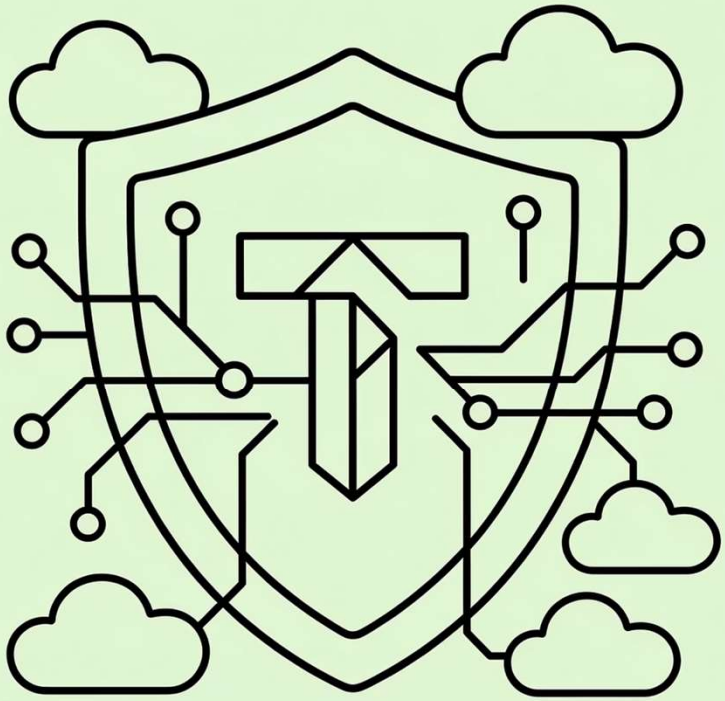
Static Analysis & Policy-as-Code (PaC)

Automating Compliance & Governance in Terraform CI/CD



Module 2

Static Analysis & Policy-as-Code

Automating Compliance & Governance in Terraform CI/CD

🎓 *MODULE 2 SLIDE STRUCTURE*

Module Title: Static Analysis & Policy-as-Code (PaC)

Subtitle: Automating Compliance & Governance in Terraform CI/CD

Title Slide

Title:

Module 2: Static Analysis & Policy-as-Code

Subtitle:

Automating Compliance & Governance in Terraform CI/CD

Footer:

