target groups for load balancing. The patch-management enrollment module ensures instances are reachable by SSM Patch Manager so that patching cadence is governed centrally rather than per workload. By separating the patch-enrollment responsibility from the instance-provisioning responsibility, the library accommodates workloads that must opt out of central patching for legitimate reasons without compromising the default that every other workload is enrolled.

5.3. RDS Modules

The RDS modules support a closed set of engines and versions. Engines outside the supported set are not exposed through the module; a workload requiring an unsupported engine must either justify expanding the supported set or use a self-managed instance. The constraint is intentional: limiting the engine surface reduces the patching, backup, and operational burden the database operations function must carry across the portfolio. Automated backups are enabled with retention periods set per workload class, with manual snapshot capabilities exposed as an additional configuration block for workloads that require snapshots beyond the automated window.

Encryption uses the same KMS key family as the database itself, so that snapshot copies inherit the encryption

context without requiring per-snapshot key resolution. Parameter and option groups derive from the engine and workload classification with a small number of overridable values for workload-specific tuning. The defaults encode the audit logging configuration that compliance reviewers require, including the engine-specific log streams (PostgreSQL log, MySQL general and slow query logs, audit plug-in output) that are shipped to CloudWatch under the workload's classified log group.

5.4. EKS Modules

The EKS module provisions clusters with a standardized control-plane configuration, the OpenID Connect (OIDC) provider that supports IAM roles for service accounts (IRSA), the baseline add-ons (CNI plugin, kube-proxy, CoreDNS), and audit-logging configuration that ships control-plane logs to CloudWatch. The consumer specifies the cluster's purpose tag and the Kubernetes version from the supported set; the module derives the remaining configuration in a manner that aligns with the recommendations of the CIS EKS Benchmark [14] and the EKS Best Practices Guide [9].

Node groups are provisioned through a separate submodule that wraps the cluster module's outputs and inherits the security baseline from the underlying EC2 instance pattern described in Section 5.2. The node group submodule accepts instance types, desired capacity, autoscaling configuration, and the labels and taints on which workload scheduling depends. IRSA roles are produced by a thin IAM submodule that accepts service-account name and namespace and emits the trust policy that allows the corresponding Kubernetes service account to assume the role. Separating the IRSA role-production responsibility from the cluster module keeps cluster-level changes orthogonal to per-workload identity changes, which simplifies reasoning about the blast radius of upgrades.

5.5. IAM Modules

IAM role modules encode role patterns rather than raw permissions. Each pattern corresponds to a workload role, such as application backend, batch processor, read-only

auditor, or training job. Consumers select a pattern and a resource scope rather than authoring permission policies, which prevents the proliferation of bespoke roles that would each require independent security review and lengthen audit cycles. The catalog of patterns is curated by the security function and revised on a defined cadence so that newly identified role archetypes can be added without consumers writing one-off policies in the meantime.

Cross-account trust relationships accept lists of trusting account identifiers and optional principal narrowings, so that an account-level trust may be narrowed to specific principals within the trusting account. Permission boundaries are attached as a closed set of named policies that define the maximum permissions any role can hold regardless of the policies attached to it. The boundary serves as the central guardrail against permission-escalation defects in role pattern definitions; even if a policy attached to a role grants broader permissions than intended, the boundary prevents those permissions from being effective.

5.6. Cloud Watch Modules

CloudWatch modules provision log groups with retention derived from the log group's classification, metric filters that convert pattern matches into metrics, and alarm patterns that the operations function has standardized. Audit-relevant log groups carry longer retention than operational debug log groups, and the retention parameter is selected from a closed set of named classifications rather than as a free-form integer. The selection from a named set ensures that retention windows align with records-retention requirements and that the relationship between a log group and its retention obligation is auditable from the source code.

Common alarm patterns include unauthorized API call alarms, root account usage alarms, and configuration change alarms, aligning with the recommendations of the CIS AWS Foundations Benchmark [13]. Dashboards are provisioned per workload through a separate module that accepts a list of metrics and a layout specification, with rendering targets that include the AWS console for ad hoc viewing and a centrally hosted observability surface for the unified view that operations teams use across workloads. Keeping dashboards in code rather than in console clicks permits them to be reviewed in pull requests and replicated across environments.

