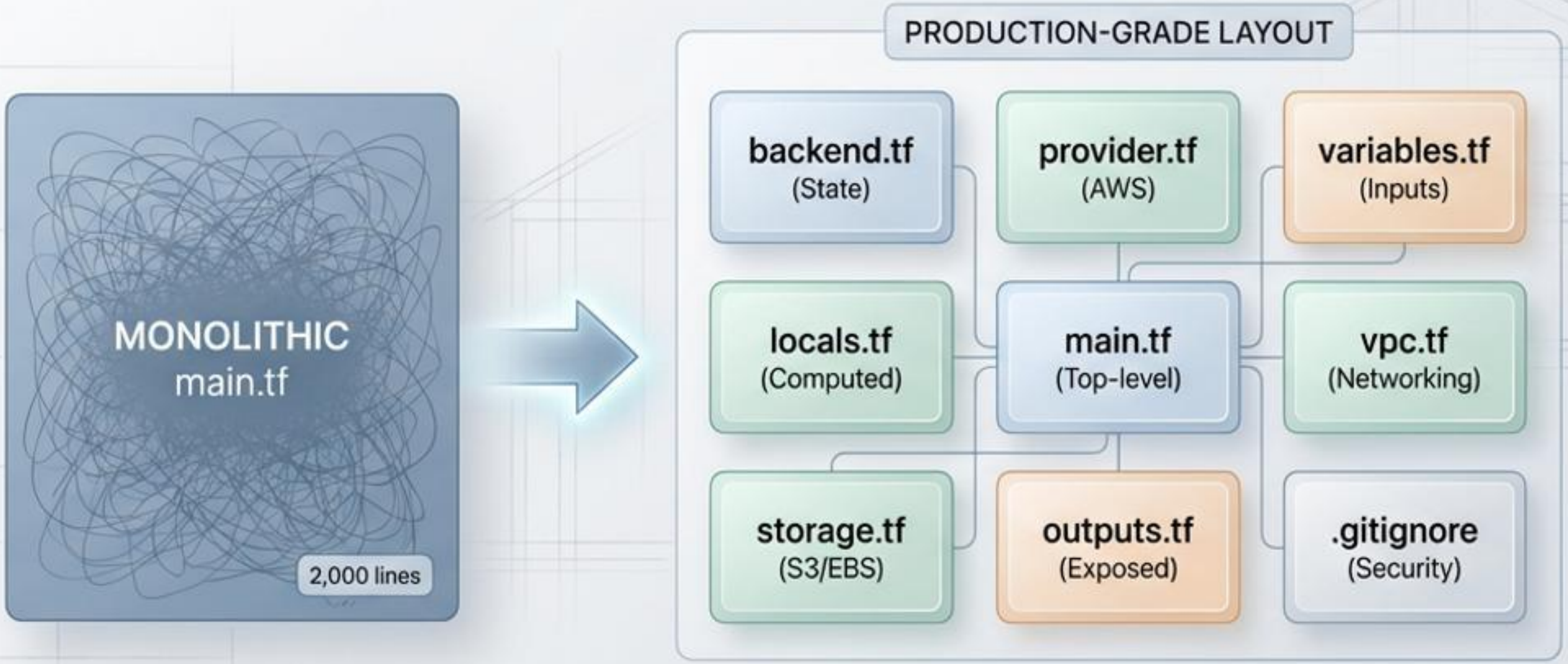
DAY 6/28

TERRAFORM FILE STRUCTURE & BEST PRACTICES



GOAL: From Monolithic to Organized, Maintainable, Team-Ready

READABILITY　MAINTAINABILITY　COLLABORATION　GIT HYGIENE

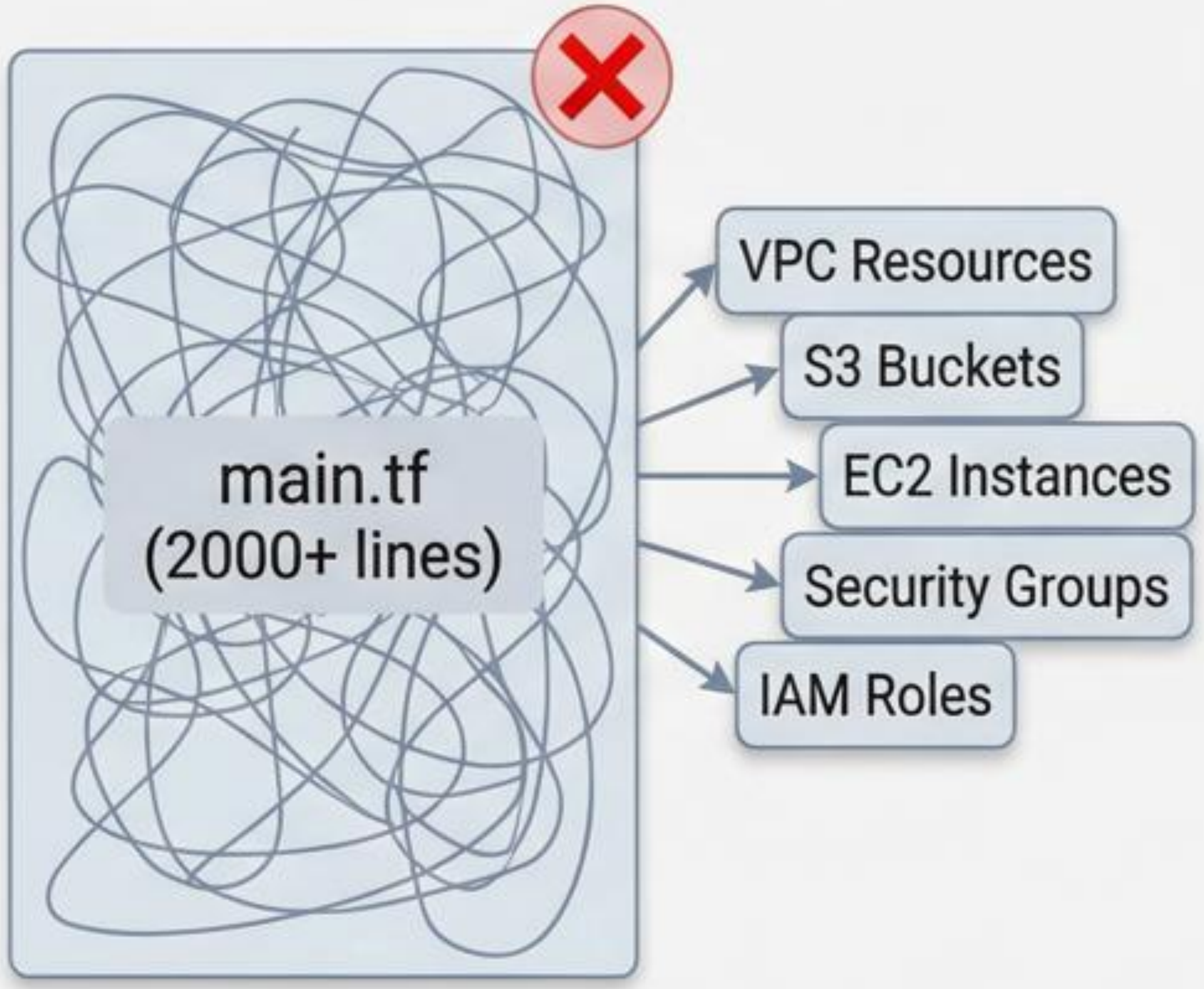
TERRAFORM AWS LABS

github.com/Push/terraform-aws-labs/day06

Structure for Maintainability: Functional Grouping

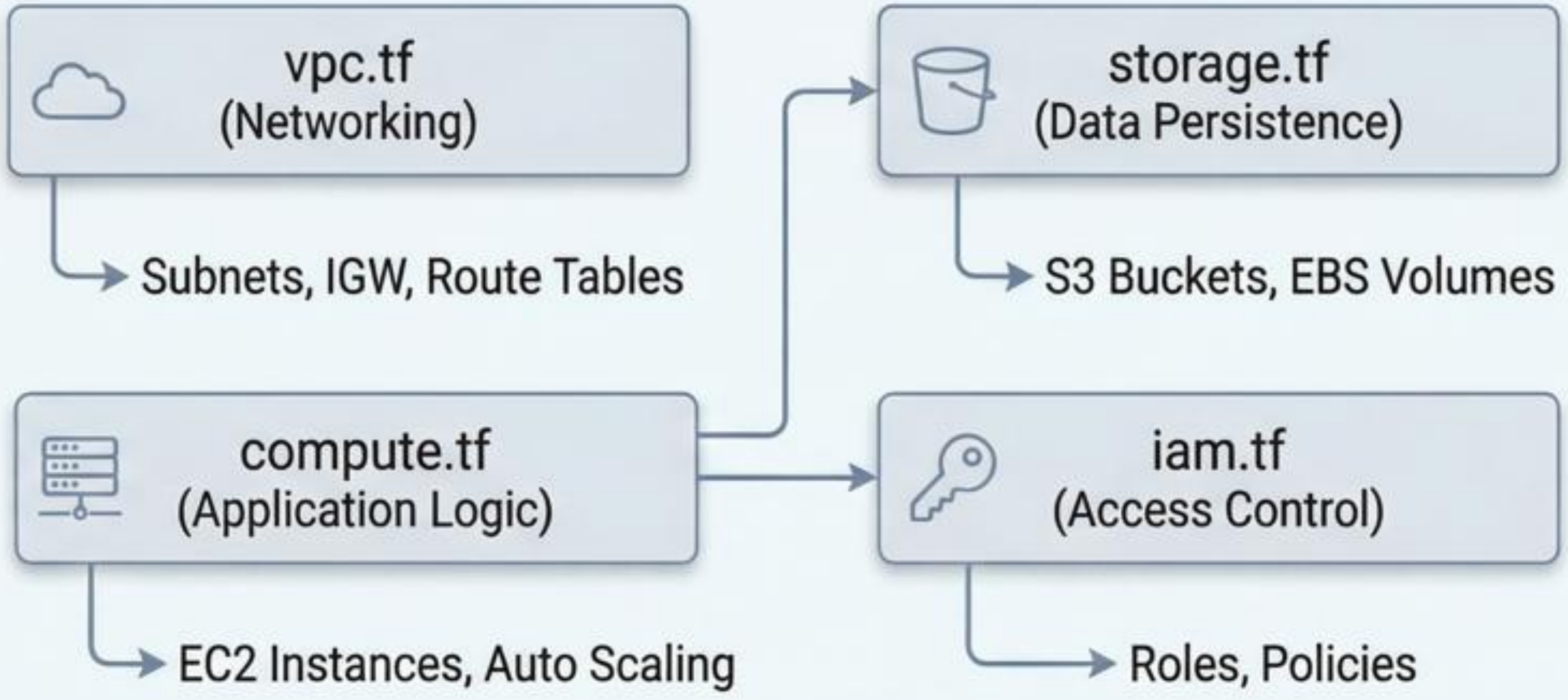
Organize files by business function, not resource type, for minimal impact and focused reviews.

