Enterprise Infrastructure as Code (IaC) Security and CI/CD Enforcement on AWS

Author: Adedayo
Specialization: Cloud Security & DevSecOps
Platform: Amazon Web Services (AWS)

1. Problem Statement

A software development organization deploying infrastructure via Terraform and CloudFormation lacked preventive security controls in CI/CD pipelines. Insecure configurations were reaching production environments. The organization required automated guardrails and pre-deployment validation.

I implemented these controls across Terraform and CloudFormation workflows to ensure that infrastructure changes meet security standards before reaching production.

The solution integrates security validation directly into CI/CD pipelines, enabling early detection and prevention of misconfigurations.

2. Objectives

The primary objectives of this implementation were to:

- Prevent insecure infrastructure deployments
- Detect misconfigurations at the code level
- Enforce least-privilege and defense-in-depth principles
- Integrate security into CI/CD pipelines
- Establish standardized security guardrails
- Reduce production security incidents

3. Infrastructure as Code Environment

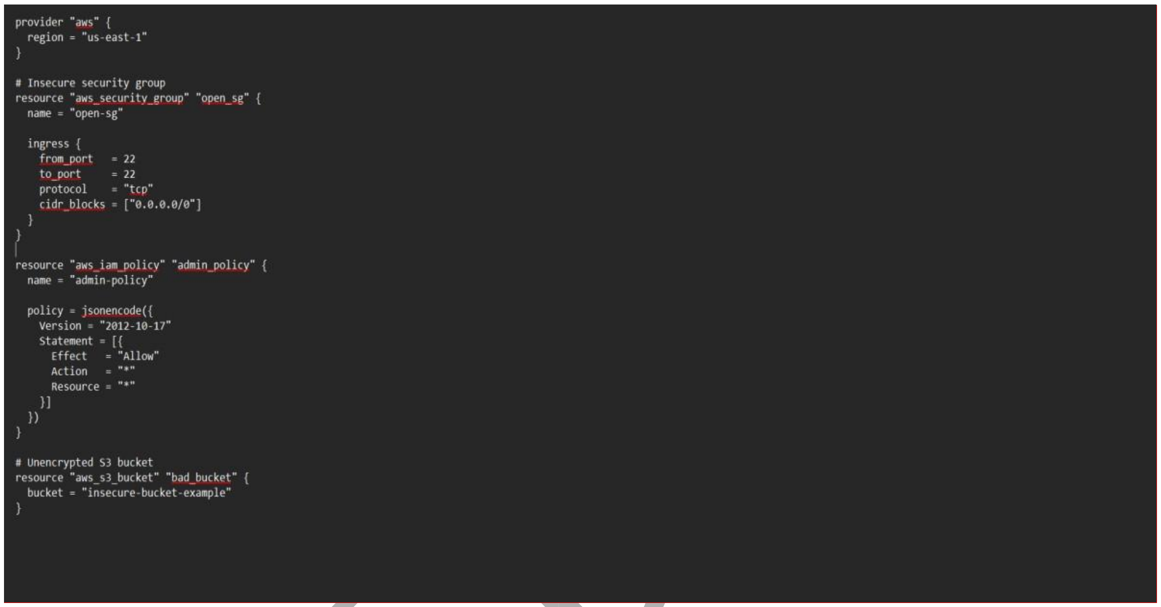
The environment uses:

- Terraform for infrastructure provisioning
- CloudFormation for service-specific deployments
- GitHub Actions for CI/CD automation

All infrastructure changes are delivered through version-controlled repositories and automated pipelines.

4. Manual IaC Security Review

Before automation was introduced, I performed manual security reviews of Terraform templates.