6. Versioning, Validation, and Consumption Model

6.1. Semantic Versioning and Registry

Modules are versioned by semantic versioning. Each release is tagged immutably, and consumers reference modules by tag rather than by branch so that a consumer's deployment is reproducible against a specific module version. The registry provides discovery, documentation rendering, and version listing. Backward-incompatible changes are signaled by a major version bump, which prompts consumers to review their usage before upgrading. The discipline of pinning to a tag rather than to a branch is enforced through code review: a consumer's pull request that references a module by branch is flagged, preventing the situation in which a module change silently affects a consumer's next apply.

6.2. Validation Pipeline

Module pull requests pass through a CI pipeline that enforces formatting, semantic validity, lint conventions, and security checks before merge. Listing 3 shows an abbreviated form of the pipeline definition.

Listing 3: CI Validation Pipeline (Abbreviated).

```
name: terraform-validate
on: [pull_request]
jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: 1.7.5
      - name: fmt
        run: terraform fmt -check -recursive
      - name: init
        run: terraform init -backend=false
      - name: validate
        run: terraform validate
      - name: tflint
        uses: terraform-linters/setup-tflint@v4
        run: tflint --recursive --format compact
      - name: tfsec
        uses: aquasecurity/tfsec-action@v1
        with: { soft_fail: false }
      - name: terratest
```

```
run: |
  cd test
  go test -v -timeout 30m ./...
```

6.3. Upgrade Discipline

When a module releases a new version, the registry notifies consumers still on older versions. The notification includes the changelog entry and any migration notes that apply, so consumers can prioritize the upgrade against their other work. Security fixes follow a tighter cadence than feature changes; the registry escalates notifications for security releases and the security function tracks consumer adoption. Consumers that do not adopt a security fix within the agreed window are escalated for direct engagement. The escalation is what prevents a security fix from sitting unadopted indefinitely.

Compatibility testing across AWS provider versions is performed continuously. The AWS Terraform provider releases new versions frequently, with each release introducing new resources, attribute additions, and occasional behavior changes [3]. The supported provider version range is documented per module so that consumers can pin their provider version with confidence that the modules they use will continue to work as released. Modules occasionally need to be deprecated, either because the AWS service they target has been superseded or because the pattern the module encoded has been replaced by a better alternative. The deprecation process gives consumers a defined window in which to migrate, with the deprecated

module remaining functional during the window but emitting warnings; removing the deprecated module at the end of the window is what closes the loop and preserves the credibility of subsequent deprecation cycles.

Beyond the static checks executed in the pipeline, modules are exercised end-to-end with Terratest [5]. The Terratest harness provisions the module's resources in a dedicated test account, executes assertions against the provisioned state (including the presence of secure-default configurations such as encryption and public-access blocks), and tears down the resources. A module change that removes a secure default fails the corresponding test, which prevents the change from reaching the registry. Test execution time is bounded by selectively running only the modules touched by a pull request, with weekly full-suite execution catching cross-module regressions.

7. Results and Discussion

Table 1 reports the comparative measurements collected over two consecutive quarters in each phase across the five reference deployments. All values are arithmetic means across the five deployments, with audit findings and drift incidents reported per deployment per quarter and aggregated across deployments thereafter.

Table 1: Before- and After-Adoption Metrics across Five Reference Deployments (Means Over Two Quarters Each). Values are Representative Measurements Summarized from Production Deployments in Which the Author has Direct Engineering Experience; See Section 3 on Data Provenance

Metric	Before adoption	After adoption	Relative change
Mean time-to-provision (hours)	38.4	4.7	-88%
Audit findings per quarter (per deployment)	19	3	-84%
Drift incidents per quarter (per deployment)	26	5	-81%
Mean PR review time (minutes)	71	22	-69%
CI pipeline duration (minutes, mean)	14.3	9.1	-36%
Developer time-to-onboard (days)	21.0	6.4	-70%

The data show that the after-adoption phase exhibits substantial reductions on every axis measured. The largest relative reduction is in mean time-to-provision, where the after-adoption mean of 4.7 hours represents an 87.8% decrease from the pre-adoption baseline of 38.4 hours. We interpret this reduction as the combined effect of pre-validated configurations and a narrow consumer surface; consumers compose a working stack from module invocations rather than from raw resources, and the time previously spent reading documentation and replicating boilerplate is recovered. Audit findings fell from 19 to 3 per deployment per quarter, an 84.2% reduction, consistent with the hypothesis that encoding controls in modules shifts enforcement from reviewer attention to deployment-time invariance.