❌ Anti-Pattern: Monolithic main.tf



Cross-cutting changes, difficult reviews, high blast radius

✅ Production Practice: Functional Modules



Benefits:

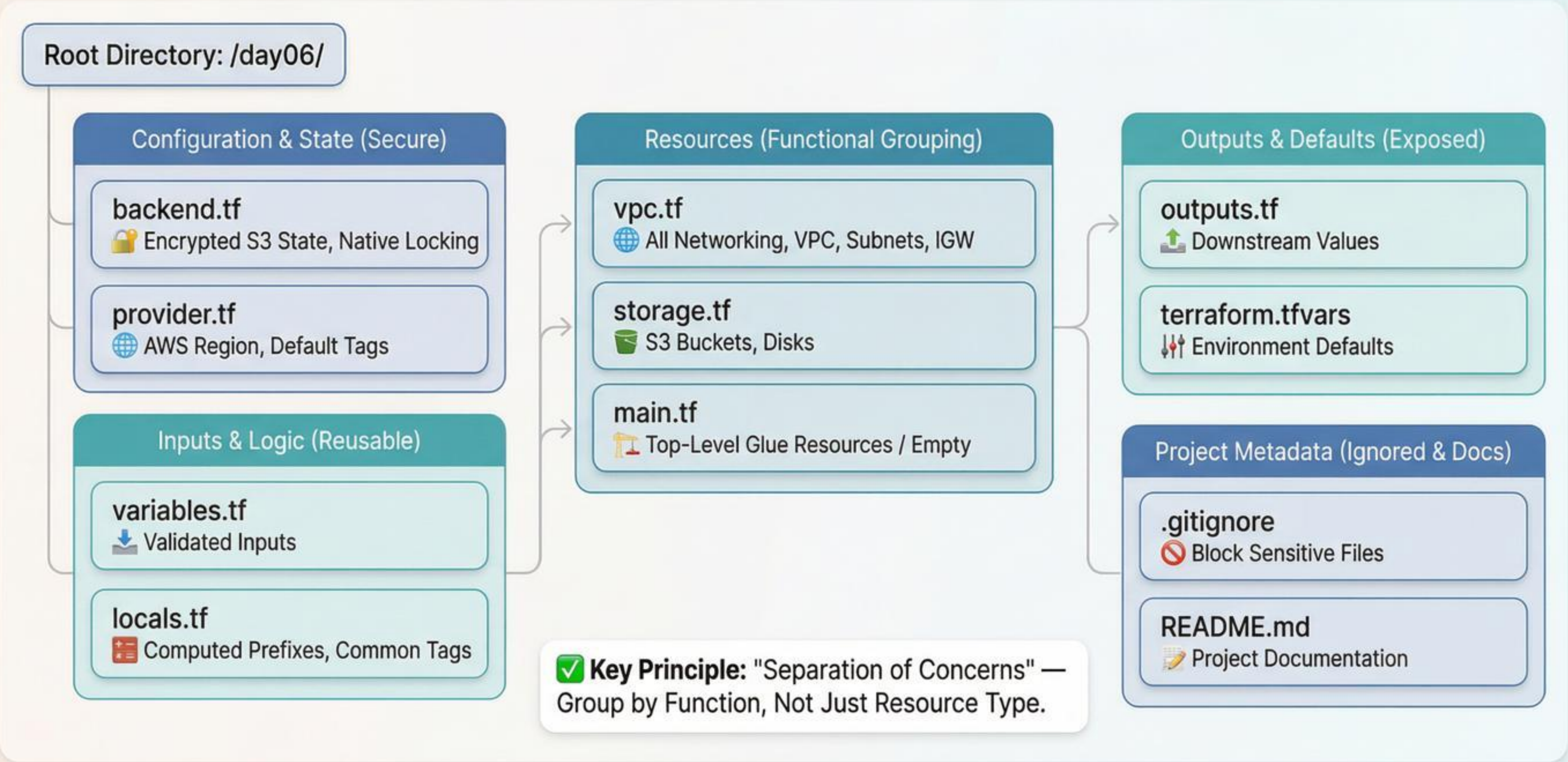
✅ **Minimal Cross-Cutting Changes**: Modify networking without touching storage.

✅ **Focused Code Reviews**: Smaller, domain-specific files for faster approval.

✅ **Reduced Blast Radius**: Isolate changes to single functions.

GOAL: From Monolithic to Organized, Maintainable, Team-Ready. Readability. Maintainabity. Collaboration. Git Hygiene.

Recommended Production-Grade Terraform File Structure



Root Directory: /day06/

Configuration & State (Secure)

backend.tf
🔐 Encrypted S3 State, Native Locking

provider.tf
🌐 AWS Region, Default Tags

Inputs & Logic (Reusable)

variables.tf
📥 Validated Inputs

locals.tf
🧮 Computed Prefixes, Common Tags

Resources (Functional Grouping)

vpc.tf
🌐 All Networking, VPC, Subnets, IGW

storage.tf
🪣 S3 Buckets, Disks

main.tf
🏗️ Top-Level Glue Resources / Empty

Outputs & Defaults (Exposed)

outputs.tf
📤 Downstream Values

terraform.tfvars
🎚️ Environment Defaults

Project Metadata (Ignored & Docs)

.gitignore
🚫 Block Sensitive Files

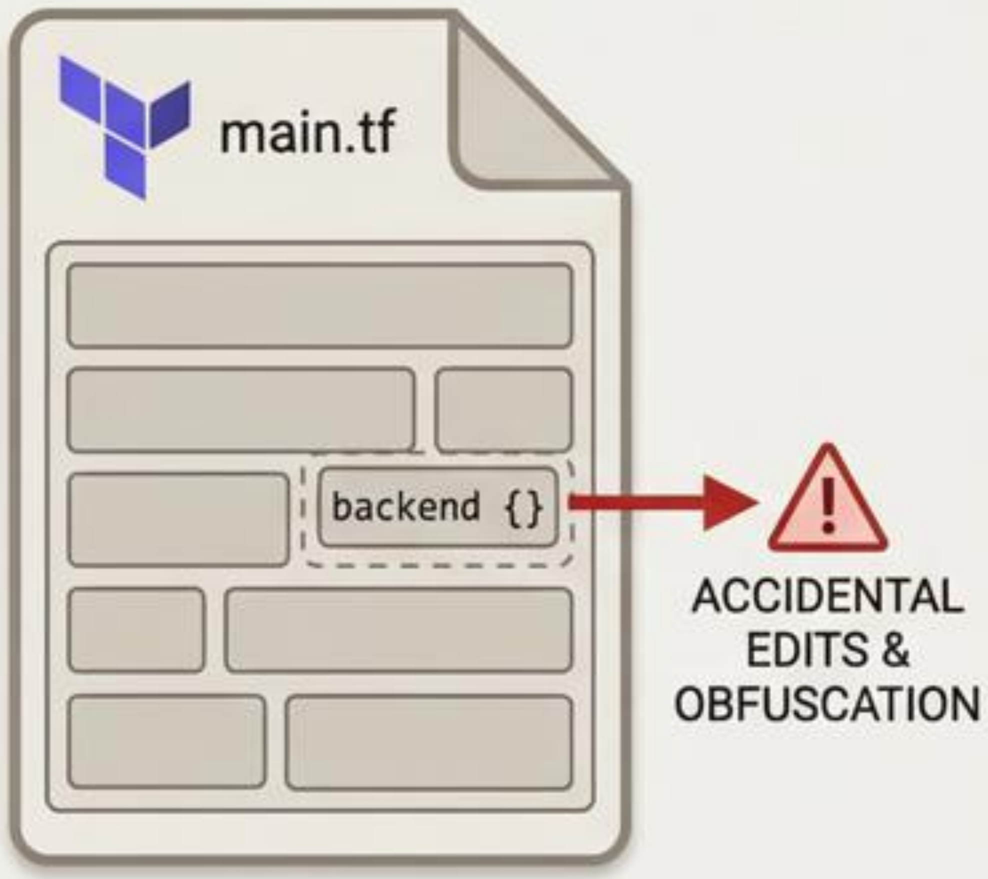
README.md
📝 Project Documentation

✅ **Key Principle:** "Separation of Concerns" — Group by Function, Not Just Resource Type.

github.com/Push/terraform-aws-labs/day06

STRATEGIC BACKEND ISOLATION: ENHANCING STATE MANAGEMENT SECURITY & CLARITY

MONOLITHIC MAIN.TF RISK



Risk of modifying state location or locking parameters within general resource configuration, leading to potential state corruption or conflicts.

ISOLATED BACKEND.TF PATTERN

```
# 🔒 backend.tf
terraform {
  backend "s3" { … }
}
```

WORKFLOW IMPLICATION

```
>_ terraform init -reconfigure
```

⚠ **MANDATORY** on any backend change. Explicit action prevents silent state divergence.

KEY BENEFITS & IMPLICATIONS

PREVENTION

Reduces risk of accidental state configuration modification during routine resource updates.

DOCUMENTATION

Serves as the single, unambiguous source of truth for state location, encryption, and locking mechanisms.

WORKFLOW SIGNAL

Forces explicit re-initialization via `terraform init --reconfigure` for any backend alteration, ensuring intentionality.

