Policy as Code for Cloud Security and Compliance

Presenter:
Iman Elsakaan
Jianyi Fan
Parth Bodana
Shaoxian Duan

Agenda

Introduction to Policy as Code

Importance of Cloud Security

What is OPA and Conftest?

Terraform + OPA Integration

Demo Walkthrough

Best Practices

Q&A

https://vpnoverview.com/news/sega-europe-security-report/
https://www.vpnmentor.com/blog/report-sennheiser-leak/
https://www.vpnmentor.com/blog/report-ghana-nss-leak/
https://www.comparitech.com/blog/information-security/utah-covid-test-center-leak/

What is Policy as Code (PaC)?

PAC MEANS WRITING SECURITY AND COMPLIANCE RULES IN CODE

ENFORCED AUTOMATICALLY DURING DEVELOPMENT AND DEPLOYMENT

What is Policy as Code (PaC)?

- Policy as Code means writing security and compliance rules as code
- Enables automated enforcement in CI/CD pipelines
- Ensures consistent and auditable compliance checks
- Policies define what is allowed or denied in infrastructure and application behavior.

Common Type of PaC

- Security
- Compliance
- Governance
- FinOps

Why Use Policy as Code?

Reduces the risk of human error

Enforces consistent rules

Increases speed and safety of deployments

Key Benefit of Using PaC

- **Proactive Security:** Catch violations before deployment.
- **Compliance Automation:** Ensure adherence to standards (SOC2, PCI-DSS, etc.)
- **Reduced Human Error:** Eliminate manual policy checks.
- **Audit Trail:** Complete history of policy decisions.
- **Automation:** Policies run automatically in pipelines
- **Consistency:** policies used among dev, test, and prod, no additional policies needed
- **Vision Control:** Track policies changes over time.
- **Speed:** Rapid policy deployment.
- **Testable:** Policies can be test easily treat as code.

Traditional vs. Policy as Code

Aspect	Traditional Approach	Policy as Code (PaC)
Policy Management	Policies exist in documents or team guidelines	Policies are written as **code**, version-controlled
Enforcement	Manual checks by security or DevOps teams	**Automated validation** in CI/CD pipelines
Timing	Late in development or pre-deployment reviews	**Early in the Dev cycle** (shift-left security)
Error Detection	After resources are deployed	**Before resources are deployed**
Scalability	Difficult to maintain across teams and projects	Scales easily with code reuse and pipelines
Auditability	Not always tracked or consistent	Changes are tracked via Git, enabling audits
Developer Experience	Slows down deployment due to late feedback	Developers get **instant feedback** on violations

Importance of Cloud Security

Why Cloud Security Matters

With the widespread adoption of **public cloud platforms** like Azure, AWS, and GCP, organizations are moving critical workloads—including sensitive data, customer information, and internal systems—to the cloud.

While the cloud offers scalability and flexibility, it also **introduces new attack surfaces and risks** if not secured properly.

Compliance Challenges in Cloud

Organizations operating in regulated industries—like **healthcare, finance, and government**—must comply with **strict legal and industry standards** for data protection, security, and privacy.

Cloud environments introduce new challenges:

- Shared responsibility between provider and customer
- Fast-changing infrastructure
- Difficulty maintaining continuous compliance at scale

How PaC Helps with Compliance

Compliance teams often work separately from developers, leading to **delays and missed requirements**.
Policy as Code (PaC) integrates compliance directly into the development process—**shifting compliance left** and making it a shared responsibility.

DevSecOps is the practice of integrating **security and compliance into DevOps workflows**

PaC is a key enabler of DevSecOps because it ensures:

- Security policies are not optional
- Compliance checks are automated
- Feedback is delivered **early in the CI/CD pipeline**

PaC embeds **security and compliance policies** into code — just like application logic or infrastructure — and treats them as **first-class citizens** in the development lifecycle.

Core Tool Used

- **Terraform:** Infrastructure as Code tool for building, changing, and versioning infrastructure
- **Open Policy Agent (OPA):** General-purpose policy engine that provides a unified way to enforce policies across your stack
- **Conftest:** Utility for testing structured configuration data against OPA policies
- **Github Actions:** CI/CD platform for automating policy enforcement in your deployment pipeline

Developer → Git Push → GitHub Actions → Terraform Plan → Conftest/OPA → Policy Validation → Deploy/Reject

Overview of Open Policy Agent (OPA)

Open-source policy engine

Works with many systems (K8s, Terraform, APIs)

Rego – OPA's Policy Language

Declarative language for defining rules

Simple syntax example for deny rule

What is Conftest?

