📘 Module 2: Static Analysis & Policy-as-Code (PaC)

Subtitle: Automating Compliance and Governance in Terraform CI/CD Pipelines

This module focuses on implementing automated static validation and governance controls in Terraform workflows. You will learn how to detect misconfigurations, security risks, and policy violations *before infrastructure is deployed* by integrating scanning and policy tools into CI/CD pipelines.

🎯 Learning Objectives

By the end of this module, you will understand:

- How static analysis tools improve Terraform code quality and security
- How to use **tflint** for best-practice and syntax validation
- How to use **checkov** for security and compliance scanning
- The fundamentals of **Policy-as-Code (PaC)** using **OPA** and **Sentinel**
- How to enforce organizational rules like encryption, tagging, and region control
- How to integrate all these checks into Jenkins CI/CD pipelines and PR workflows

---

🤔 Why Do We Need This?

Real-World Scenario

An organization deploys infrastructure using Terraform. A developer accidentally:

- Creates an S3 bucket without encryption
- Allows public access
- Deploys resources in an unapproved region
- Omits required cost-center tags

The code deploys successfully, but security and governance teams discover violations later — causing rework, outages, and compliance risks.

Problem Without Static Validation

Issue	Impact
No pre-checks	Security risks reach production
Manual reviews	Slow and inconsistent
Governance only after deployment	Expensive remediation
No standard enforcement	Drift across teams

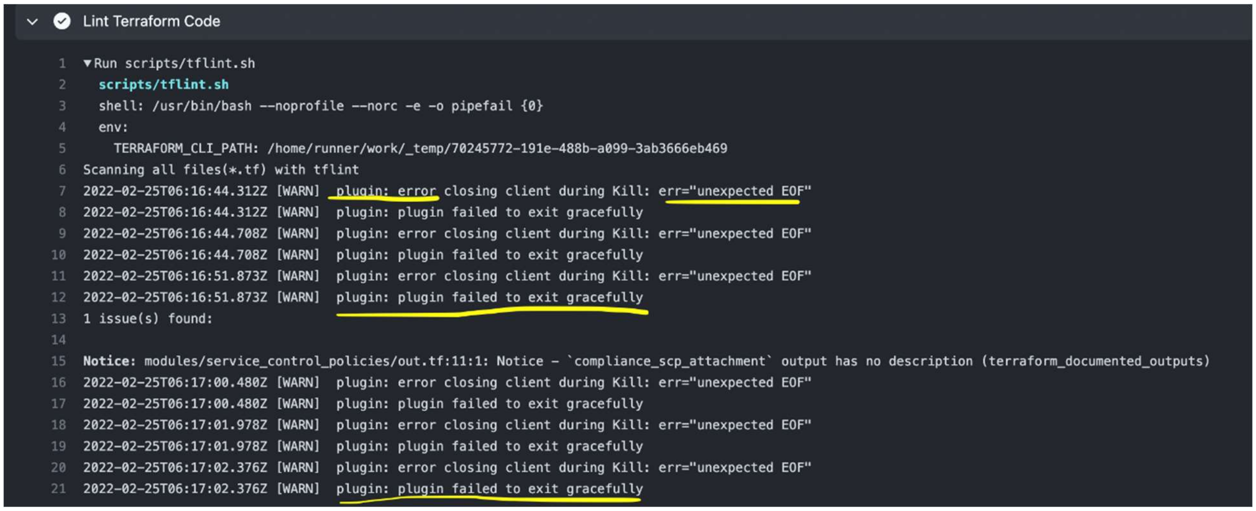
How This Module Solves It

We introduce:

- **Static Analysis** → Detect errors before deployment
- **Security Scanning** → Catch misconfigurations
- **Policy-as-Code** → Enforce governance rules automatically
- **Pipeline Integration** → Stop bad code at PR stage