TEAM CLARITY

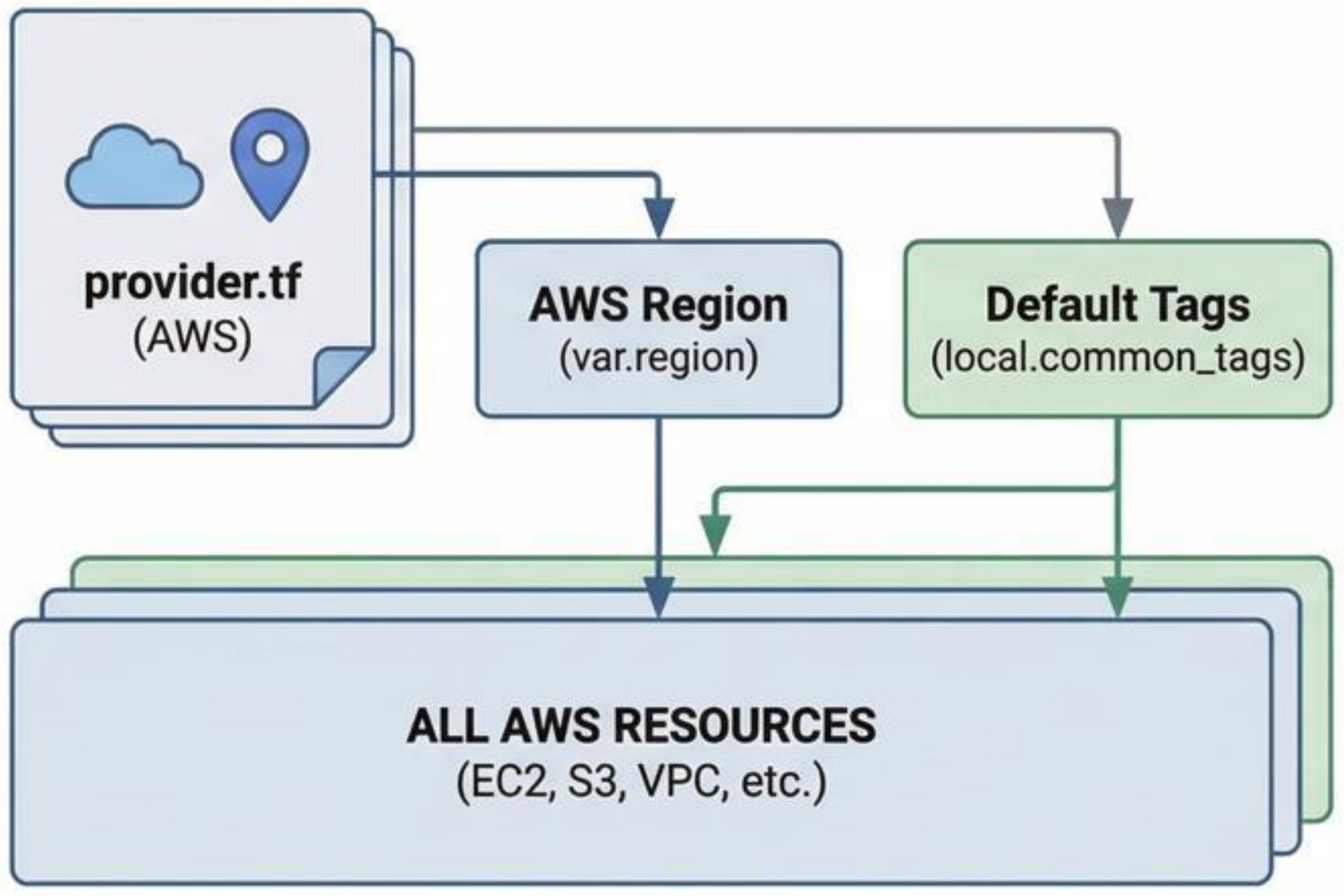
Provides immediate visibility into critical state infrastructure for all engineers, enhancing collaboration.

Isolating the backend block in its own file clarifies that any change to bucket, key or encryption demands `terraform init -reconfigure`; keeping it separate prevents accidental edits and documents the single source of truth for state location and locking.

DAY 6/28: TERRAFORM FILE STRUCTURE & BEST PRACTICES

Provider.tf – Cloud Provider & Default Tags

CLOUD PROVIDER CONFIGURATION



BEST PRACTICE: provider.tf

```
# 🌐 provider.tf
provider "aws" {
  region = var.region

  # ✅ Apply common tags to ALL resources (DRY!)
  default_tags {
    tags = local.common_tags
  }
}
```

✅ BENEFIT

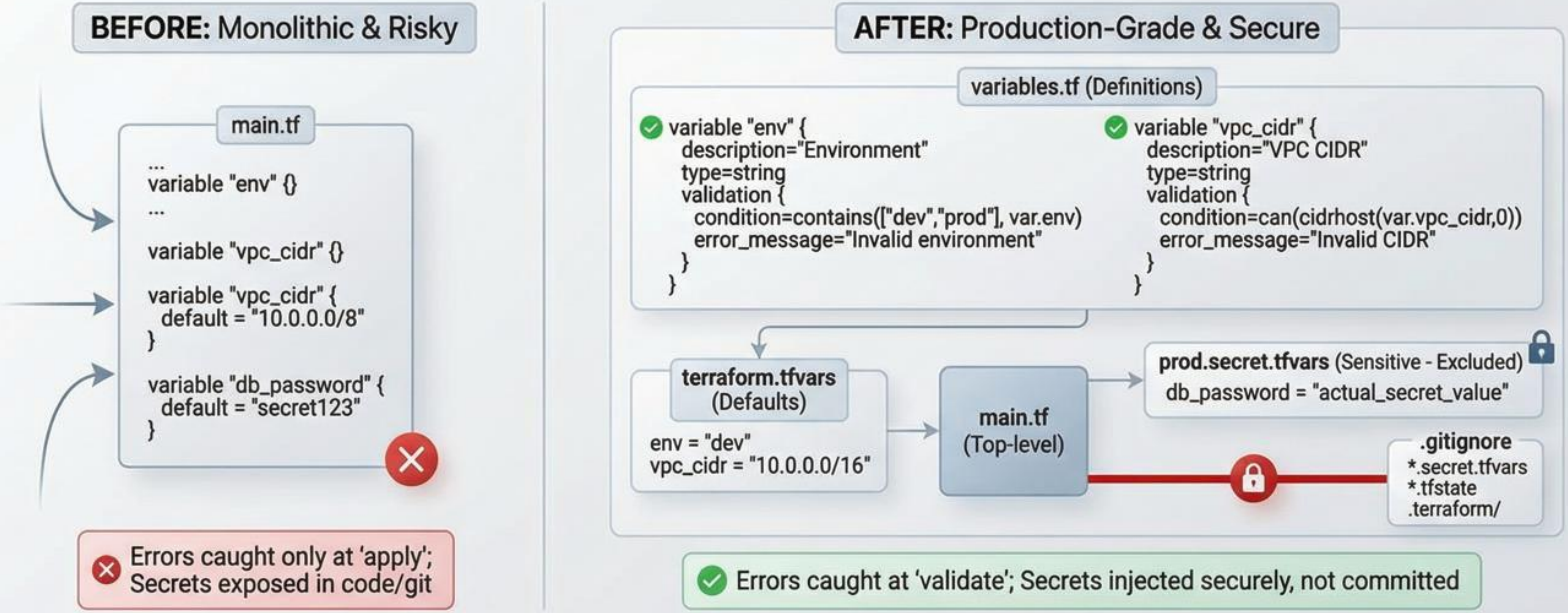
`default_tags` eliminates repeating `tags = {...}` in *every* resource and guarantees consistent ownership, environment, and application labels across the entire footprint.

TERRAFORM
AWS LABS

github.com/Push/terraform-aws-labs/day06

INPUT VARIABLE BEST PRACTICES: VALIDATION & SECURITY

Goal: Catch Errors Early & Secure Sensitive Data



BEFORE: Monolithic & Risky

main.tf

```
...
variable "env" {}
...

variable "vpc_cidr" {}

variable "vpc_cidr" {
  default = "10.0.0.0/8"
}

variable "db_password" {
  default = "secret123"
}
```

☒ Errors caught only at 'apply';
Secrets exposed in code/git

AFTER: Production-Grade & Secure

variables.tf (Definitions)

```
✓ variable "env" {
    description="Environment"
    type=string
    validation {
      condition=contains(["dev","prod"], var.env)
      error_message="Invalid environment"
    }
  }
```

```
✓ variable "vpc_cidr" {
    description="VPC CIDR"
    type=string
    validation {
      condition=can(cidrhost(var.vpc_cidr,0))
      error_message="Invalid CIDR"
    }
  }
```

terraform.tfvars
(Defaults)

```
env = "dev"
vpc_cidr = "10.0.0.0/16"
```

prod.secret.tfvars (Sensitive - Excluded)

```
db_password = "actual_secret_value"
```

.gitignore

```
*.secret.tfvars
*.tfstate
.terraform/
```

✓ Errors caught at 'validate'; Secrets injected securely, not committed

Summary: Use descriptions, types, and validation for robustness; isolate sensitive values from version control.

TERRAFORM AWS LABS

github.com/Push/terraform-aws-labs/day06

Locals.tf: Computed Values & DRY Logic

Centralized Naming, Tags, and Computed Values for Maintainability.

❌ Hardcoded & Repetitive (Before Locals)

```
resource "aws_s3_bucket" "app_bucket" {
  bucket = "my-app-dev-logs"
  tags = {
    Environment = "dev"
    Owner       = "team-a"
    Application = "my-app"
  }
}

resource "aws_ec2_instance" "web" {
  tags = {
    Environment = "dev"
    Owner       = "team-a"
    Application = "my-app"
  }
}

# Duplication & Error-Prone
```

✅ Centralized & Dynamic (With Locals)

```
# locals.tf
locals {
  # Reusable Naming Convention
  name_prefix = "${var.app}-${var.env}"

  # Consistent Tag Map
  common_tags = {
    Environment = var.env
    Application = var.app
    Owner       = "terraform-team"
    Terraform   = "true"
  }

  # Globally Unique S3 Bucket Name
  bucket_name = "${local.name_prefix}-logs-${random_string.suffix.result}"
}
```