CLI tool to evaluate config files using OPA

Works with JSON/YAML (Terraform plans, K8s manifests)

Terraform Overview

IaC tool for managing cloud infrastructure

Terraform plan and apply lifecycle

OPA + Terraform Integration

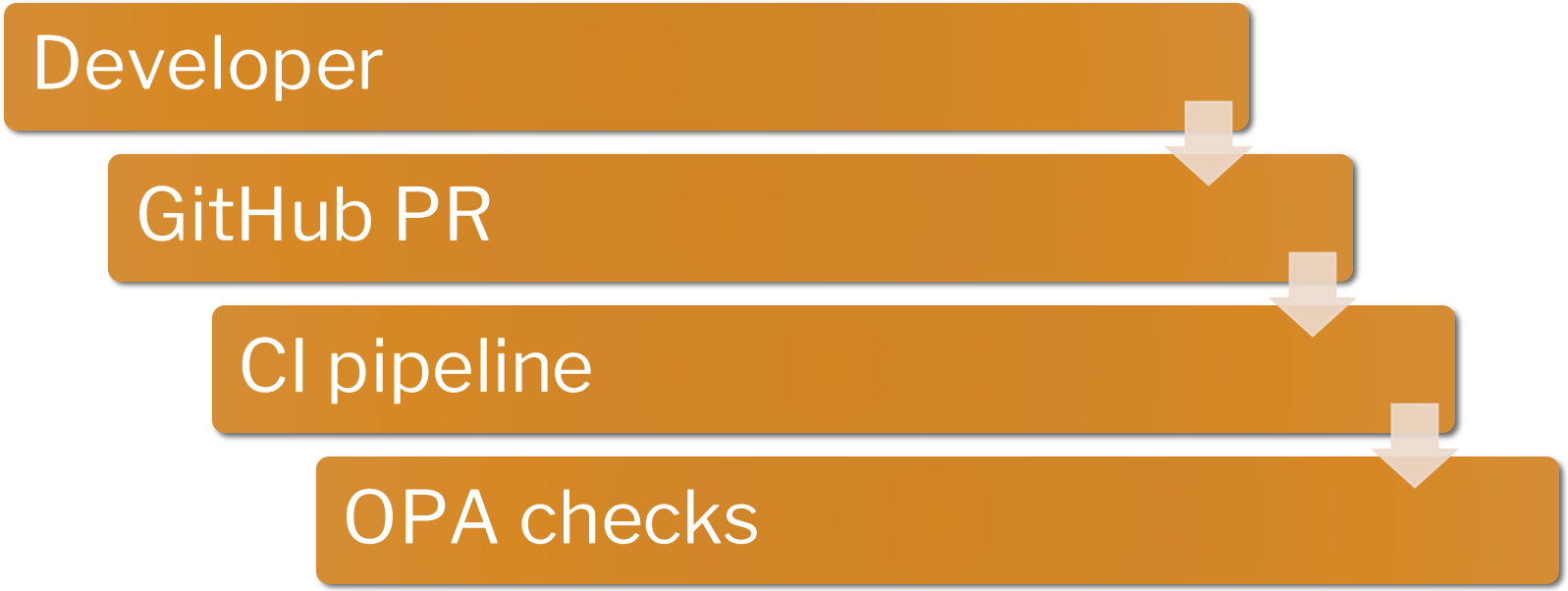
Terraform plan output as JSON

Conftest checks that plan using Rego rules

GitHub Actions in CI/CD

- AUTOMATES CHECKS ON PUSH OR PR
- INTEGRATES WITH TERRAFORM AND CONFTEST

CI/CD Workflow Diagram



Policy #1 Prohibit overly permissive firewall rules

```
# Strategy 1: Prohibit overly permissive firewall rules
deny[msg] {
  resource := input.resource_changes[_]
  resource.type == "azurerm_network_security_group"
  rule := resource.change.after.security_rule[_]
  rule.direction == "Inbound"
  rule.source_address_prefix == "0.0.0.0/0"
  rule.destination_port_range != "443"
  msg := sprintf("NSG '%s' contains an overly permissive inbound rule allowing '0.0.0.0/0' on port '%s'.",
}
```

Policy #2 - Enforce Owner tags

```
# Strategy 2: Enforce Owner tags
deny[msg] {
  resource := input.resource_changes[_]
  allowed_types := ["azurerm_resource_group", "azurerm_network_security_group"]
  resource.type == allowed_types[_]
  not resource.change.after.tags["Owner"]
  msg := sprintf("Resource '%s' is missing required tag 'Owner'.", [resource.change.after.name])
}
```

GitHub Actions Integration

main.tf | README.MD | policy.rego | terraform-conftest.yml

.github > workflows > terraform-conftest.yml

```yaml
jobs:
  terraform-conftest:
    steps:
      # Verify Conftest policies existence
      - name: Check OPA policies
        run: |
          if [ ! -d "policies" ] || [ -z "$(ls -A policies/*.rego 2>/dev/null)" ]; then
            echo "Error: No OPA policy files found in 'policies' directory"
            exit 1
          fi
          echo "Found OPA policies:"
          ls -l policies/*.rego

      # Verify Conftest policies
      - name: Verify Conftest Policies
        run: |
          conftest verify -p policies 2> conftest_verify_error.log
          if [ $? -ne 0 ]; then
            echo "Invalid OPA policies detected. Error details:"
            cat conftest_verify_error.log
            exit 1
          fi
          echo "OPA policies verified successfully"

      # Run Conftest test with OPA policies and detailed logs
      - name: Run Conftest Test with OPA Policies
        run: |
```

GitHub Actions Integration

Conftest Test

succeeded 27 minutes ago in 5s

Search logs

Set up job	0s
Checkout code	1s
Install Conftest	1s
Check Terraform Plan JSON	0s
Check OPA policies	0s
Verify Conftest Policies	0s
Run Conftest Test with OPA Policies	0s
Upload Conftest logs on failure	0s
Post Checkout code	0s
Complete job	0s

Thank You!

Quiz Time.