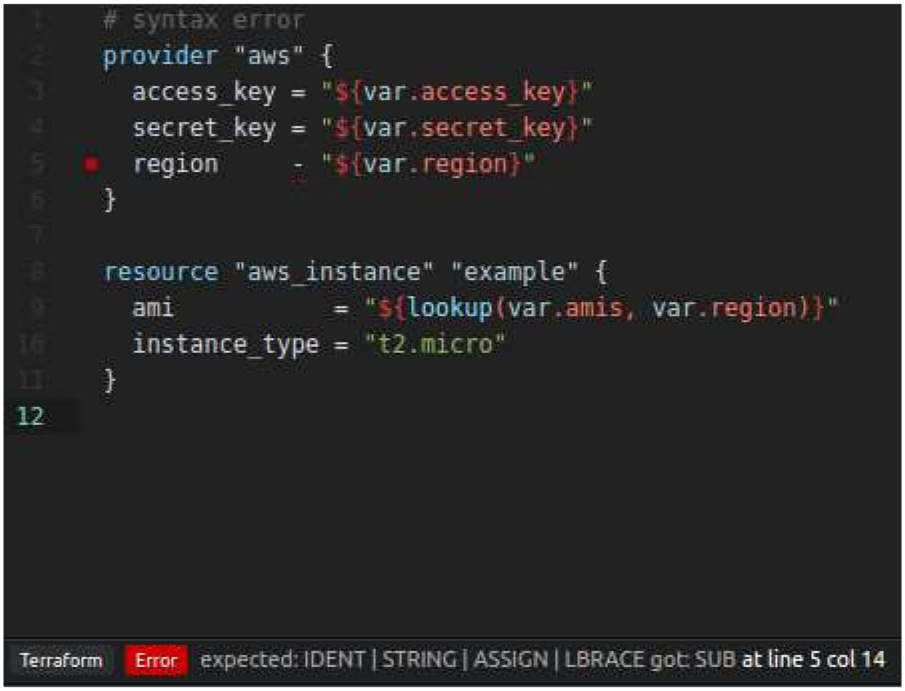
🧠 Concept Deep Dive

🔹 tflint – Terraform Linting Tool



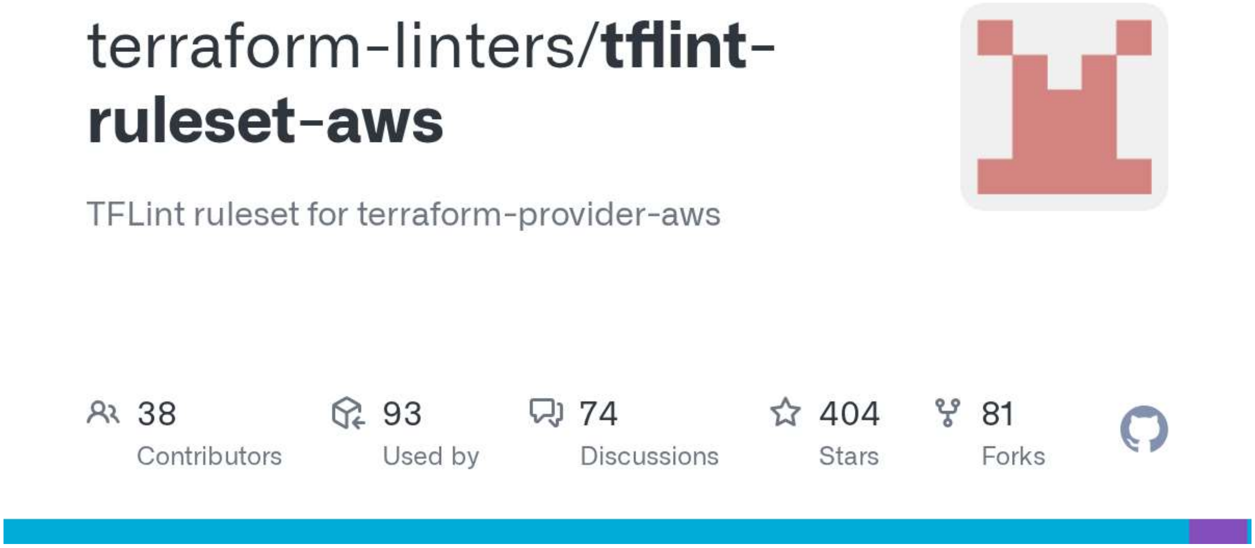
Lint Terraform Code

```
▼Run scripts/tflint.sh
  scripts/tflint.sh
  shell: /usr/bin/bash --noprofile --norc -e -o pipefail {0}
  env:
    TERRAFORM_CLI_PATH: /home/runner/work/_temp/70245772-191e-488b-a099-3ab3666eb469
Scanning all files(*.tf) with tflint
2022-02-25T06:16:44.312Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:16:44.312Z [WARN]  plugin: plugin failed to exit gracefully
2022-02-25T06:16:44.708Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:16:44.708Z [WARN]  plugin: plugin failed to exit gracefully
2022-02-25T06:16:51.873Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:16:51.873Z [WARN]  plugin: plugin failed to exit gracefully
1 issue(s) found:

Notice: modules/service_control_policies/out.tf:11:1: Notice - `compliance_scp_attachment` output has no description (terraform_documented_outputs)
2022-02-25T06:17:00.480Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:17:00.480Z [WARN]  plugin: plugin failed to exit gracefully
2022-02-25T06:17:01.978Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:17:01.978Z [WARN]  plugin: plugin failed to exit gracefully
2022-02-25T06:17:02.376Z [WARN]  plugin: error closing client during Kill: err="unexpected EOF"
2022-02-25T06:17:02.376Z [WARN]  plugin: plugin failed to exit gracefully
```



```
# syntax error
provider "aws" {
  access_key = "${var.access_key}"
  secret_key = "${var.secret_key}"
  region     - "${var.region}"
}

resource "aws_instance" "example" {
  ami           = "${lookup(var.amis, var.region)}"
  instance_type = "t2.micro"
}

```

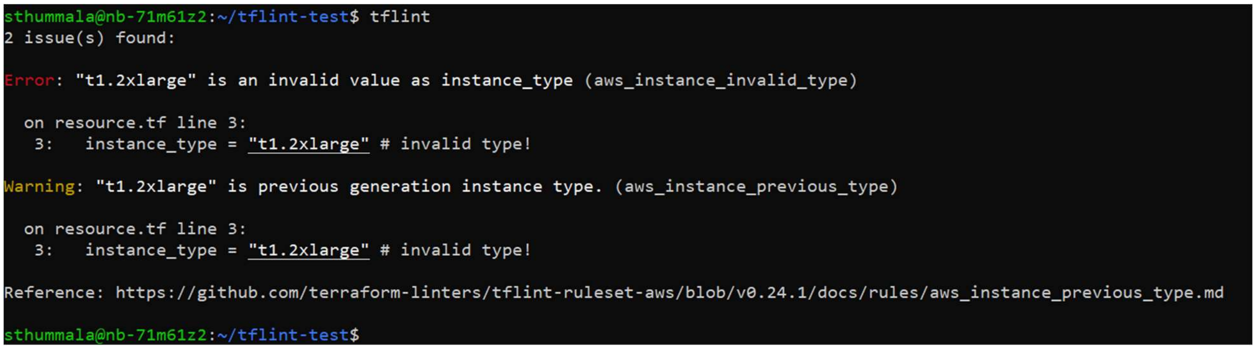
Terraform Error expected: IDENT | STRING | ASSIGN | LBRACE got: SUB at line 5 col 14



terraform-linters/**tflint-ruleset-aws**

TFLint ruleset for terraform-provider-aws

38 Contributors · 93 Used by · 74 Discussions · 404 Stars · 81 Forks



```
sthummala@nb-71m61z2:~/tflint-test$ tflint
2 issue(s) found:

Error: "t1.2xlarge" is an invalid value as instance_type (aws_instance_invalid_type)

  on resource.tf line 3:
   3:    instance_type = "t1.2xlarge" # invalid type!

Warning: "t1.2xlarge" is previous generation instance type. (aws_instance_previous_type)

  on resource.tf line 3:
   3:    instance_type = "t1.2xlarge" # invalid type!

Reference: https://github.com/terraform-linters/tflint-ruleset-aws/blob/v0.24.1/docs/rules/aws_instance_previous_type.md

sthummala@nb-71m61z2:~/tflint-test$
```

Explanation

TFLint is a Terraform linter that detects:

- Syntax mistakes
- Deprecated arguments
- Unused variables
- Provider best practice violations