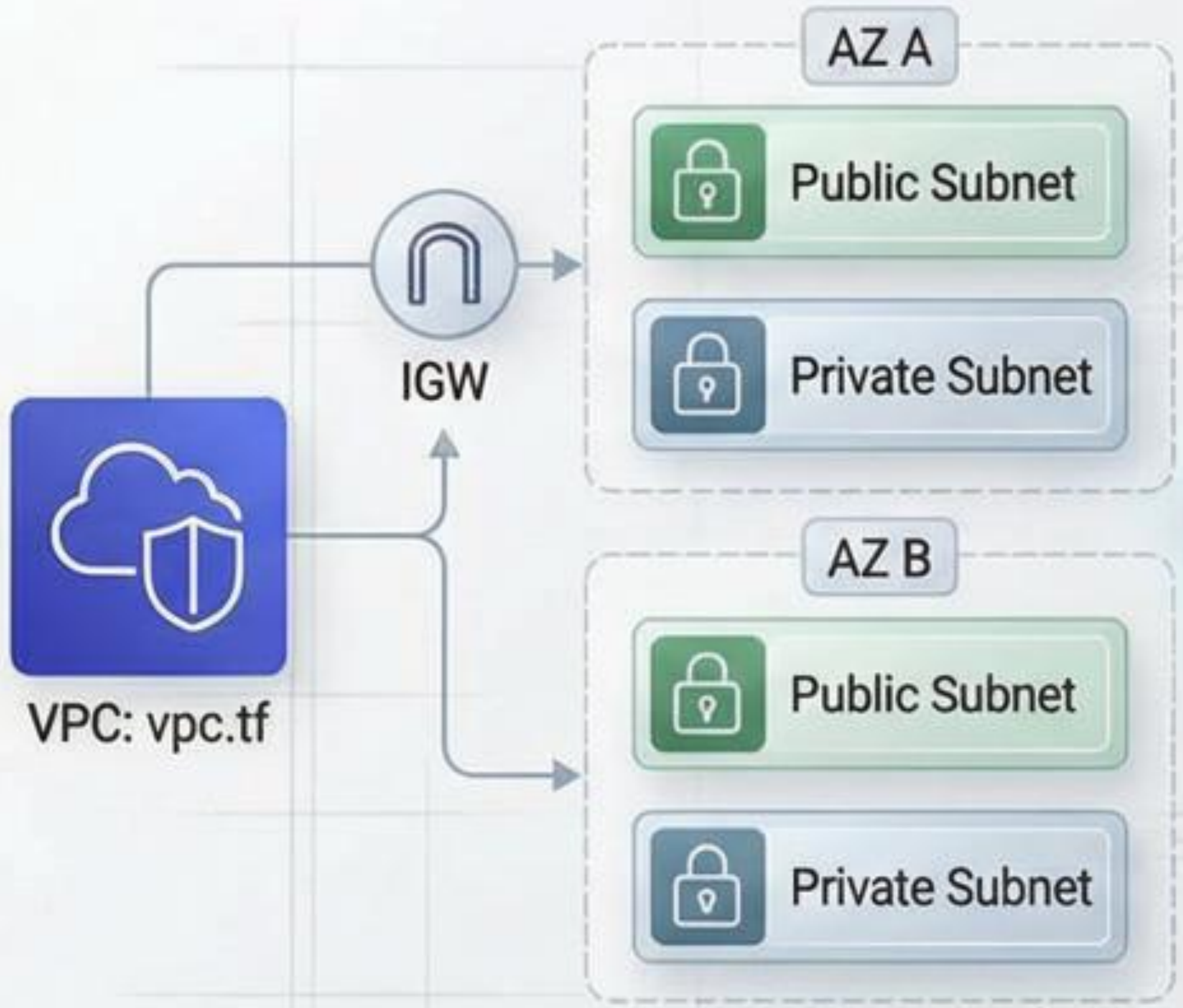
```
# storage.tf
resource "aws_s3_bucket" "logs" {
  bucket = local.bucket_name
  tags   = local.common_tags
}
# Single Source of Truth, Cascading Updates
```

KEY BENEFITS

- 🔄 **Eliminates Duplication (DRY):** Define once, use everywhere.
- 🚀 **Simplifies Updates:** Change `var.app` or `var.env` in one place, everything updates automatically.
- 🛡️ **Ensures Consistency:** Uniform naming and tags across all resources.
- ✅ **Prevents Typos:** Reduces human error from manual repetition.

Day 6 Goal: Move to Organized, Maintainable Structure
github.com/Push/terraform-aws-labs/day06

NETWORKING: OPTIMIZED vpc.tf & MULTI-AZ STRATEGY



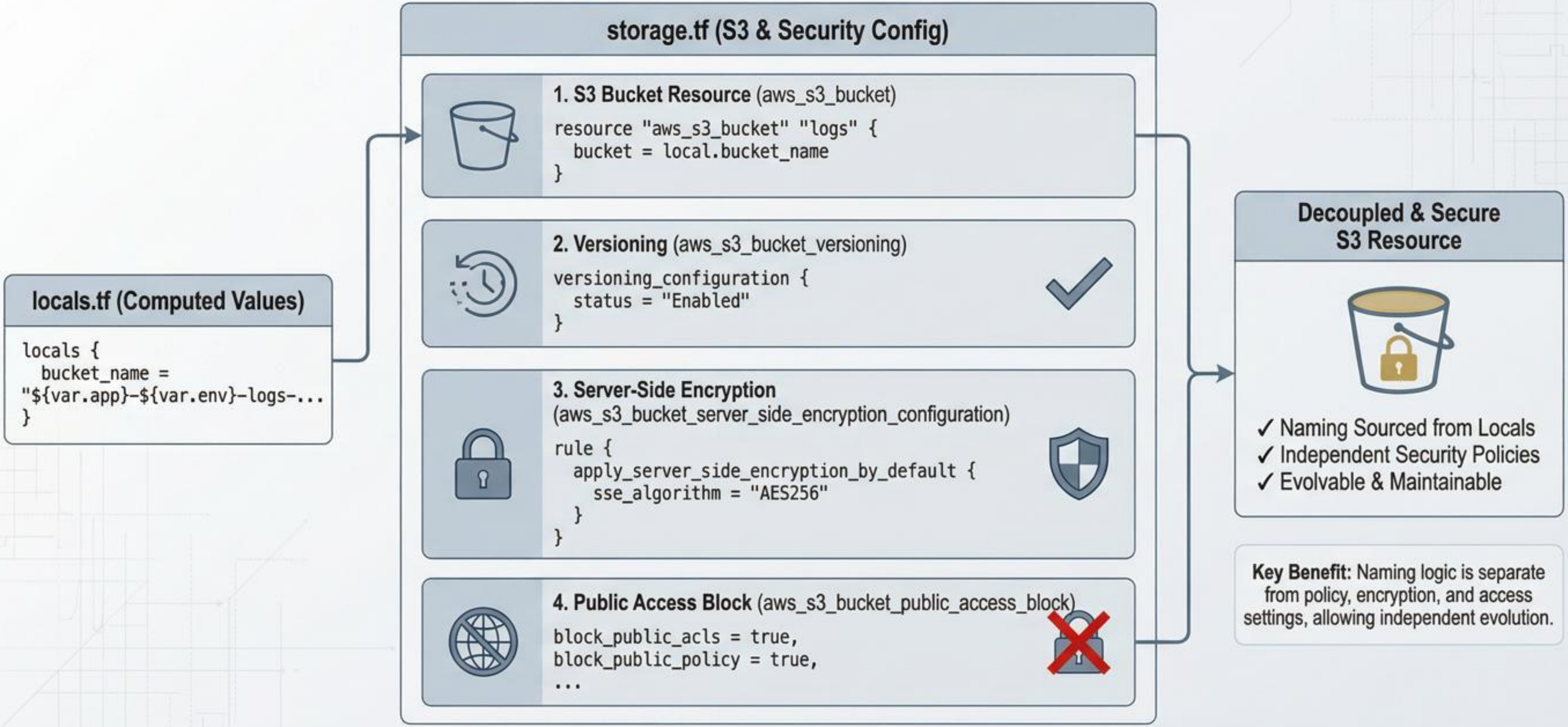
vpc.tf Examples

```
resource 'aws_vpc' 'main' {
  ...
  tags = {
    Name = local.name_prefix
  }
}

resource 'aws_subnet' 'public' {
  for_each = toset(var.azs)
  ...
  tags = {
    Name = '${local.name_prefix}-public-${each.key}'
  }
}
```

✓ **BEST PRACTICE:** Multi-AZ via for_each, consistent tagging with locals

- ☑ **CONSOLIDATION:** VPC, IGW, Subnets, Route Tables live in `vpc.tf`.
- ☑ **SCALABILITY:** `for_each` simplifies multi-AZ deployments.
- ☑ **CONSISTENCY:** Tags reference `local.name_prefix` for uniform naming.

Storage.tf: Secure-by-Default S3 Configuration & Decoupled Naming



locals.tf (Computed Values)

```
locals {
  bucket_name =
"${var.app}-${var.env}-logs-...
}
```

storage.tf (S3 & Security Config)

1. S3 Bucket Resource (aws_s3_bucket)

```
resource "aws_s3_bucket" "logs" {
  bucket = local.bucket_name
}
```

2. Versioning (aws_s3_bucket_versioning)

```
versioning_configuration {
  status = "Enabled"
}
```

3. Server-Side Encryption (aws_s3_bucket_server_side_encryption_configuration)

```
rule {
  apply_server_side_encryption_by_default {
    sse_algorithm = "AES256"
  }
}
```

4. Public Access Block (aws_s3_bucket_public_access_block)

```
block_public_acls = true,
block_public_policy = true,
...
```

Decoupled & Secure S3 Resource

✓ Naming Sourced from Locals
✓ Independent Security Policies
✓ Evolvable & Maintainable

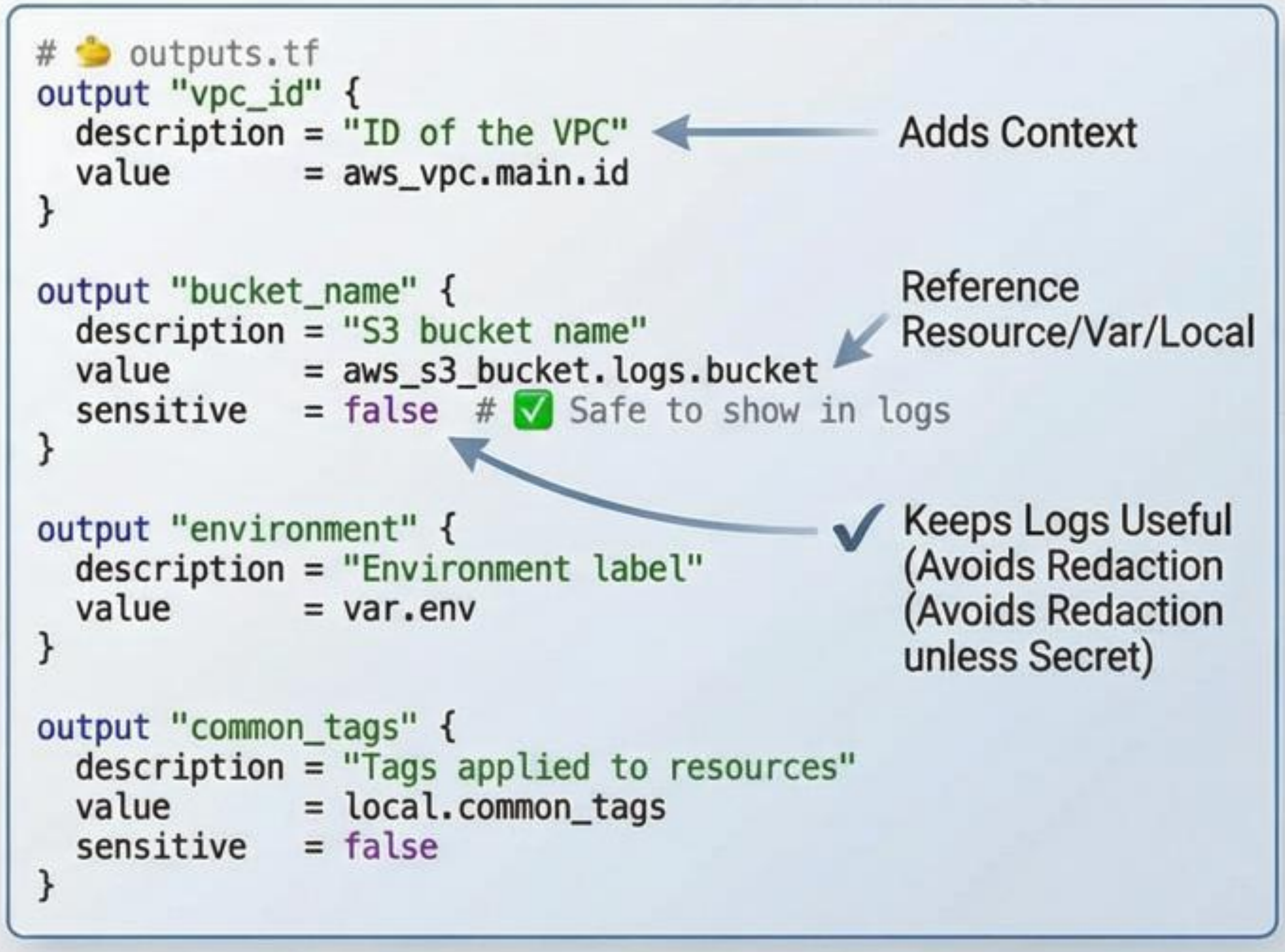
Key Benefit: Naming logic is separate from policy, encryption, and access settings, allowing independent evolution.