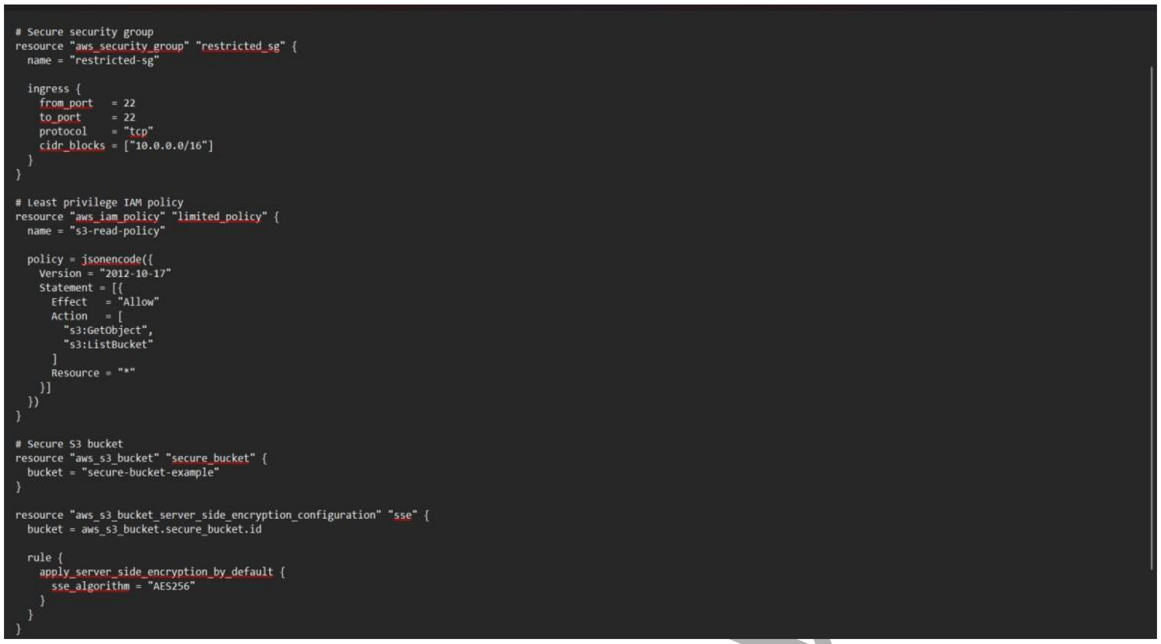


```
provider "aws" {
  region = "us-east-1"
}

# Insecure security group
resource "aws_security_group" "open_sg" {
  name = "open-sg"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_iam_policy" "admin_policy" {
  name = "admin-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect   = "Allow"
      Action   = "*"
      Resource = "*"
    }]
  })
}

# Unencrypted S3 bucket
resource "aws_s3_bucket" "bad_bucket" {
  bucket = "insecure-bucket-example"
}
```

The review focused on identifying:

- Overly permissive IAM policies
- Public network exposure
- Insecure security group rules
- Missing encryption controls
- Improper storage configurations