When to Use

✓ Before every commit
✓ Inside CI pipeline
✓ During PR validation

Syntax

```
tflint
```

Example

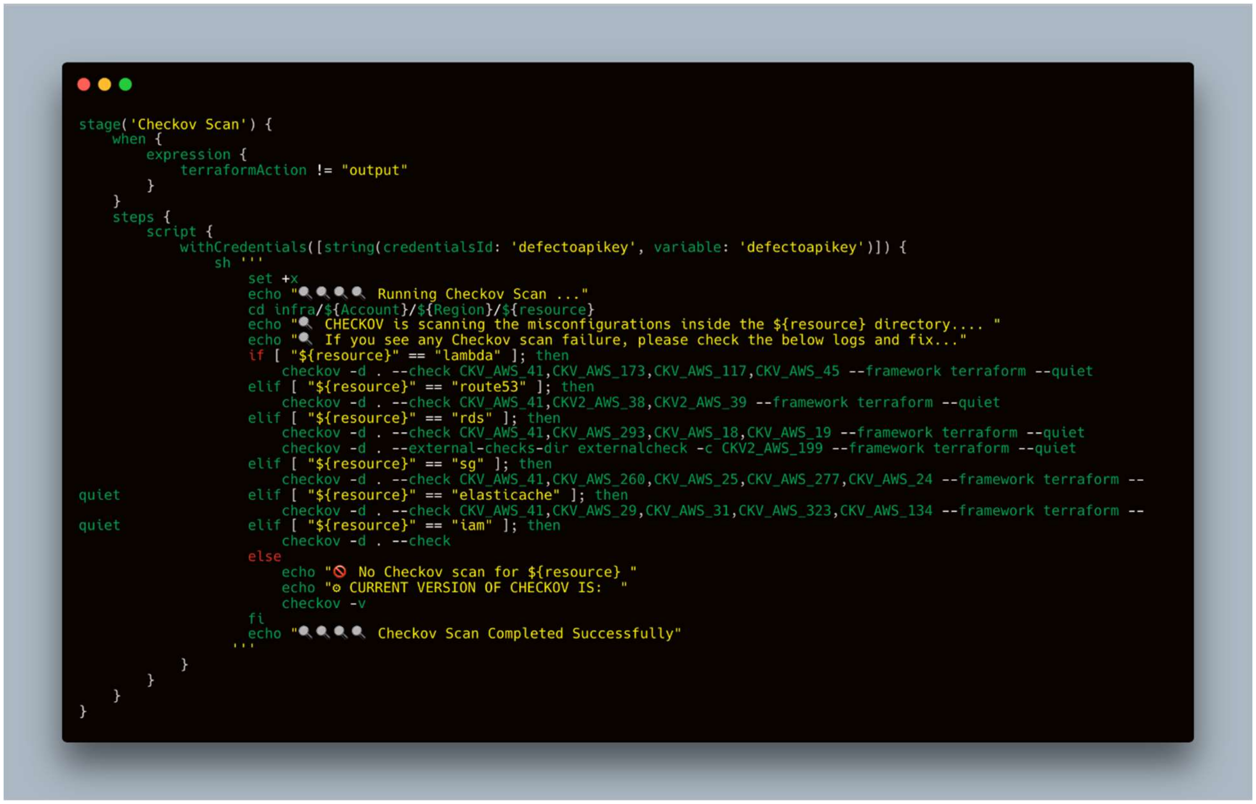
```
resource "aws_instance" "web" {

  ami           = "ami-12345678"

  instance_type = "t2.micro"

}
```

If the AMI is invalid or outdated → tflint flags it.

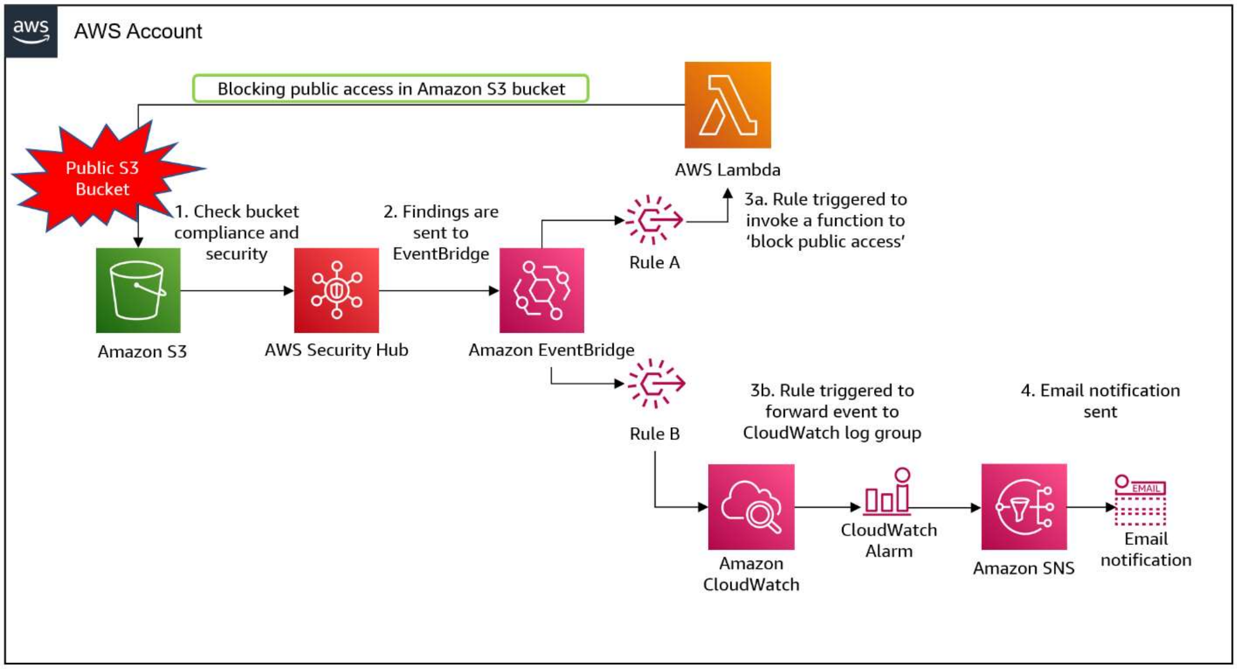
Output

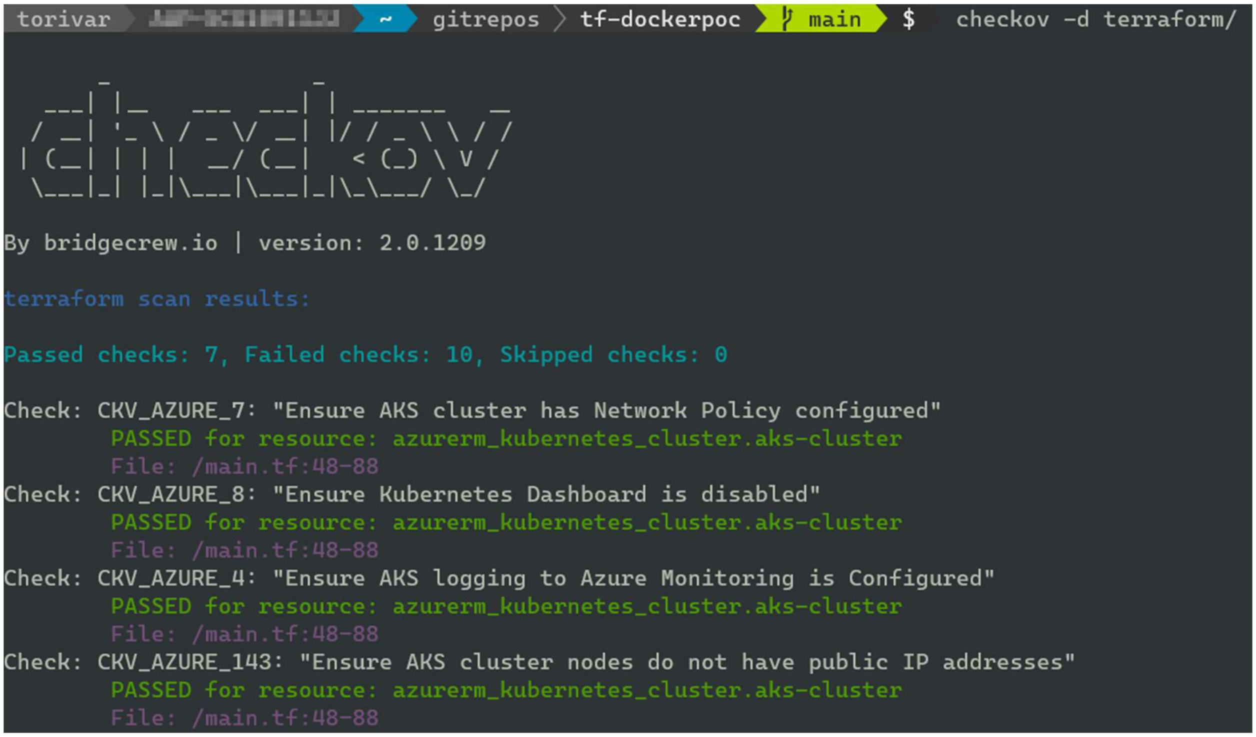
Shows rule violations and suggested fixes.