outputs.tf: Exposing Essential Values for Downstream Use

Purpose & Benefits

- **Purpose:** Expose key infrastructure data.
- **For Downstream:** CI/CD pipelines, other tools, human readability.
- **Clarity:** Descriptions clarify value purpose.
- **Security:** Use ‘sensitive’ sparingly.

Example: outputs.tf & Best Practice



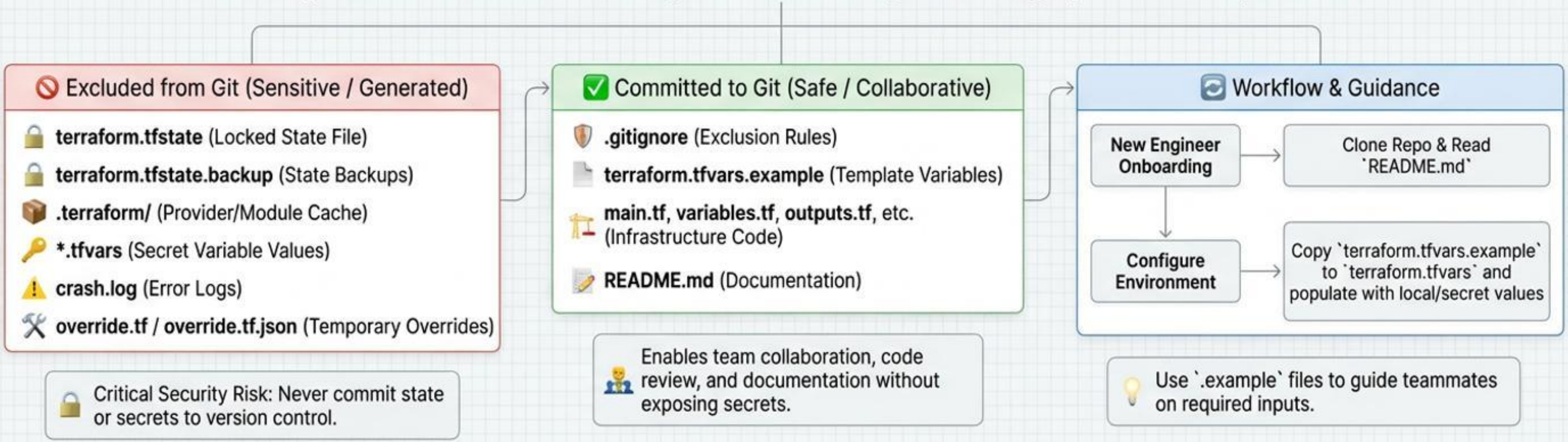
```
# outputs.tf
output "vpc_id" {
  description = "ID of the VPC"
  value       = aws_vpc.main.id
}

output "bucket_name" {
  description = "S3 bucket name"
  value       = aws_s3_bucket.logs.bucket
  sensitive   = false  # ✅ Safe to show in logs
}

output "environment" {
  description = "Environment label"
  value       = var.env
}

output "common_tags" {
  description = "Tags applied to resources"
  value       = local.common_tags
  sensitive   = false
}
```

Adds Context
Reference Resource/Var/Local
✓ Keeps Logs Useful (Avoids Redaction (Avoids Redaction unless Secret)

Pro Tip: Only mark outputs as `sensitive = true` if they contain actual secrets (e.g., database passwords, access keys) to prevent accidental exposure.

`.gitignore` & Git Hygiene: Secure Collaboration

Preventing accidental credential leakage and maintaining state integrity in shared repositories.



🚫 Excluded from Git (Sensitive / Generated)

- 🔒 **terraform.tfstate** (Locked State File)
- 🔒 **terraform.tfstate.backup** (State Backups)
- 📦 **.terraform/** (Provider/Module Cache)
- 🔑 ***.tfvars** (Secret Variable Values)
- ⚠️ **crash.log** (Error Logs)
- 🛠 **override.tf / override.tf.json** (Temporary Overrides)

🔒 Critical Security Risk: Never commit state or secrets to version control.

✅ Committed to Git (Safe / Collaborative)

- 🛡 **.gitignore** (Exclusion Rules)
- 📄 **terraform.tfvars.example** (Template Variables)
- 🏗 **main.tf, variables.tf, outputs.tf**, etc. (Infrastructure Code)
- 📝 **README.md** (Documentation)

🧑‍🤝‍🧑 Enables team collaboration, code review, and documentation without exposing secrets.

🔄 Workflow & Guidance

New Engineer Onboarding → Clone Repo & Read `README.md`

Configure Environment → Copy `terraform.tfvars.example` to `terraform.tfvars` and populate with local/secret values

💡 Use `.example` files to guide teammates on required inputs.

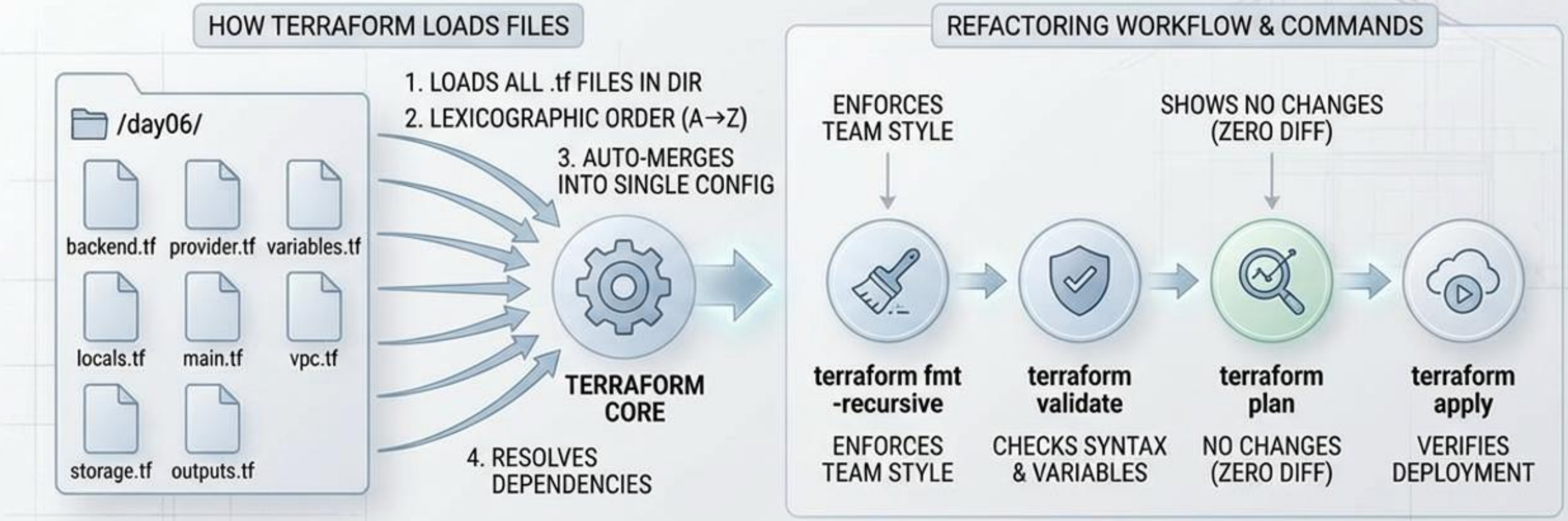
Sample `.gitignore` Snippet

```
# 🔒 Terraform State & Backups
terraform.tfstate
terraform.tfstate.backup
.terraform.tfstate.lock.info

# 📦 Provider & Module Cache
.terraform/
# 🔑 Secret Variable Files
*.tfvars.json

# ⚠️ Crash Logs
crash.log

# 🛠 Override Files
override.tf
override.tf.json

# 📄 Keep Example Files
!terraform.tfvars.example
```

Golden Rule: Commit templates (`.example`), ignore actual secrets (`.tfvars`), and secure state remotely.

