1 Terraform State Architecture Rollout Kit (Multi-Environment)

Version: 2026-02-24
Author: Mathieu Kessler
Site: https://www.talk-nerdy-to-me.com
Companion article: https://www.talk-nerdy-to-me.com/blog/terraform-state-management-scale-environment-isolation

1.1 Purpose

This playbook helps teams move from ad hoc Terraform state management to a boundary-driven architecture that scales across multiple environments and many services.

It focuses on the real operational problem: - environment isolation - blast radius control - CI-only production apply - drift detection at scale - safe migration from messy layouts

It does **not** assume Terraform workspaces are a security boundary.

1.2 Core Principle

Remote backends are necessary for collaboration, durability, and locking.

They are not a complete state architecture.

A state architecture must answer: - What belongs together in one state file? - What must be separated by environment? - Who can read/write each state? - How do plans and applies happen safely? - How do we detect and remediate drift?

1.3 State Boundary Rules (Recommended)

1. Split by environment first (`dev`, `staging`, `prod` are separate state boundaries)
2. Split by owner / blast radius second (`platform` vs `service`)
3. Split by change cadence third (shared infra vs fast-moving app stacks)
4. Keep state small enough for reviewable plans and low lock contention
5. Do not mix shared networking/DNS with app compute in the same state file
6. Document dependencies explicitly (outputs/data sources/orchestration)

1.4 Reference Repository Shape (Conceptual)

```
infra/
  modules/
    networking/
    dns/
    service-api/
    service-worker/

  live/
    prod/
      platform/
        networking/
        dns/
        observability/
      services/
        payments-api/
        orders-api/
        web/
    staging/
      platform/
        networking/
        dns/
      services/
        payments-api/
        orders-api/
        web/
    dev/
      platform/
        networking/
        dns/
      services/
        payments-api/
        orders-api/
```

```
        web/

  ci/
    drift-check/
    policy-check/
```

1.5 Backend Key Naming Pattern (Conceptual)

```
org/platform/prod/networking.tfstate
org/platform/prod/dns.tfstate
org/services/prod/payments-api.tfstate
org/services/staging/payments-api.tfstate
org/services/dev/web.tfstate
```

Naming goals: - environment is obvious - owner category is obvious - stack name is obvious - easy IAM scoping and audit reporting

1.6 Pattern Decision Guide

1.7 1) Workspace per Environment

Use when: - a single stack is replicated across environments - same credentials/access model is acceptable - low compliance / low blast-radius requirements

Failure modes: - operator selects wrong workspace - weak environment isolation - easy accidental cross-environment apply

Rule of thumb: - convenience feature, not a security boundary

1.8 2) Directory per Environment

Use when: - you need explicit boundaries and review clarity - environments require separate backends and/or access controls - production safety matters more than minimizing folders

Failure modes: - copy/paste drift if modules and conventions are weak - directory sprawl without standards

Rule of thumb: - best default for most production teams

1.9 3) Terragrunt Orchestration

Use when: - you manage many stacks with dependency-aware orchestration needs - Terraform/OpenTofu layout discipline alone is not enough - you need DRY config composition at scale

Failure modes: - extra abstraction layer to govern - CLI behavior changes impact wrapper scripts and team habits - execution scope surprises if upgrades are not migration-tested

Rule of thumb: - powerful when you need orchestration discipline, not just DRY syntax

1.10 Guardrails (Non-Negotiables)

- Separate cloud accounts/subscriptions/projects per environment whenever possible
- Separate backend objects/keys per environment and stack category
- **Production apply is CI-only** (no ad hoc laptop apply)
- Plan artifacts reviewed before apply
- Environment approvals / required reviewers for production workflows
- Least-privilege state and infra credentials
- Terraform and Terragrunt versions pinned in CI
- Scheduled drift detection with owner routing