Reviewed components included:



```
# Secure security group
resource "aws_security_group" "restricted_sg" {
  name = "restricted-sg"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/16"]
  }
}

# Least privilege IAM policy
resource "aws_iam_policy" "limited_policy" {
  name = "s3-read-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect   = "Allow"
      Action   = [
        "s3:GetObject",
        "s3:ListBucket"
      ]
      Resource = "*"
    }]
  })
}

# Secure S3 bucket
resource "aws_s3_bucket" "secure_bucket" {
  bucket = "secure-bucket-example"
}

resource "aws_s3_bucket_server_side_encryption_configuration" "sse" {
  bucket = aws_s3_bucket.secure_bucket.id

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

- IAM roles and policies
- Security groups
- S3 bucket settings
- Encryption configurations
- Network definitions

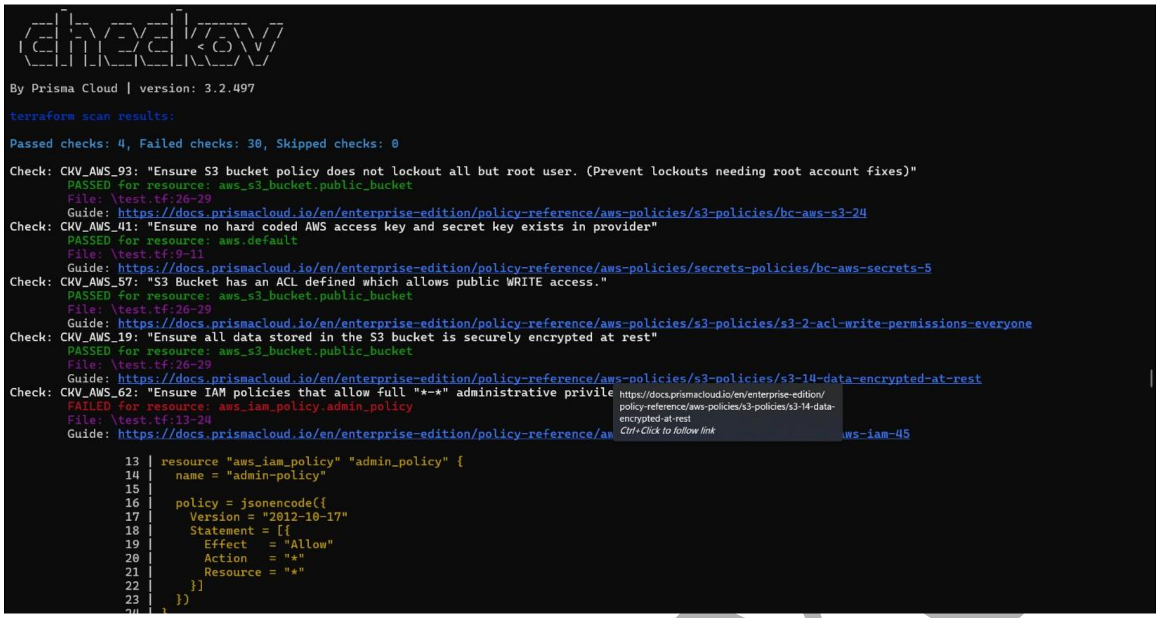
Multiple misconfigurations were identified and remediated using secure design patterns.

This process established a secure baseline for automation.

5. Automated Security Scanning

Automated scanning was implemented using:

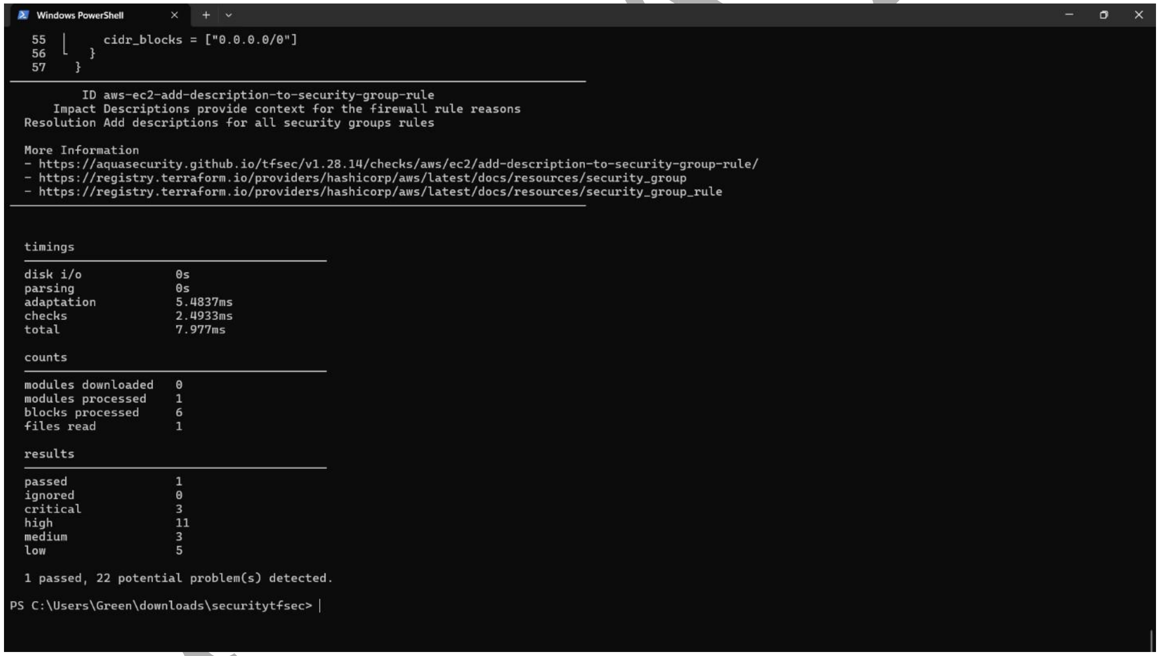
- Checkov



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

By Prisma Cloud | version: 3.2.497

terraform scan results:

Passed checks: 4, Failed checks: 30, Skipped checks: 0

Check: CKV_AWS_93: "Ensure S3 bucket policy does not lockout all but root user. (Prevent lockouts needing root account fixes)"
        PASSED for resource: aws_s3_bucket.public_bucket
        File: \test.tf:26-29
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/bc-aws-s3-24
Check: CKV_AWS_41: "Ensure no hard coded AWS access key and secret key exists in provider"
        PASSED for resource: aws.default
        File: \test.tf:9-11
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/secrets-policies/bc-aws-secrets-5
Check: CKV_AWS_57: "S3 Bucket has an ACL defined which allows public WRITE access."
        PASSED for resource: aws_s3_bucket.public_bucket
        File: \test.tf:26-29
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/s3-2-acl-write-permissions-everyone
Check: CKV_AWS_19: "Ensure all data stored in the S3 bucket is securely encrypted at rest"
        PASSED for resource: aws_s3_bucket.public_bucket
        File: \test.tf:26-29
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/s3-14-data-encrypted-at-rest
Check: CKV_AWS_62: "Ensure IAM policies that allow full "*-*" administrative privile
        FAILED for resource: aws_iam_policy.admin_policy
        File: \test.tf:13-24
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aw                      ws-iam-45

                13 | resource "aws_iam_policy" "admin_policy" {
                14 |   name = "admin-policy"
                15 |
                16 |   policy = jsonencode({
                17 |     Version = "2012-10-17"
                18 |     Statement = [{
                19 |       Effect   = "Allow"
                20 |       Action   = "*"
                21 |       Resource = "*"
                22 |     }]
                23 |   })
```