DAY 6/28:

TERRAFORM FILE STRUCTURE & BEST PRACTICES



HOW TERRAFORM LOADS FILES

/day06/

backend.tf provider.tf variables.tf

locals.tf main.tf vpc.tf

storage.tf outputs.tf

1. LOADS ALL .tf FILES IN DIR
2. LEXICOGRAPHIC ORDER (A→Z)
3. AUTO-MERGES INTO SINGLE CONFIG
4. RESOLVES DEPENDENCIES

TERRAFORM CORE

REFACTORING WORKFLOW & COMMANDS

ENFORCES TEAM STYLE

SHOWS NO CHANGES (ZERO DIFF)

terraform fmt -recursive
ENFORCES TEAM STYLE

terraform validate
CHECKS SYNTAX & VARIABLES

terraform plan
NO CHANGES (ZERO DIFF)

terraform apply
VERIFIES DEPLOYMENT

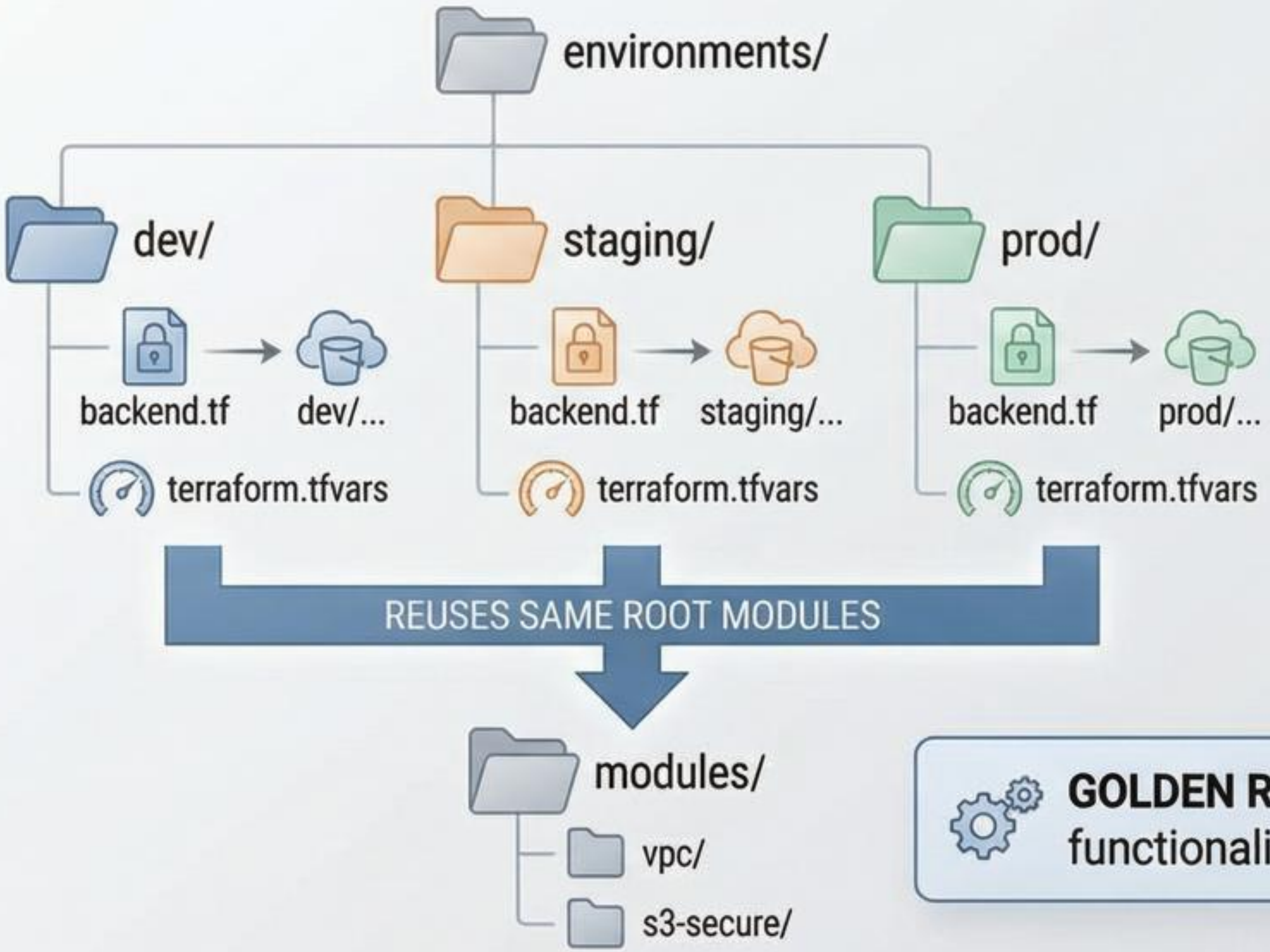
GOAL: CONFIRM REFACTOR IS IDENTICAL TO MONOLITH & TEAM-READY

READABILITY MAINTAINABILITY COLLABORATION GIT HYGIENE

TERRAFORM AWS LABS
github.com/Push/terraform-aws-labs/day06

ADVANCED PATTERNS: ENVIRONMENT-SPECIFIC DIRECTORIES

Isolating backends and variables per environment for maintainable scale.



TRADE-OFF & MITIGATION

✅ **PROS**: Isolated state, independent variables, clean separation.

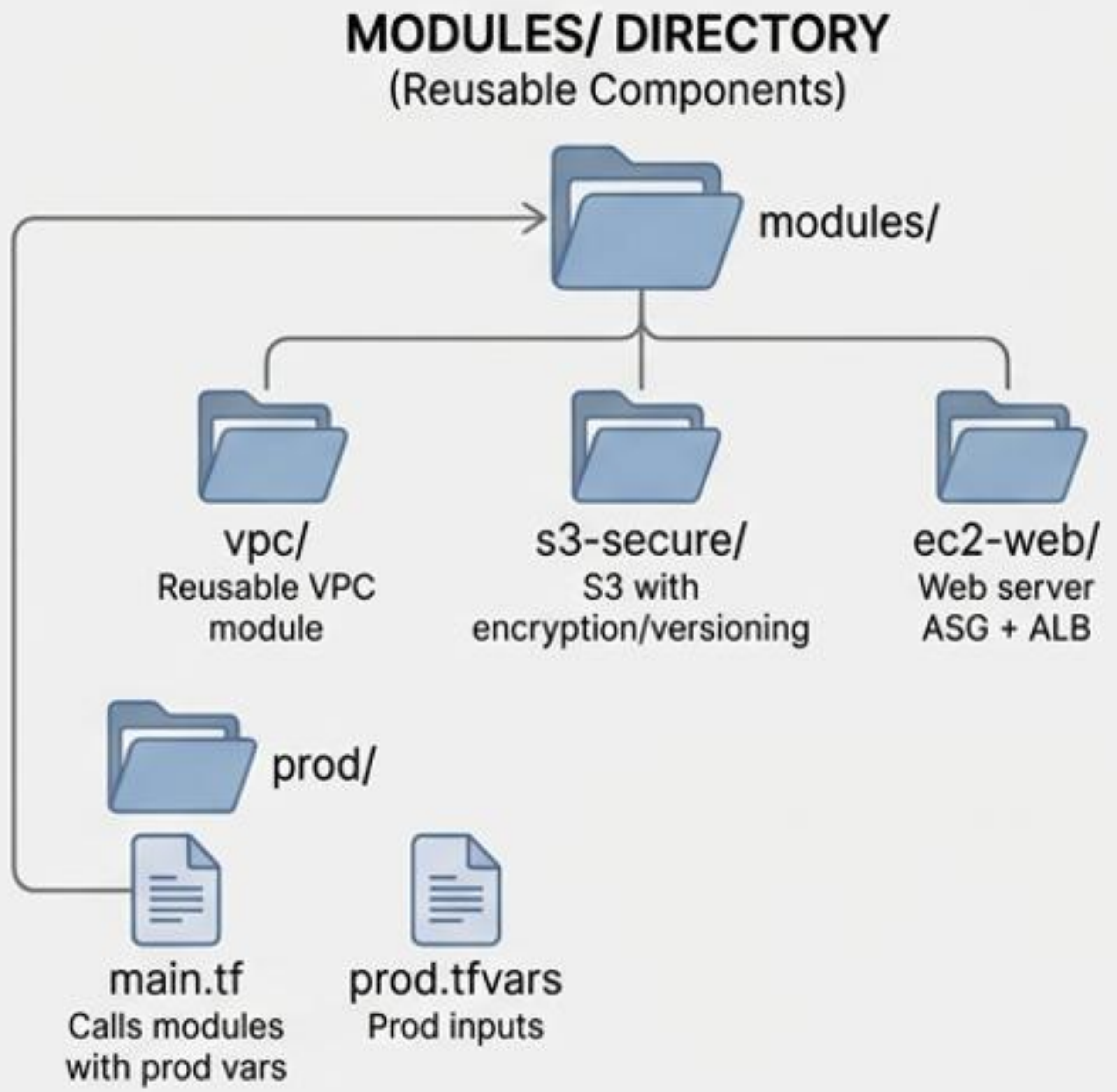
❌ **CONS**: Duplicated 'backend.tf' code.

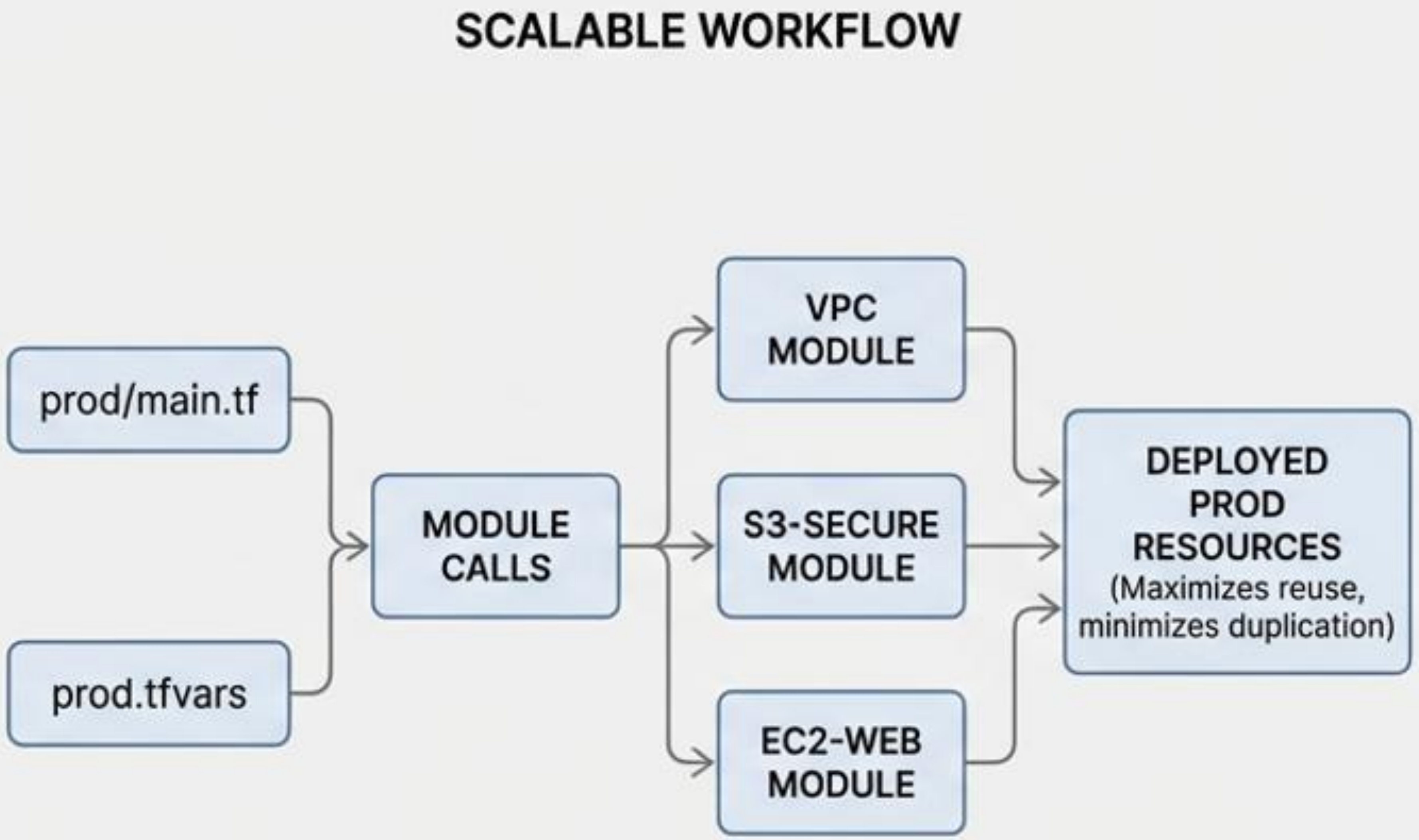
MITIGATION: Module composition once modules are introduced (Day 12+).