1.11 CI-Only Production Apply Pattern (Conceptual)

```yaml
name: Terraform Apply (Prod)

on:
  workflow_dispatch:
    inputs:
      stack:
        description: "Stack path"
        required: true

permissions:
  contents: read
  id-token: write

jobs:
  plan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
      - run: terraform -chdir=infra/${{ inputs.stack }} init -
        input=false
      - run: terraform -chdir=infra/${{ inputs.stack }} plan -
        input=false -out=tfplan
      - uses: actions/upload-artifact@v4
        with:
          name: tfplan-${{ inputs.stack }}
          path: infra/${{ inputs.stack }}/tfplan

  apply:
    needs: plan
    runs-on: ubuntu-latest
    environment: production  # protect with required reviewers
```

```
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
      - uses: actions/download-artifact@v4
        with:
          name: tfplan-${{ inputs.stack }}
          path: infra/${{ inputs.stack }}
      - run: terraform -chdir=infra/${{ inputs.stack }} init -
        input=false
      - run: terraform -chdir=infra/${{ inputs.stack }} apply -
        input=false tfplan
```

1.12 Drift Detection SOP (Scheduled)

Why teams suffer from drift: - they only discover it during incidents or urgent changes - drift checks are manual and inconsistent - outputs are noisy and not owner-routed

Recommended pattern: - run drift checks in CI on a schedule - scope by stack boundary, not “all infra in one shot” - use `terraform plan -refresh-only -detailed-exitcode` - route findings to owners with evidence - track remediation like reliability debt

1.12.1 Exit code semantics (important)

- `0` = no changes
- `2` = drift / changes detected
- `1` = error (broken check, auth issue, provider issue, etc.)

1.13 Wrong Prod Apply Runbook (Condensed)

1. **Contain**

- Freeze further applies for affected stack
- Stop parallel “quick fixes” from multiple laptops

2. **Capture evidence**

- Commit, operator, timestamp, stack path/workspace, logs
- State snapshot / backend version metadata

3. **Assess impact**

- What changed?
- Any replacements/deletions?
- Downstream impact (DNS/networking/IAM/app dependencies)?

4. **Choose recovery path**

- forward-fix (preferred when understood)
- rollback via reviewed Terraform change
- provider-native restore only if incident severity requires it

5. **Restore guardrails**

- move all follow-up changes into CI-only flow
- tighten the controls that allowed the event

6. **Post-incident improvements**

- redesign state boundaries if blast radius was too large
- reduce credential scope
- improve review/approval workflow
- update runbooks and training

1.14 Terragrunt Upgrade Checklist (CLI Safety)

- Pin Terragrunt version in CI
- Read release notes before upgrade
- Test CLI changes in a non-prod pipeline
- Validate `--filter` behavior and execution scope assumptions
- Review wrapper scripts and docs used by engineers
- Roll out to one team/repo first
- Monitor for unexpected multi-unit executions

1.15 Migration Plan (No Big-Bang Freeze)

1. Inventory current states and owners
2. Define target boundaries (env + platform/service split)
3. Enforce CI-only prod apply before prod migration
4. Migrate one non-prod service stack first
5. Validate drift checks, approvals, rollback steps
6. Migrate shared platform stacks separately with extra review depth
7. Pin versions during migration (avoid simultaneous CLI upgrades)
8. Repeat by bounded slices, not all stacks at once

1.16 What Success Looks Like

- Production applies happen through reviewed CI workflows only
- State boundaries match ownership and blast radius
- Drift is discovered on schedule, not during outages
- Lock contention is reduced because states are smaller
- Engineers do not rely on workspace switching memory for safety

1.17 Source Verification (Checked 2026-02-24)

Primary sources: - Terraform workspaces docs: https://developer.hashicorp.com/terraform/language/state/workspaces - Terraform state docs: https://developer.hashicorp.com/terraform/language/state - Terraform backend docs: https://developer.hashicorp.com/terraform/language/backend - Terraform plan docs: https://developer.hashicorp.com/terraform/cli/commands/plan - Terragrunt v0.98.0

release notes: https://github.com/gruntwork-io/terragrunt/releases/tag/v0.98.0 - Terragrunt CLI redesign docs (`--filter`): https://terragrunt.gruntwork.io/docs/migrate/cli-redesign/#use-the-new-filter-flag - Terragrunt RFC / issue #4060: https://github.com/gruntwork-io/terragrunt/issues/4060

1.18 Links

- Deep dive article: https://www.talk-nerdy-to-me.com/blog/terraform-state-management-scale-environment-isolation
- Companion playbook page: https://www.talk-nerdy-to-me.com/playbooks/terraform-state-architecture-rollout-kit