◆ checkov – Static Security Scanner

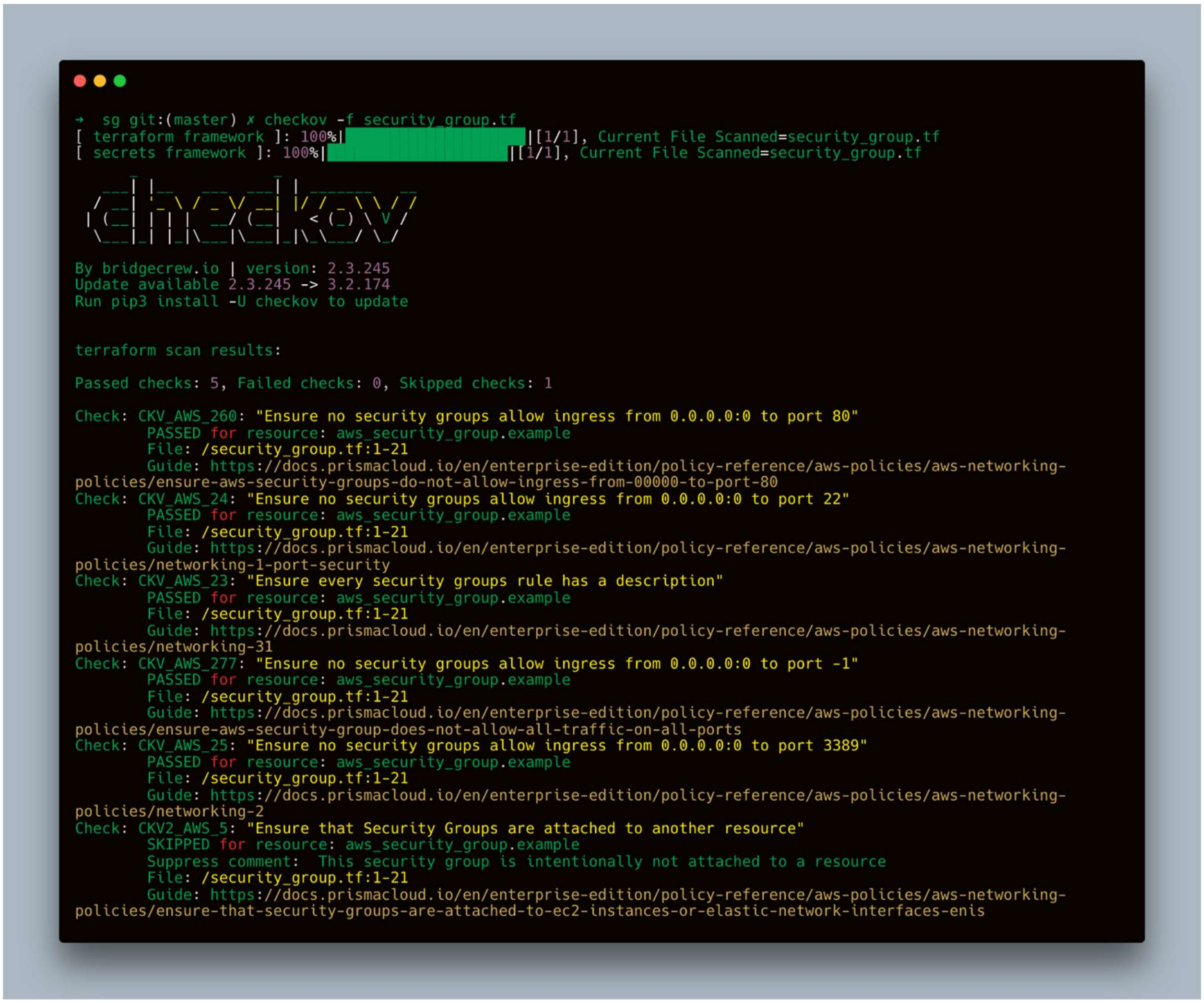


```
stage('Checkov Scan') {
    when {
        expression {
            terraformAction != "output"
        }
    }
    steps {
        script {
            withCredentials([string(credentialsId: 'defectoapikey', variable: 'defectoapikey')]) {
                sh '''
                    set +x
                    echo "🔍🔍🔍🔍 Running Checkov Scan ..."
                    cd infra/${Account}/${Region}/${resource}
                    echo "🔍 CHECKOV is scanning the misconfigurations inside the ${resource} directory.... "
                    echo "🔍 If you see any Checkov scan failure, please check the below logs and fix..."
                    if [ "${resource}" == "lambda" ]; then
                        checkov -d . --check CKV_AWS_41,CKV_AWS_173,CKV_AWS_117,CKV_AWS_45 --framework terraform --quiet
                    elif [ "${resource}" == "route53" ]; then
                        checkov -d . --check CKV_AWS_41,CKV2_AWS_38,CKV2_AWS_39 --framework terraform --quiet
                    elif [ "${resource}" == "rds" ]; then
                        checkov -d . --check CKV_AWS_41,CKV_AWS_293,CKV_AWS_18,CKV_AWS_19 --framework terraform --quiet
                        checkov -d . --external-checks-dir externalcheck -c CKV2_AWS_199 --framework terraform --quiet
                    elif [ "${resource}" == "sg" ]; then
                        checkov -d . --check CKV_AWS_41,CKV_AWS_260,CKV_AWS_25,CKV_AWS_277,CKV_AWS_24 --framework terraform --
quiet               elif [ "${resource}" == "elasticache" ]; then
                        checkov -d . --check CKV_AWS_41,CKV_AWS_29,CKV_AWS_31,CKV_AWS_323,CKV_AWS_134 --framework terraform --
quiet               elif [ "${resource}" == "iam" ]; then
                        checkov -d . --check
                    else
                        echo "🚫 No Checkov scan for ${resource} "
                        echo "⚙ CURRENT VERSION OF CHECKOV IS:  "
                        checkov -v
                    fi
                    echo "🔍🔍🔍🔍 Checkov Scan Completed Successfully"
                '''
            }
        }
    }
}
```