https://docs.prismacloud.io/en/enterprise-edition/
policy-reference/aws-policies/s3-policies/s3-14-data-
encrypted-at-rest
Ctrl+Click to follow link

- tfsec



```
  55  │     cidr_blocks = ["0.0.0.0/0"]
  56  └   }
  57    }
─────────────────────────────────────────────────────────────────────────────────
          ID aws-ec2-add-description-to-security-group-rule
      Impact Descriptions provide context for the firewall rule reasons
  Resolution Add descriptions for all security groups rules

  More Information
  - https://aquasecurity.github.io/tfsec/v1.28.14/checks/aws/ec2/add-description-to-security-group-rule/
  - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group
  - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group_rule
─────────────────────────────────────────────────────────────────────────────────


  timings
  ──────────────────────────────────────────
  disk i/o             0s
  parsing              0s
  adaptation           5.4837ms
  checks               2.4933ms
  total                7.977ms

  counts
  ──────────────────────────────────────────
  modules downloaded   0
  modules processed    1
  blocks processed     6
  files read           1

  results
  ──────────────────────────────────────────
  passed               1
  ignored              0
  critical             3
  high                 11
  medium               3
  low                  5

  1 passed, 22 potential problem(s) detected.

PS C:\Users\Green\downloads\securitytfsec> |
```

These tools analyze Terraform templates for common cloud security risks.

Detected issues included:

- Public resource exposure
- Wildcard IAM permissions
- Unencrypted storage

- Insecure networking rules

Findings were remediated and scans were re-run to validate compliance.