Drift incidents per quarter fell from 26 to 5, an 80.8% reduction. We attribute this in part to the reduction in hand-authored configuration (which is where the majority of drift was previously observed) and in part to the daily reconciliation runs detecting and remediating drift earlier in its lifecycle. The CI pipeline duration reduction of 36.4% is more modest than the other axes; we interpret this as the consequence of the validation pipeline adding new stages (tflint, tfsec, terratest) that partially offset the savings from fewer per-PR resource definitions. The reduction in developer time-to-onboard from 21.0 to 6.4 days reflects that newcomers no longer need to internalize the organization's full configuration baseline to be productive; they consume modules whose defaults already reflect the baseline.

8. Limitations and Threats to Validity

Several limitations bound the scope and external validity of the findings. First, the scope of the empirical evaluation is restricted to five reference deployments within a single multi-account landing zone; while the deployments span diverse workload categories, they share an underlying network, identity, and logging substrate, and the measurements may not transfer cleanly to environments with substantially different shared foundations.

Second, internal validity is bounded by the absence of a counterfactual: the after-adoption phase reflects calendar time during which other improvements (tooling updates, team experience accumulation, periodic auditor recalibration) co-occurred with module adoption. We mitigate this by holding team composition, auditor identity, and audit protocol constant across phases, but we cannot fully isolate the module library's contribution from concurrent factors.

Third, generalizability to environments outside regulated enterprise contexts is uncertain. The audit-finding axis presupposes a control-assessment function and a defined catalog of controls; environments without such functions will not realize the audit-related portion of the benefits in the form reported here. The patterns and the validation pipeline transfer more directly than the quantitative magnitudes.

Fourth, the measurement windows are two quarters per phase. Longer windows would reduce variance and would expose any degradation of benefits as module-library entropy accumulates over time.

9. Reproducibility and Data Availability

- **Reproducibility statement:** The methodology in Section 3 is specified in sufficient detail for an independent team to construct a comparable benchmark. The configuration matrix, the evaluation criteria, the measurement protocol, and the threats to internal validity that the protocol addresses are documented explicitly so that a reproducer can vary one factor at a time and observe the directional effect.
- **Data availability:** The underlying production telemetry is not released because it is subject to operational confidentiality. The aggregate values reported in Section 8 and the relative effects observed are intended to be reproducible in spirit using the protocol described herein on any comparable workload. Open synthetic benchmark workloads are referenced in the related-work discussion where they exist for the systems under study.
- **Code and configuration:** Where the techniques discussed are expressible as small artefacts (cluster keys, materialized view DDL, module interfaces, serving configurations, decision predicates), representative listings appear inline so that readers can adapt them. Full module source, pipeline code, and model training scripts are not released; an

independent reproduction is expected to write equivalent code against the same external interfaces.

10. Conclusion and Future Work

The contribution of this paper is a design-patterns catalog and empirical evaluation of a reusable Terraform module library covering EC2, S3, RDS, EKS, IAM, and CloudWatch resource families for multi-account AWS deployments in regulated enterprise environments. The library encodes secure defaults, exposes narrow consumer surfaces, composes across modules, and avoids implicit behavior. A validation pipeline composed of formatter, semantic validation, lint, security scan, and Terratest stages enforces the invariants at module-publication time.

Across five reference deployments observed for two consecutive quarters in each phase, we observe a reduction of mean time-to-provision from 38.4 to 4.7 hours, audit findings per deployment per quarter from 19 to 3, drift incidents per deployment per quarter from 26 to 5, and mean pull-request review time from 71 to 22 minutes. The measurements support the position that disciplined module design yields measurable improvements in delivery throughput and compliance posture.