aws
AWS Account
Blocking public access in Amazon S3 bucket
AWS Lambda
Public S3 Bucket
1. Check bucket compliance and security
2. Findings are sent to EventBridge
Rule A
3a. Rule triggered to invoke a function to 'block public access'
Amazon S3
AWS Security Hub
Amazon EventBridge
Rule B
3b. Rule triggered to forward event to CloudWatch log group
4. Email notification sent
EMAIL
Amazon CloudWatch
CloudWatch Alarm
Amazon SNS
Email notification



```
torivar [redacted] ~ gitrepos > tf-dockerpoc  main  $  checkov -d terraform/

       _               _
   ___| |__   ___  ___| | _______   __
  / __| '_ \ / _ \/ __| |/ / _ \ \ / /
 | (__| | | |  __/ (__|   < (_) \ V /
  \___|_| |_|\___|\___|_|\_\___/ \_/

By bridgecrew.io | version: 2.0.1209

terraform scan results:

Passed checks: 7, Failed checks: 10, Skipped checks: 0

Check: CKV_AZURE_7: "Ensure AKS cluster has Network Policy configured"
        PASSED for resource: azurerm_kubernetes_cluster.aks-cluster
        File: /main.tf:48-88
Check: CKV_AZURE_8: "Ensure Kubernetes Dashboard is disabled"
        PASSED for resource: azurerm_kubernetes_cluster.aks-cluster
        File: /main.tf:48-88
Check: CKV_AZURE_4: "Ensure AKS logging to Azure Monitoring is Configured"
        PASSED for resource: azurerm_kubernetes_cluster.aks-cluster
        File: /main.tf:48-88
Check: CKV_AZURE_143: "Ensure AKS cluster nodes do not have public IP addresses"
        PASSED for resource: azurerm_kubernetes_cluster.aks-cluster
        File: /main.tf:48-88
```
]



```
➜  sg git:(master) ✗ checkov -f security_group.tf
[ terraform framework ]: 100%|████████████████████|[1/1], Current File Scanned=security_group.tf
[ secrets framework ]: 100%|████████████████████|[1/1], Current File Scanned=security_group.tf

       _               _
   ___| |__   ___  ___| | _______   __
  / __| '_ \ / _ \/ __| |/ / _ \ \ / /
 | (__| | | |  __/ (__|   < (_) \ V /
  \___|_| |_|\___|\___|_|\_\___/ \_/

By bridgecrew.io | version: 2.3.245
Update available 2.3.245 -> 3.2.174
Run pip3 install -U checkov to update


terraform scan results:

Passed checks: 5, Failed checks: 0, Skipped checks: 1

Check: CKV_AWS_260: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 80"
        PASSED for resource: aws_security_group.example
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/ensure-aws-security-groups-do-not-allow-ingress-from-00000-to-port-80
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
        PASSED for resource: aws_security_group.example
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/networking-1-port-security
Check: CKV_AWS_23: "Ensure every security groups rule has a description"
        PASSED for resource: aws_security_group.example
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/networking-31
Check: CKV_AWS_277: "Ensure no security groups allow ingress from 0.0.0.0:0 to port -1"
        PASSED for resource: aws_security_group.example
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/ensure-aws-security-group-does-not-allow-all-traffic-on-all-ports
Check: CKV_AWS_25: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 3389"
        PASSED for resource: aws_security_group.example
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/networking-2
Check: CKV2_AWS_5: "Ensure that Security Groups are attached to another resource"
        SKIPPED for resource: aws_security_group.example
        Suppress comment:  This security group is intentionally not attached to a resource
        File: /security_group.tf:1-21
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-
policies/ensure-that-security-groups-are-attached-to-ec2-instances-or-elastic-network-interfaces-enis
```

Explanation

Checkov scans Terraform for:

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

Syntax

checkov -d .

Example Issue

resource "aws_s3_bucket" "data" {

```
  bucket = "my-data-bucket"
}
```

Checkov flags:

- Encryption missing
- Versioning not enabled
- Public access risk

🔹 Policy-as-Code (PaC)

Policy-as-Code means **writing governance rules as code** and enforcing them automatically.

Two major tools:

Tool	Platform	Use Case
Open Policy Agent (OPA)	Open-source	Works with CI pipelines
HashiCorp Sentinel	Terraform Cloud/Enterprise	Enterprise governance

---

🔹 Example OPA Policy (Mandatory Encryption)

```
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"
}
```

This blocks unencrypted S3 buckets.

🔹 Example Governance Rules

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

🛠 Practical Example: Terraform + Scanning Pipeline

```
# Step 1: Terraform validation
terraform validate


# Step 2: Linting
tflint


# Step 3: Security Scan
checkov -d .


# Step 4: Policy Enforcement (OPA example)
opa eval --input terraform-plan.json --data policy.rego "data.terraform.security.deny"
```

🔚 Step-by-Step Breakdown

Let's break this down step by step:

1. Developer raises a Pull Request
2. Jenkins pipeline starts automatically
3. Terraform syntax is validated
4. tflint checks best practices

5. checkov scans for security risks
6. OPA/Sentinel evaluates governance policies
7. Pipeline fails if violations exist
8. Developer fixes issues before merge

🧩 Combining with Other Terraform Concepts

PaC works with:

- Variables (enforcing tag variables)
- Modules (governance at module level)
- for_each (ensuring all resources comply)
- CI pipelines (PR validation gates)

⚠️ Common Mistakes / Gotchas

- Ignoring scan warnings in CI
- Running scans only locally
- Not updating policy rules
- Overly strict policies blocking innovation
- Not version-controlling policies

📌 Key Takeaways

- Static analysis prevents bad Terraform from reaching production
- tflint ensures code quality
- checkov enforces security best practices
- OPA and Sentinel enforce governance
- CI/CD integration enables automated compliance
- Policy-as-Code scales governance across teams

🧪 Knowledge Check (MCQs)

