Terraform x OPA/Conftest の tips

Open Policy Agent Rego
Knowledge Sharing Meetup #2021.07
Ryo Kubota @ryok6t

自己紹介

- Ryo Kubota (@ryok6t)
- FiNC Technologies
- SRE Team manager

前提

- マイクロサービスを AWS EKS 上にデプロイ
- 各サービスのインフラは以下でコード管理
  - Kubernetes の manifest
  - Terraform
- 各開発チームが自身でこれらのコードを書いている
- 質の担保のために Conftest を利用

以前の発表

- https://speakerdeck.com/ryokbt/terraformfalserebiyuwoconftestdezi-dong-hua-suru

本日の内容

- （特に）Terraform で OPA を使う場合のちょっとしたコツ
- 自分が OPA を導入する前に知りたかったことのまとめ

まずは plan を JSON に

- plan 結果を JSON にするところから
  - terraform plan -out plan.tfplan
  - terraform show -json plan.tfplan | conftest test -

Terraform plan の JSON

リソースとそれに対する変更一覧

リソースの情報

加える操作（create, update など）

変更前/変更後の状態

ポリシーを書く

- 「resource type が security group で、port が

  フルオープンなものが1個でもあれば violation」

violation/deny/warn の prefix をつける

```
violation_port_full_open[msg] {
    # resource changes を1つずつ確認
    resource := input.resource_changes[_]

    # これ以降を全て満たすものが1つでもあれば violation
    resource.type == "aws_security_group_rule"

    block := resource.change.after.rule.cidr_blocks[_]
    contains(block, "0.0.0.0/0")
    msg := "port full open"
}
```

共通の処理をまとめる

- 共通の処理をするケースが多い
  - e.g. 特定の resource type の時のみ適用する
  - 先の例だと、security group の時だけ適用するなど
- 毎回書くのは面倒

function を使った共通処理

```
violation_port_full_open[msg] {
    resource := input.resources("aws_security_group_rule")[_]

    block := resource.change.after.rule.cidr_blocks[_]
    contains(block, "0.0.0.0/0")
    msg := "port full open"
}

resources(type) = all {
    all := [name |
        name:= input.resource_changes[_]
        name.type == type
    ]
}
```

受け取った type に
当てはまるものだけを返す

共通処理を別のファイルに切り出す

- 共通処理用の package を作っておく

```
package base

import input as tfplan

resources(type) = all {
    all := [name |
        name:= tfplan.resource_changes[_]
        name.type == type
    ]
}
```

共通処理を別ファイルから呼び出す

```
package tmp

import data.base

violation_port_full_open[msg] {
    resource := base.resources("aws_security_group_rule")[_]

    block := resource.change.after.rule.cidr_blocks[_]
    contains(block, "0.0.0.0/0")
    msg := "port full open"
}
```

共通処理用の package を import

base.resources で呼び出せるように

例外ケースを扱う

- There is no rule without exceptions
- 特定のリソースだけルールの対象外にしたいケースが存在

“exception” を使って例外に対応

- 例外のロジックに当てはまった場合、rules で指定したものは無視される
- 以下では “deny_foo”, “violation_foo” などが無視される

```
exception[rules] {
  # 例外として扱うロジックを書く

  # 対象外とする rule
  rules = ["foo","bar"]
}
```

テストを書く

- Rego の文法にはクセがあるため、テストが重要
- Conftest 自体で簡単にテストが可能

テストの方法

- foo.rego に対して、foo_test.rego というファイルを用意
- conftest verify を実行

テストの例（violation になるケース）

```
test_port_full_open {
    plan = `
      resource_changes:
        - name: some_rule
          type: aws_security_group_rule
          change:
            actions:
              - create
            after:
              rule:
                cidr_blocks:
                  - 0.0.0.0/0
    `
    input := yaml.unmarshal(plan)

    violation_port_full_open["port full open"] with input as input
}
```

フルオープンになっているので
violation になることをテスト

テストコードの構造

```
test_port_full_open {
    plan = `
      resource_changes:
        - name: some_rule
          type: aws_security_group_rule
          change:
            actions:
              - create
            after:
              rule:
                cidr_blocks:
                  - 0.0.0.0/0
    `
    input := yaml.unmarshal(plan)

    violation_port_full_open["port full open"] with input as input
}
```

plan の JSON と同じ構造で
テストデータを生成

with/as を使って
テストデータを与える

テストの例(violation にならないケース)

```
test_port_not_open {
    plan = `
      resource_changes:
        - name: some_rule
          type: aws_security_group_rule
          change:
            actions:
              - create
            after:
              rule:
                cidr_blocks:
                  - 8.8.8.8/32
    `
    input := yaml.unmarshal(plan)

    not violation_port_full_open["port full open"] with input as input
}
```

テストの例(violation にならないケース)

- not をつけるだけ

```
test_port_not_open {
    plan = `
      resource_changes:
        - name: some_rule
          type: aws_security_group_rule
          change:
            actions:
              - create
            after:
              rule:
                cidr_blocks:
                  - 8.8.8.8/32
    `
    input := yaml.unmarshal(plan)

    not violation_port_full_open["port full open"] with input as input
}
```

テストデータについて

- 公式ドキュメントでは、テストコードに JSON を直接書いている

```
test_get_another_user_denied {
    not allow with input as {"path": ["users", "bob"], "method": "GET", "user_id": "alice"}
}
```

- 引用: https://www.openpolicyagent.org/docs/latest/policy-testing/

テストデータについて

- しかし JSON を直接書くと辛いケースが多い
- 現在は yaml で書いて yaml.unmarshal している

最後にデバッグについて

- policy やそのテストを書いているとデバッグがしたくなる
- 任意の箇所に trace(string) を仕込むことでデバッグ出力が可能
- `—trace` オプション付きで Conftest を実行

```
trace(sprintf("name: %v", [resource.name]))
```