GOLDEN RULE: Structure for maintainability, not just functionality. If it's hard to understand, simplify it.

DAY 6/28: TERRAFORM FILE STRUCTURE & BEST PRACTICES

Advanced Patterns: Service-Based Modules (Scalable)





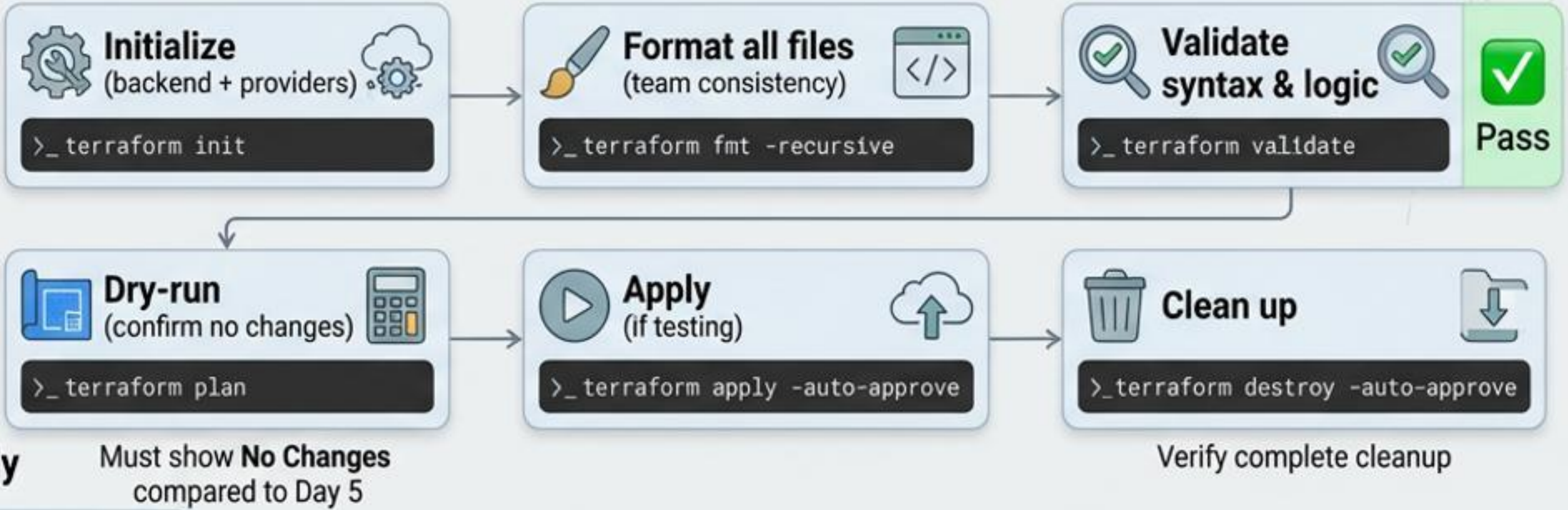
GitHub repo: https://github.com/aprod/main.tf

🧪 Lab: Validate & Test Your Structure

Goal: Verify Refactored Structure Behaves Identically to Monolithic Config

✅ Success Criteria

- ✓ **terraform validate** passes with no errors.
- ✓ **terraform plan** shows *identical* resources as Day 5 (no unexpected changes).
- ✓ **terraform destroy** cleans up all resources without errors.



Initialize (backend + providers)

```
>_ terraform init
```

Format all files (team consistency)

```
>_ terraform fmt -recursive
```

Validate syntax & logic

```
>_ terraform validate
```

✅ Pass

Dry-run (confirm no changes)

```
>_ terraform plan
```

Must show **No Changes** compared to Day 5

Apply (if testing)

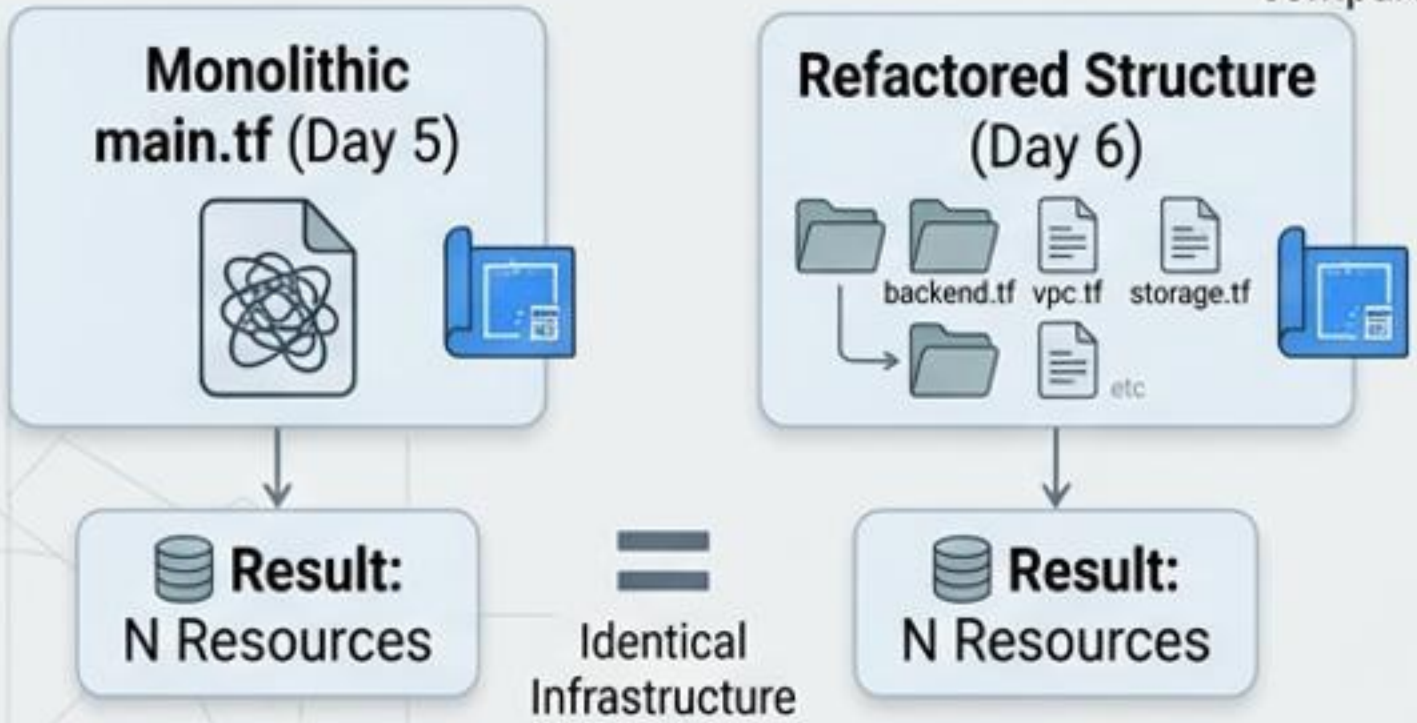
```
>_ terraform apply -auto-approve
```

Clean up

```
>_ terraform destroy -auto-approve
```

Verify complete cleanup

Proving the Split Files Behave Identically



Monolithic main.tf (Day 5)

Refactored Structure (Day 6)

Result: N Resources

= Identical Infrastructure

Result: N Resources

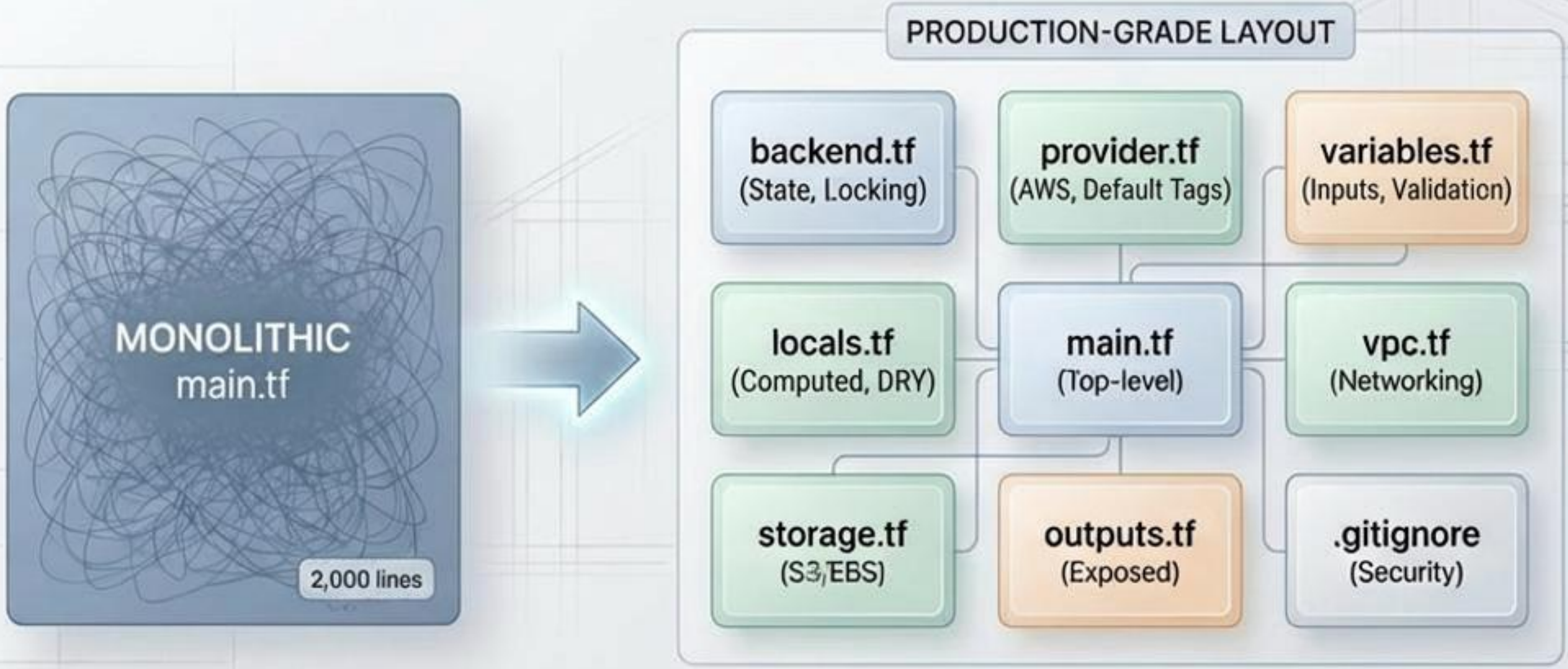
➡️ Summary: Structure = Maintainability

Anti-Pattern	Production Practice
❌ One main.tf	✅ Logical files (vpc.tf, storage.tf)
❌ Hardcoded values	✅ variables.tf + .tfvars
❌ Repeated tags	✅ locals.tf + default_tags