6. CI/CD Pipeline Enforcement

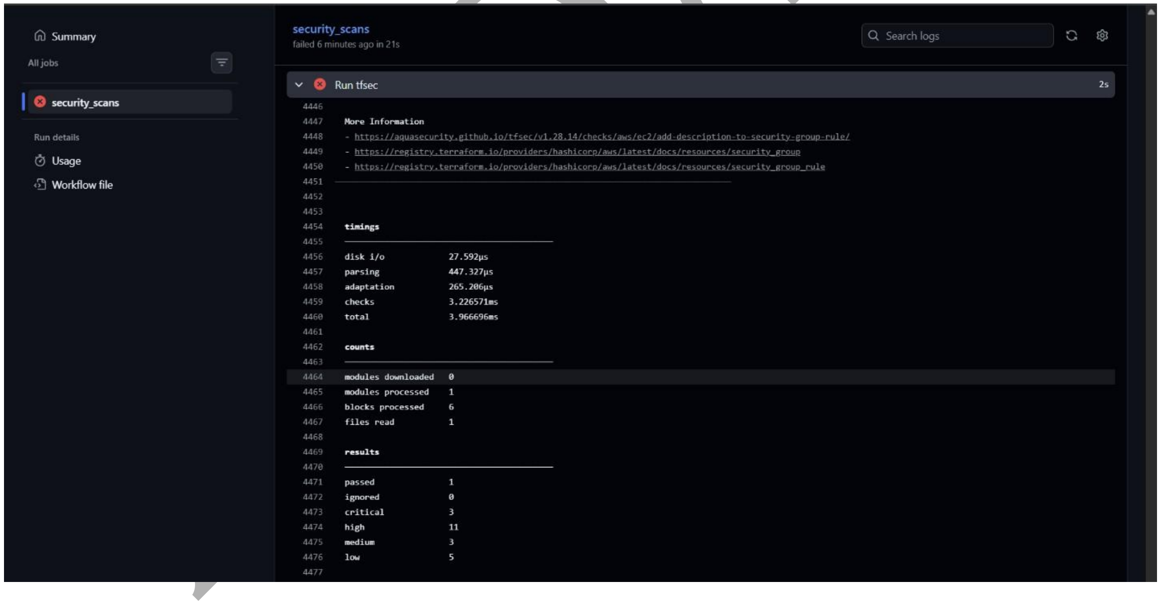
IaC security scanning was enforced using GitHub Actions.

The pipeline executes Checkov and tfsec on:

- Every push
- Every pull request

When critical or high-risk violations are detected, the pipeline fails and blocks deployment.

This ensures insecure infrastructure cannot progress through the delivery process.



```

More Information
- https://aquasecurity.github.io/tfsec/v1.28.14/checks/aws/ec2/add-description-to-security-group-rule/
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group_rule
────────────────────────────────────────────────────────────────────────────



timings
──────────────────────────────────────────
disk i/o             27.592µs
parsing              447.327µs
adaptation           265.206µs
checks               3.226571ms
total                3.966696ms

counts
──────────────────────────────────────────
modules downloaded   0
modules processed    1
blocks processed     6
files read           1

results
──────────────────────────────────────────
passed               1
ignored              0
critical             3
high                 11
medium               3
low                  5

```

7. Security Guardrail Framework

A formal guardrail policy was established to define minimum security requirements.

7.1 Encryption Guardrails

- All S3 buckets must be encrypted
- Customer-managed KMS keys for sensitive data
- CloudTrail logs encrypted with KMS
- Unencrypted storage prohibited

7.2 Network Exposure Guardrails

- No unrestricted access to administrative ports
- Restricted inbound CIDR ranges
- Limited outbound traffic
- Documented public exposure

7.3 IAM Permission Guardrails

- Least-privilege enforcement
- No wildcard permissions
- Purpose-specific roles
- Mandatory review

7.4 Logging and Monitoring Guardrails

- CloudTrail enabled in all regions
- Protected log storage
- S3 access logging enabled
- Logging protected from disablement

7.5 CI/CD Enforcement Guardrails

- Mandatory security scans
- Automatic pipeline failures
- No unauthorized overrides

7.6 Approved Standards

- Approved Terraform modules
- Standardized configurations

- Reviewed deviations

8. Local Development Security Controls

Developers were required to install Checkov and tfsec locally.

Local scans, formatting, and validation were mandatory before commits.

This reduced the introduction of insecure code into repositories.

9. CloudFormation Security Scanning

cfn-nag was integrated into the CI/CD pipeline to scan CloudFormation templates.

If no templates were present, the step was skipped automatically.

This ensured consistent enforcement across IaC formats.

10. Governance and Documentation

All security standards, review procedures, and pipeline controls were documented.

Documentation supported:

- Audit readiness
- Compliance reporting
- Security reviews
- Knowledge transfer

11. Outcomes and Impact

This implementation delivered:

- Early detection of IaC security risks
- Reduced misconfigurations

- Strong DevSecOps integration
- Standardized infrastructure security
- Improved deployment reliability
- Reduced security incidents

12. Conclusion

I designed and implemented an enterprise-grade IaC security framework that integrates preventive controls into development and deployment workflows.

Through automated scanning, CI/CD enforcement, and formal guardrails, this solution ensures cloud infrastructure is deployed securely and consistently.