1. What is the main purpose of tflint?

A. Deploy Terraform

B. Monitor AWS resources

C. Lint Terraform for best practices

D. Encrypt state files

2. Which tool detects security misconfigurations?

A. tflint

B. checkov

C. Sentinel

D. Terraform fmt

3. What does Policy-as-Code enable?

A. Faster Terraform apply

B. Automated governance enforcement

C. State file encryption

D. Module versioning

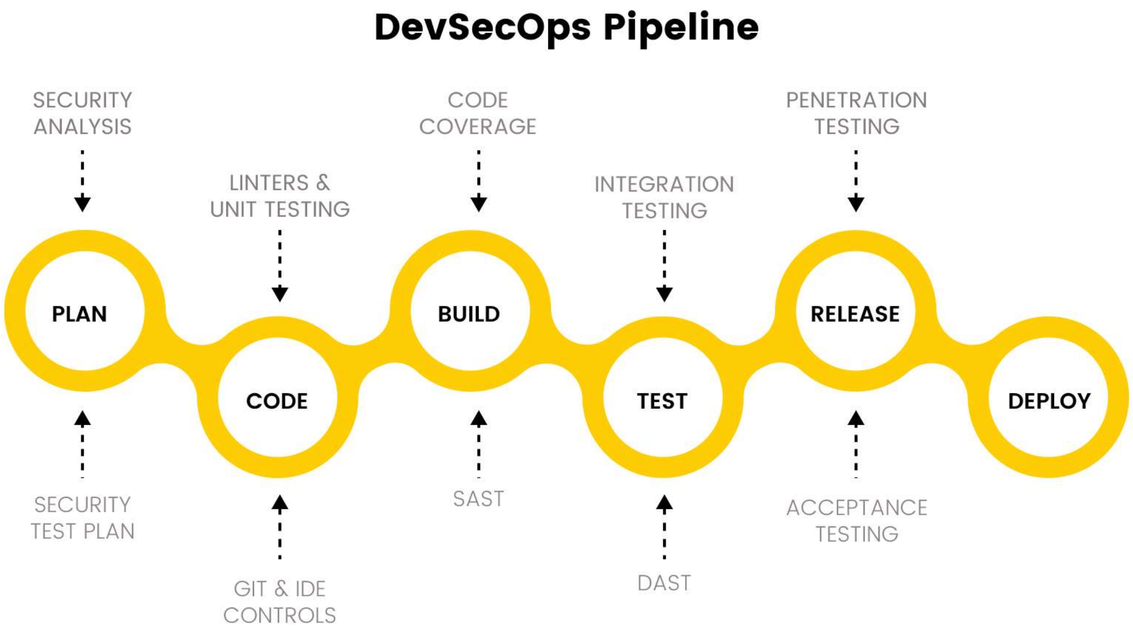
✅ Answers

1 → C (Linting tool)

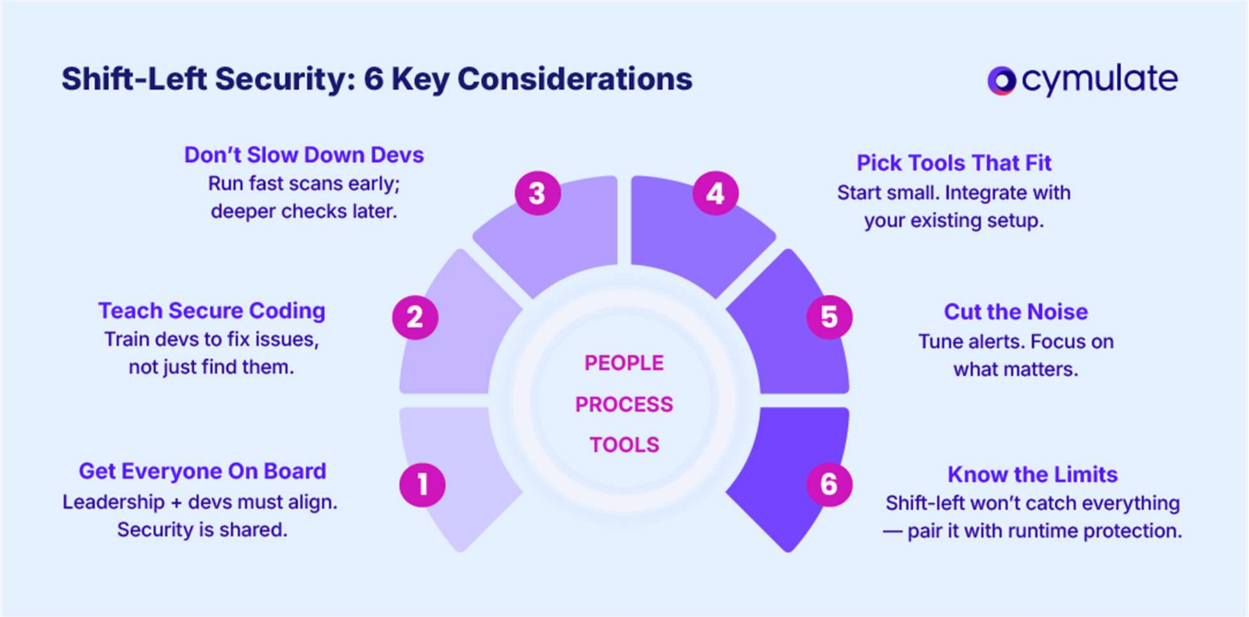
2 → B (Security scanning)

3 → B (Governance enforcement)

🧠 14. Static Analysis Maturity Model



DevSecOps Pipeline



Shift-Left Security: 6 Key Considerations

1 Get Everyone On Board
Leadership + devs must align. Security is shared.

2 Teach Secure Coding
Train devs to fix issues, not just find them.

3 Don't Slow Down Devs
Run fast scans early; deeper checks later.

4 Pick Tools That Fit
Start small. Integrate with your existing setup.

5 Cut the Noise
Tune alerts. Focus on what matters.

6 Know the Limits
Shift-left won't catch everything — pair it with runtime protection.

Level	Stage	Description
Level 1	Manual Review	Humans check Terraform code
Level 2	Basic Linting	tflint validates syntax and standards

Level	Stage	Description
Level 3	Security Scanning	checkov scans IaC for vulnerabilities
Level 4	Policy Enforcement	OPA/Sentinel block violations
Level 5	Continuous Governance	Org-wide policies enforced automatically

Key Insight: Advanced Terraform practices operate at **Level 4 or higher**.