🎯 ***Golden Rule:*** *"If a new engineer can't understand your structure in 5 minutes — simplify it."*

DAY 6/28: TERRAFORM FILE STRUCTURE & BEST PRACTICES

🔗 **Repo:** github.com/Push/terraform-aws-labs/day06



STEP-BY-STEP REFACTOR

- ☑ Replace a single bloated main.tf with logical files.
- ☑ Replace hard-coded strings with validated variables.
- ☑ Replace copy-pasted tags with locals and default_tags.
- ☑ Replace committed state with remote backend and .gitignore.
- ☑ Replace silent failures with validation blocks to achieve production-grade maintainability.

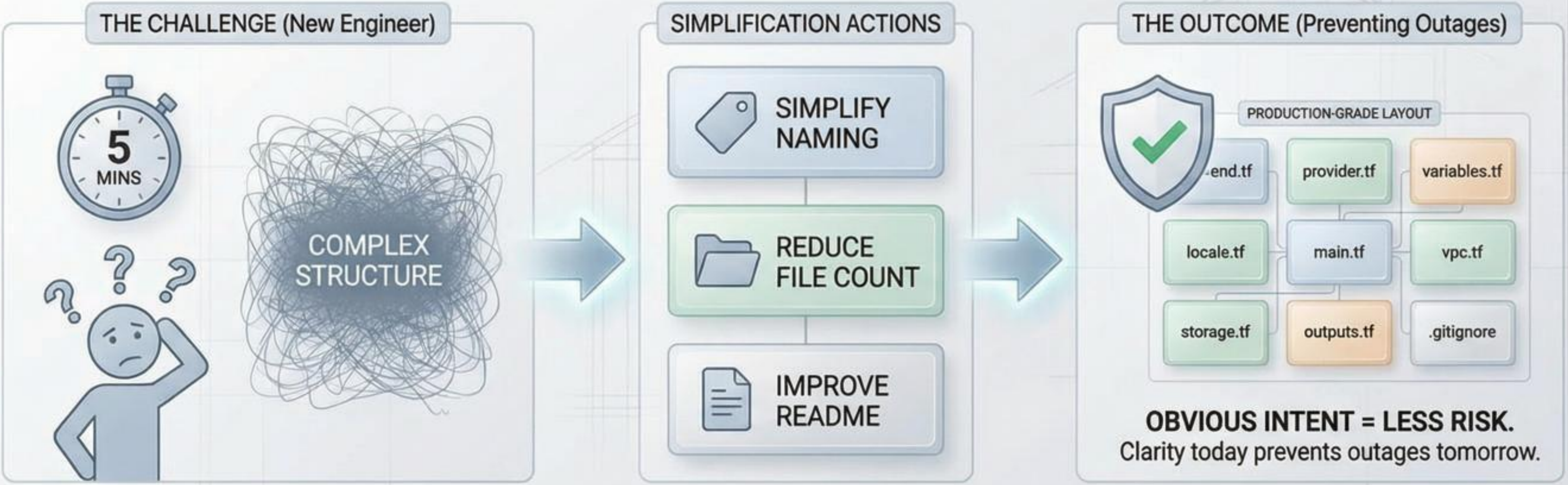
GOAL: From Monolithic to Organized, Maintainable, Team-Ready

READABILITY MAINTAINABILITY COLLABORATION GIT HYGIENE

TERRAFORM AWS LABS
github.com/Push/terraform-aws-labs/day06

THE 5-MINUTE RULE: CLARITY IS RELIABILITY

Goal: Ensure repository structure is immediately understandable for any engineer.



THE CHALLENGE (New Engineer)

SIMPLIFICATION ACTIONS

- SIMPLIFY NAMING
- REDUCE FILE COUNT
- IMPROVE README

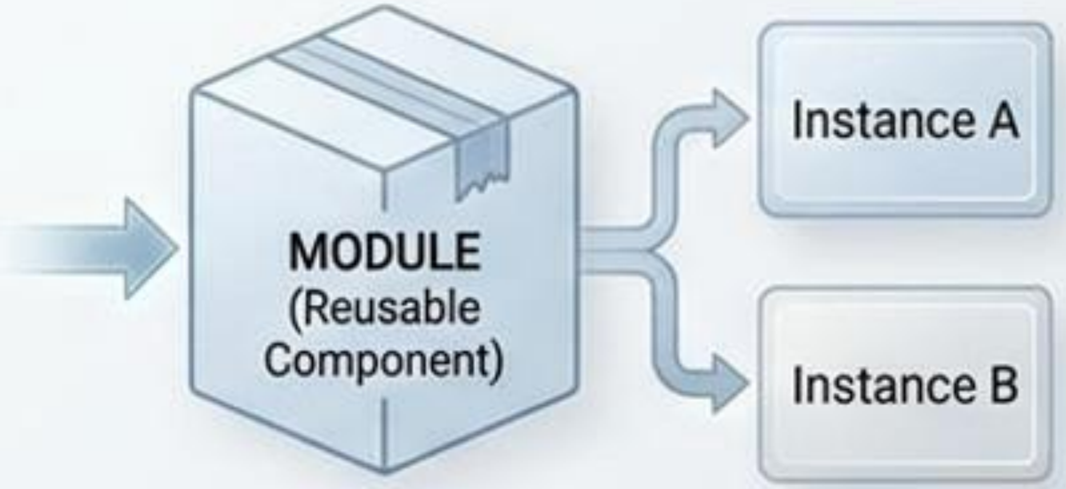
THE OUTCOME (Preventing Outages)

OBVIOUS INTENT = LESS RISK.
Clarity today prevents outages tomorrow.

READABILITY · MAINTAINABILITY · COLLABORATION · GIT HYGIENE

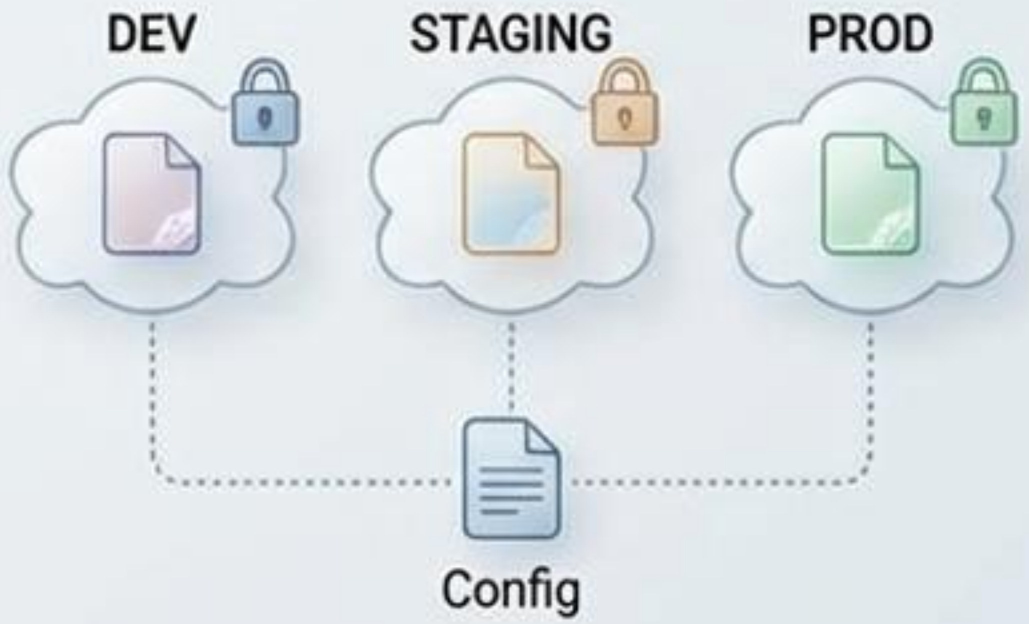
FUTURE SESSIONS:
Building on Foundations for Automation & Scale

MODULES
(Packaging for Reuse)



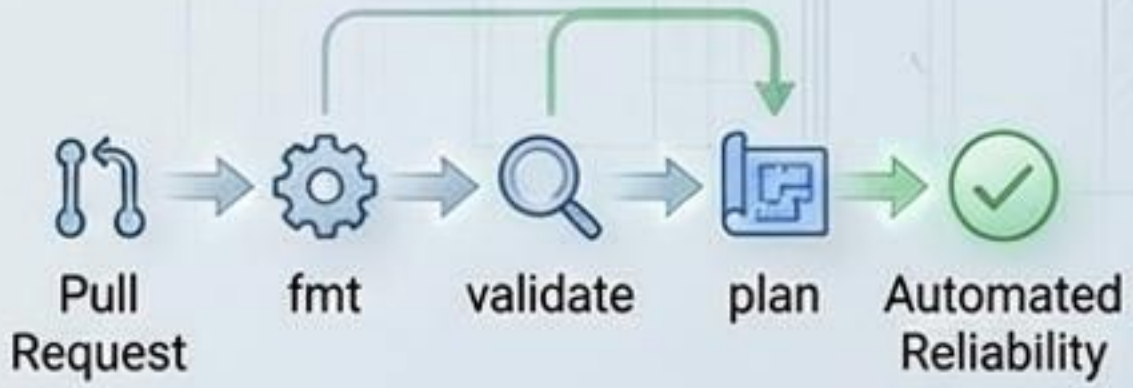
- Encapsulate resources into reusable components. Promotes DRY, consistency, and simplified management.

WORKSPACES
(State Isolation)



- Manage multiple environments with isolated state files from a single configuration. Prevents accidental cross-environment changes.

CI PIPELINES
(Enforced Reliability)



- Automate `fmt`, `validate`, and `plan` on every PR. Ensure code quality and prevent drift.

Turning clean code into automated reliability through modularity, isolation, and enforced workflows.