Terraform Advanced CI/CD Series

Module Overview

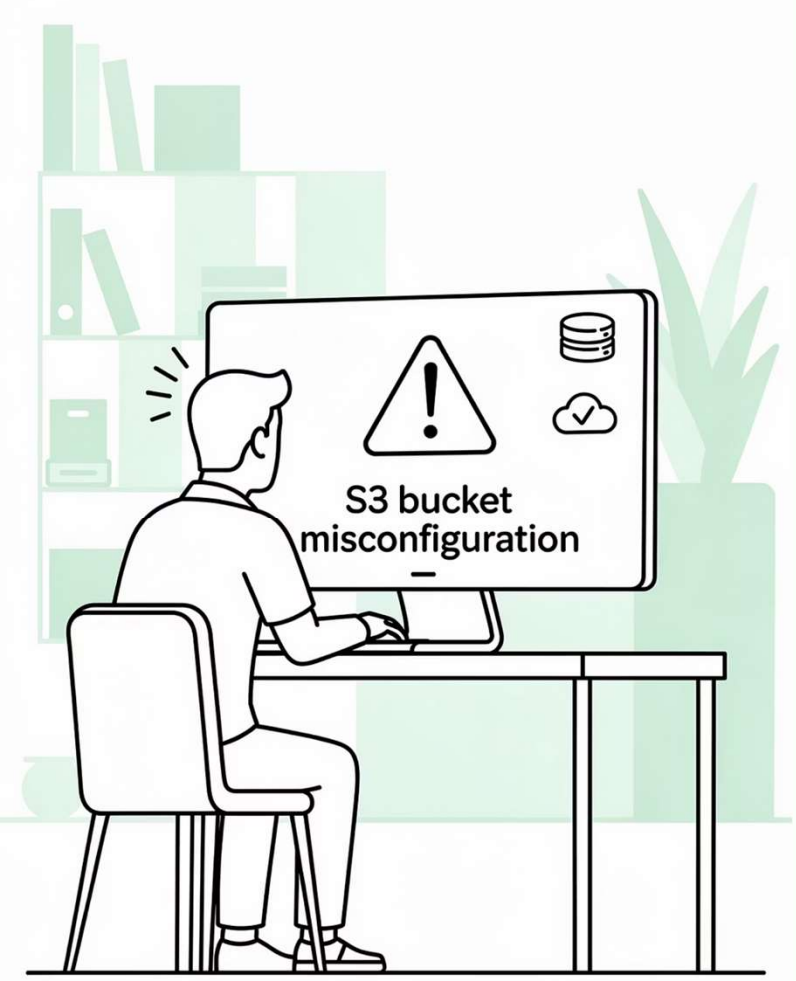
What This Module Covers

- Preventing bad infrastructure before deployment
- Static code validation tools
- Security misconfiguration detection
- Policy-as-Code governance
- CI/CD integration for automated enforcement

Learning Objectives

By the end of this module, you will:

1. Understand Terraform static analysis
2. Use tflint for linting
3. Use checkov for security scanning
4. Understand OPA & Sentinel policies
5. Enforce compliance in CI/CD pipelines



PROBLEM CONTEXT

Real-World Failure Scenario

A Developer Accidentally:

- Creates S3 bucket **without encryption**
- Leaves **public access open**
- Deploys in **unapproved region**
- Skips **mandatory cost tags**

Result: Security & compliance issues after deployment

Problems Without Static Validation

Issue	Impact
No pre-checks	Risks reach production
Manual reviews	Slow & inconsistent
Governance post-deploy	Expensive fixes
No enforcement	Drift across teams

How This Module Solves It

We introduce:

- ✓ *Static Analysis*
  Detect errors early
- ✓ *Security Scanning*
  Catch misconfigurations
- ✓ *Policy-as-Code*
  Enforce governance rules
- ✓ *CI/CD Gates*
  Stop bad PRs automatically

🔧 *STATIC ANALYSIS TOOLS*

What is Static Analysis?

Definition:

Checking Terraform code *without deploying resources*

Benefits:

- Faster feedback
- Prevents security mistakes
- Improves code quality
- Enables automation

TFLINT SECTION

tflint Overview

Purpose: Terraform linting tool

Detects:

- *Syntax issues*
- *Deprecated arguments*
- *Unused variables*
- *Provider best-practice violations*

When to Use tflint

1	2	3
✓ *Before every commit*	✓ *In CI pipeline*	✓ *During PR validation*

Command:

```
tflint
```

tflint Example

```
resource "aws_instance" "web" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}
```

Possible Findings

- Invalid AMI
- Deprecated instance type
- Region mismatch

🔐 *CHECKOV SECTION*

checkov Overview

Purpose: Terraform security scanner

Detects:

- Open security groups
- Public S3 buckets
- Missing encryption
- IAM wildcard permissions

Running checkov

```
checkov -d .
```

Scans entire Terraform directory for risks.

checkov Example Finding

```
resource "aws_s3_bucket" "data" {
  bucket = "my-data-bucket"
}
```

Issues Found:

- ❌ Encryption missing
- ❌ Versioning not enabled
- ❌ Public access risk

POLICY-AS-CODE

What is Policy-as-Code?

Definition: Writing governance rules as code

Goal: Automatically enforce compliance

Tool	Platform	Use Case
OPA	Open-source	CI/CD pipelines
Sentinel	Terraform Cloud	Enterprise governance

Governance Rules Examples

Rule	Purpose
Mandatory encryption	Security compliance
Required tags	Cost allocation
Region restriction	Regulatory control
Instance size limits	Cost optimization

Example OPA Policy

```
package terraform.security

deny[msg] {
 input.resource_type == "aws_s3_bucket"
 not input.config.server_side_encryption_configuration
 msg = "S3 bucket must have encryption enabled"
}
```