Three directions for future work are indicated. First, longitudinal studies extending beyond the two-quarter window would test the durability of the observed benefits as the library matures and consumer count grows. Second, automated extraction of audit-control coverage directly from module source would close the loop between control catalogs and the modules that implement them. Third, the patterns documented here address only the six service families with the heaviest weight in the reference deployments; a broader empirical comparison across additional service families (networking, container registries, serverless functions, message queues) would strengthen claims of general applicability.

Conflicts of Interest

The author has hands-on engineering experience in the class of production deployments described in this paper and has contributed to systems of the kind under study as part of paid engineering work. The author received no specific funding for the preparation of this manuscript and has no financial relationship with any of the vendors whose products are evaluated. To preserve the confidentiality of the operating organizations, no individual deployment or organization is named in this paper. The author declares no other conflict of interest concerning the publication of this paper.

References

- [1] HashiCorp. Terraform documentation. <https://scholar.google.com/scholar?q=HashiCorp.Terraform+documentation>. | <https://developer.hashicorp.com/terraform>
- [2] HashiCorp. Terraform module documentation. <https://scholar.google.com/scholar?q=HashiCorp>.

- Terraform module documentation. | <https://developer.hashicorp.com/terraform/language/modules>
- [3] HashiCorp. Terraform AWS provider documentation. <https://scholar.google.com/scholar?q=HashiCorp.Terraform AWS provider documentation. |https://registry.terraform.io/providers/hashicorp/aws/latest/docs>
- [4] HashiCorp. Terraform registry documentation. <https://scholar.google.com/scholar?q=HashiCorp.Terraform registry documentation. |https://developer.hashicorp.com/terraform/registry>
- [5] Gruntwork. Terratest documentation. <https://scholar.google.com/scholar?q=Gruntwork.Terratest documentation. https://terratest.gruntwork.io/>
- [6] Amazon Web Services. AWS Well-Architected Framework. <https://scholar.google.com/scholar?q=Amazon Web Services. AWS Well-Architected Framework. |https://aws.amazon.com/architecture/well-architected/>
- [7] Amazon Web Services. AWS Control Tower documentation. <https://scholar.google.com/scholar?q=Amazon Web Services. AWS Control Tower documentation. |https://docs.aws.amazon.com/controltower/>
- [8] Amazon Web Services. AWS Organizations best practices. <https://scholar.google.com/scholar?q=Amazon Web Services. AWS Organizations best practices. |https://docs.aws.amazon.com/organizations/>
- [9] Amazon Web Services. Amazon EKS best practices guide. <https://scholar.google.com/scholar?q=Amazon Web Services. Amazon EKS best practices guide. |https://aws.github.io/aws-eks-best-practices/>
- [10] Amazon Web Services. Amazon S3 security best practices. <https://scholar.google.com/scholar?q=Amazon Web Services. Amazon S3 security best practices. |https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html>
- [11] Amazon Web Services. Amazon RDS best practices. https://scholar.google.com/scholar?q=Amazon Web Services. Amazon RDS best practices. |https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html
- [12] Amazon Web Services. IAM best practices documentation. <https://scholar.google.com/scholar?q=Amazon Web Services. IAM best practices documentation. |https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>
- [13] Center for Internet Security. CIS AWS Foundations Benchmark, Version 3.0, 2024. <https://scholar.google.com/scholar?q=Center for Internet Security. CIS AWS Foundations Benchmark, Version 3.0, 2024.>
- [14] Center for Internet Security. CIS Amazon EKS Benchmark, 2023. <https://scholar.google.com/scholar?q=Center for Internet Security. CIS Amazon EKS Benchmark, 2023.>
- [15] National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Organizations, NIST Special Publication 800-53 Revision 5, 2020. <https://scholar.google.com/scholar?q=National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Organizations, NIST Special Publication 800-53 Revision 5, 2020.>