🧩 15. Advanced tflint Usage

Custom Rule Configuration

tflint.hcl

```
plugin "aws" {
  enabled = true
  version = "0.29.0"
  source  = "github.com/terraform-linters/tflint-ruleset-aws"
}

rule "aws_instance_invalid_type" {
  enabled = true
}

rule "terraform_unused_declarations" {
  enabled = true
}
```

Why This Matters

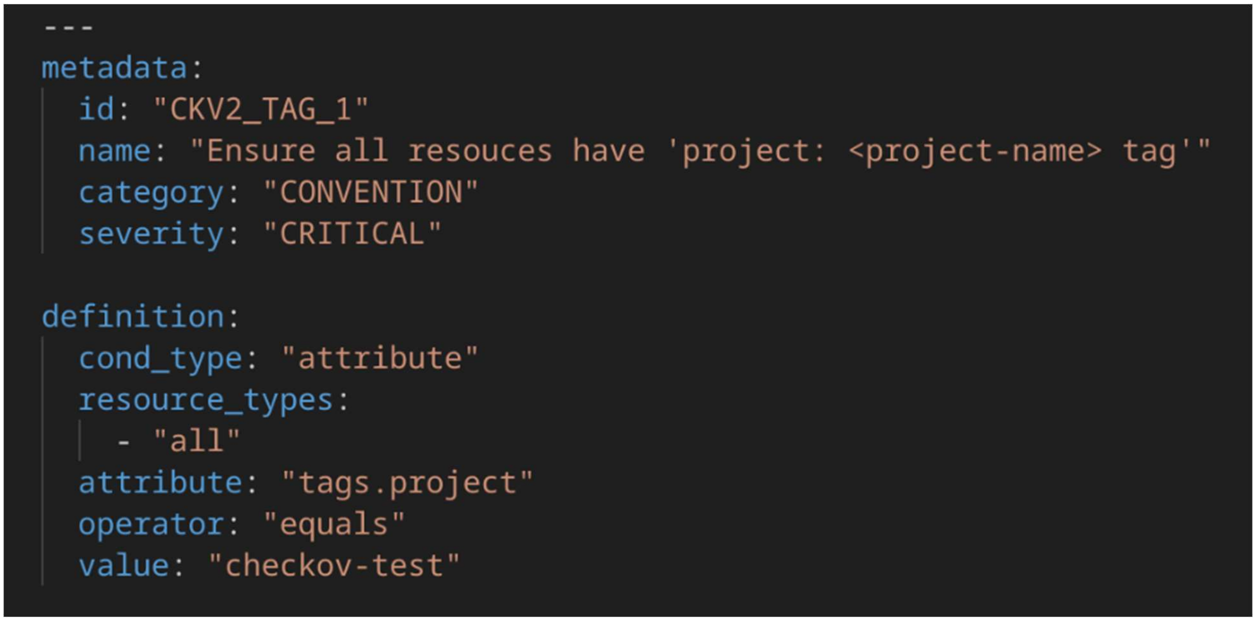
Enterprises standardize:

- Approved instance families

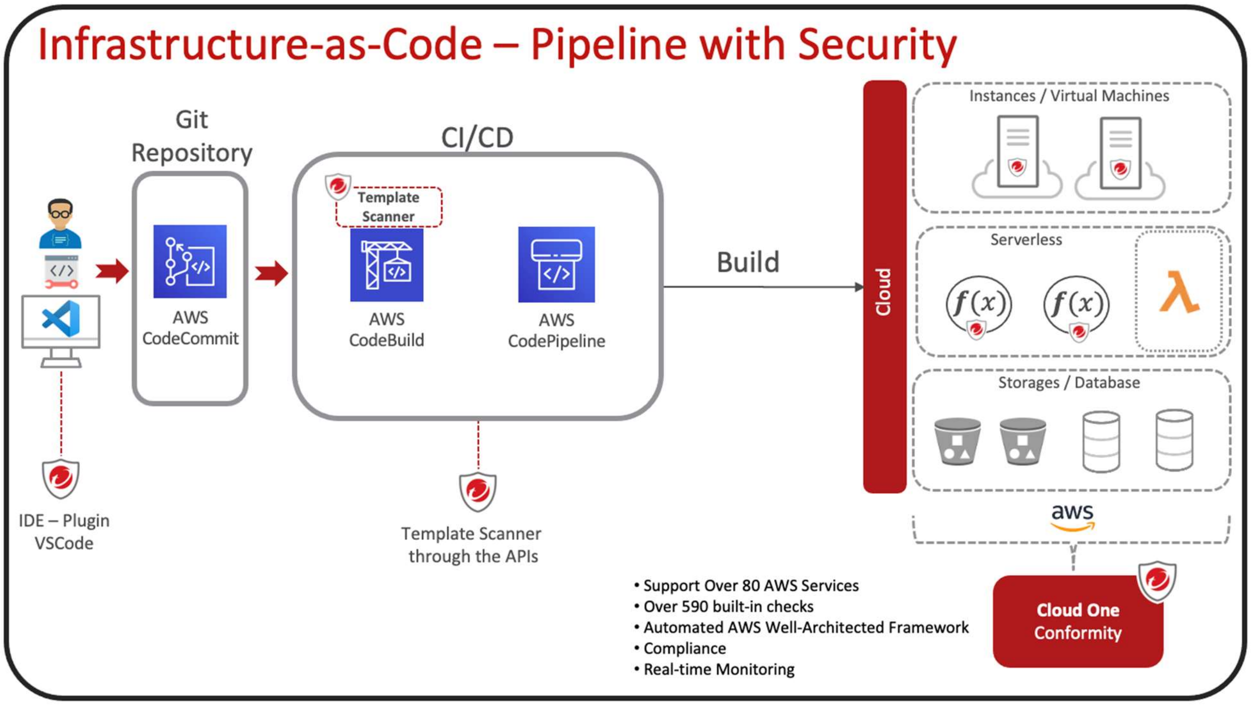
- Disallowed regions
- Naming conventions

You can even build **custom rule plugins** in Go for organization-specific standards.

🔍 16. Deep Security Scanning with checkov



```yaml
---
metadata:
  id: "CKV2_TAG_1"
  name: "Ensure all resouces have 'project: <project-name> tag'"
  category: "CONVENTION"
  severity: "CRITICAL"

definition:
  cond_type: "attribute"
  resource_types:
    - "all"
  attribute: "tags.project"
  operator: "equals"
  value: "checkov-test"
```


Infrastructure-as-Code – Pipeline with Security
Git Repository
AWS CodeCommit
CI/CD
Template Scanner
AWS CodeBuild
AWS CodePipeline
Build
Cloud
Instances / Virtual Machines
Serverless
Storages / Database
aws
IDE – Plugin VSCode
Template Scanner through the APIs
• Support Over 80 AWS Services
• Over 590 built-in checks
• Automated AWS Well-Architected Framework
• Compliance
• Real-time Monitoring
Cloud One Conformity

Skipping False Positives (Advanced)

```
resource "aws_s3_bucket" "logs" {
  bucket = "internal-logs"


  #checkov:skip=CKV_AWS_18:Logging handled by central SIEM
}
```

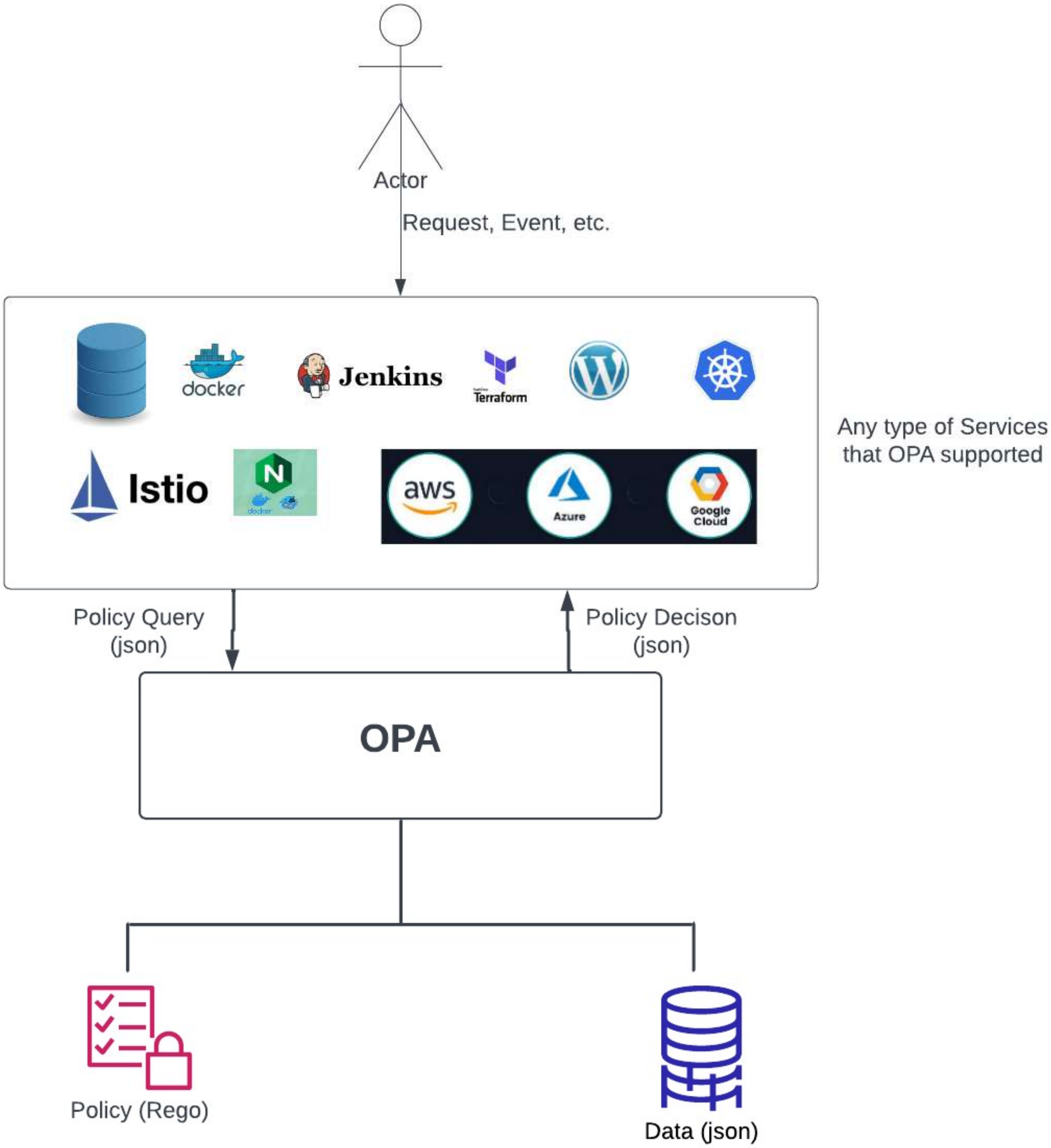
Running Only High Severity

```
checkov -d . --check HIGH
```

Generating SARIF Reports (for GitHub/Jenkins)

```
checkov -d . -o sarif > results.sarif
```

🏛 17. Policy-as-Code Architecture



Layers of Enforcement

Layer	Tool	Purpose
Developer Laptop	tflint	Early feedback
PR Pipeline	checkov	Security checks
Plan Stage	OPA	Policy validation
Apply Stage	Sentinel	Enterprise enforcement
Runtime	Cloud Guardrails	SCP / Azure Policy

🧾 18. Writing Advanced OPA Policies

Enforce Tagging Standard

```
package terraform.tags


required_tags = {"Environment", "Owner", "CostCenter"}


deny[msg] {
  input.resource_type == "aws_instance"
  some tag in required_tags
  not input.config.tags[tag]
  msg = sprintf("Missing required tag: %s", [tag])
}
```

Restrict Regions

```
package terraform.region


deny[msg] {
  input.provider.region != "us-east-1"
  msg = "Only us-east-1 region is allowed"
}
```

🏢 19. Sentinel Enterprise Example

Used in **Terraform Cloud/Enterprise**:

```
import "tfplan/v2" as tfplan


main = rule {
  all tfplan.resources.aws_s3_bucket as _, bucket {
    bucket.applied.server_side_encryption_configuration is not null
  }
}
```

Sentinel operates at the **organization level**, not just project level.

🔚 20. Jenkins Advanced Pipeline Integration

```
stage('Static Analysis') {
  steps {
    sh 'terraform validate'
    sh 'tflint --format json'
  }
}


stage('Security Scan') {
  steps {
    sh 'checkov -d . --quiet'
  }
}
```

```
stage('Policy Check') {
  steps {
    sh 'opa eval --data policy.rego --input tfplan.json "data.terraform.deny"'
  }
}
```

Pipeline Design Tip:
Fail fast at each stage → Developers get immediate feedback.

🧠 21. Policy Design Best Practices

Principle	**Description**
Start permissive	Avoid blocking teams initially
Gradual tightening	Move from warn → fail
Version policies	Store in Git
Central governance repo	One source of truth
Document exceptions	Formal waiver process

⚠️ 22. Enterprise Pitfalls

- Overly strict policies block innovation
- Too many exceptions weaken governance
- Policies not tested → pipeline failures
- No policy ownership → outdated rules

📌 Advanced Key Takeaways

- Static analysis is part of a **layered security model**

- tflint enforces Terraform hygiene
- checkov enforces security best practices
- OPA enables **flexible, code-based governance**
- Sentinel provides **enterprise-grade enforcement**
- Mature organizations implement **multi-stage policy gates**

🧪 Advanced Knowledge Check

1. Which stage should block region violations?
A. tflint
B. checkov
C. OPA
D. terraform fmt

2. What is the benefit of SARIF output from checkov?
A. Faster scans
B. Integration with security dashboards
C. Reduces policy failures
D. Encrypts Terraform state

3. Where does Sentinel operate?
A. Local CLI
B. GitHub Actions
C. Terraform Cloud/Enterprise
D. AWS Lambda

✅ Answers

1 → C
2 → B
3 → C

If you'd like, next I can provide:

✓ A **full enterprise OPA policy bundle**

✓ A **complete Jenkinsfile with PR gates**

✓ A **diagram slide showing governance flow from dev → prod**