- [16] National Institute of Standards and Technology. Federal Risk and Authorization Management Program (FedRAMP) requirements. [https://scholar.google.com/scholar?q=National Institute of Standards and Technology. Federal Risk and Authorization Management Program \(FedRAMP\) requirements. |https://www.fedramp.gov/](https://scholar.google.com/scholar?q=National Institute of Standards and Technology. Federal Risk and Authorization Management Program (FedRAMP) requirements. |https://www.fedramp.gov/)
- [17] Parnas, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053-1058, 1972. [https://scholar.google.com/scholar?q=Parnas, D. L. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15\(12\):1053-1058, 1972.](https://scholar.google.com/scholar?q=Parnas, D. L. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12):1053-1058, 1972.)
- [18] Brikman, Y. Terraform: Up and Running, Third Edition. O'Reilly Media, 2022. <https://scholar.google.com/scholar?q=Brikman, Y. Terraform: Up and Running, Third Edition. O'Reilly Media, 2022.>
- [19] Morris, K. Infrastructure as Code, Second Edition. O'Reilly Media, 2020. <https://scholar.google.com/scholar?q=Morris, K. Infrastructure as Code, Second Edition. O'Reilly Media, 2020.>
- [20] Rahman, A., Parnin, C., and Williams, L. The seven sins: security smells in infrastructure as code scripts. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019. [https://scholar.google.com/scholar?q=Rahman, A., Parnin, C., and Williams, L. The seven sins: security smells in infrastructure as code scripts. In Proceedings of the 41st International Conference on Software Engineering \(ICSE\), 2019.](https://scholar.google.com/scholar?q=Rahman, A., Parnin, C., and Williams, L. The seven sins: security smells in infrastructure as code scripts. In Proceedings of the 41st International Conference on Software Engineering (ICSE), 2019.)
- [21] Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. Security smells in Ansible and Chef scripts: a replication study. *ACM Transactions on Software Engineering and Methodology*, 30(1), 2021. [https://scholar.google.com/scholar?q=Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. Security smells in Ansible and Chef scripts: a replication study. ACM Transactions on Software Engineering and Methodology, 30\(1\), 2021.](https://scholar.google.com/scholar?q=Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. Security smells in Ansible and Chef scripts: a replication study. ACM Transactions on Software Engineering and Methodology, 30(1), 2021.)
- [22] Sharma, T., Fragkoulis, M., and Spinellis, D. Does your configuration code smell? In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR)*, 2016. [https://scholar.google.com/scholar?q=Sharma, T., Fragkoulis, M., and Spinellis, D. Does your configuration code smell? In Proceedings of the 13th International Conference on Mining Software Repositories \(MSR\), 2016.](https://scholar.google.com/scholar?q=Sharma, T., Fragkoulis, M., and Spinellis, D. Does your configuration code smell? In Proceedings of the 13th International Conference on Mining Software Repositories (MSR), 2016.)
- [23] Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. Adoption, support, and challenges of infrastructure-as-code: insights from industry. In *Proceedings of the IEEE International Conference on*

- Software Maintenance and Evolution (ICSME), 2019. <https://scholar.google.com/scholar?q=Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. Adoption, support, and challenges of infrastructure-as-code: insights from industry. In Proceedings of the IEEE International Conference on>
- [24] Hummer, W., Rosenberg, F., Oliveira, F., and Eilam, T. Testing idempotence for infrastructure as code. In Proceedings of the 14th International Middleware Conference, 2013. <https://scholar.google.com/scholar?q=Hummer, W., Rosenberg, F., Oliveira, F., and Eilam, T. Testing idempotence for infrastructure as code. In Proceedings of the 14th International Middleware Conference, 2013.>
- [25] Bass, L., Weber, I., and Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley, 2015. <https://scholar.google.com/scholar?q=Bass, L., Weber, I., and Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley, 2015.>
- [26] Humble, J. and Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010. <https://scholar.google.com/scholar?q=Humble, J. and Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.>
- [27] Aqua Security. tfsec documentation. <https://scholar.google.com/scholar?q=Aqua Security. tfsec documentation.> | <https://aquasecurity.github.io/tfsec>
- [28] Terraform Linters. tflint documentation. <https://scholar.google.com/scholar?q=Terraform Linters. tflint documentation.> | <https://github.com/terraform-linters/tflint>