

Aqui está o código Rego que você gerou:

```
package terraform

# Política de negação para
aws_s3_bucket_public_access_block
#
# Objetivo:
# Negar a CRIAÇÃO de qualquer recurso
aws_s3_bucket_public_access_block
# quando QUALQUER um dos seguintes atributos estiver
definido como false:
# - block_public_acls
# - block_public_policy
# - ignore_public_acls
# - restrict_public_buckets
#
# Entrada:
# - input.resource_changes (Terraform plan em JSON)
#
# Compatível com OPA 1.11.0 (Rego v1 – sintaxe com if)

deny[msg] if {
  # Itera sobre as mudanças de recursos do plano
  rc := input.resource_changes[_]

  # Apenas o recurso desejado
  rc.type == "aws_s3_bucket_public_access_block"

  # Apenas quando houver criação do recurso
  rc.change.actions[_] == "create"

  # Estado final planejado
  after := rc.change.after

  # Lista dos campos obrigatórios de proteção
  public_access_flags := [
    after.block_public_acls,
    after.block_public_policy,
    after.ignore_public_acls,
```

```

        after.restrict_public_buckets
    ]

    # Se QUALQUER um deles for false, a política nega
    public_access_flags[] == false

    # Mensagem de erro retornada pelo deny
    msg := sprintf(
        "Criação negada: o recurso %s (%s) possui
        configurações inseguras de acesso público ao S3. Todos os
        atributos de Public Access Block devem ser true.",
        [rc.name, rc.address]
    )
}

```

Revise este código procurando por erros de lógica comuns ao analisar JSONs de planos do Terraform (ex: iterar sobre `resource_changes` incorretamente).
 Liste os problemas potenciais e, em seguida, reescreva o código corrigido.

Perfeito — o código **está quase correto**, mas há alguns **pontos clássicos de armadilha ao analisar** `terraform plan -json` que podem causar falsos positivos, falsos negativos ou até erros em tempo de avaliação no OPA.

Vou dividir em duas partes:

1. Problemas potenciais / armadilhas comuns
2. Versão corrigida e mais robusta da política

1. Problemas potenciais no código atual

1. `rc.change.after` pode ser `null`

No Terraform plan JSON:

- Em **delete**, `after` = `null`
- Em **create**, `before` = `null`
- Em **update**, ambos existem

👉 Seu código filtra `create`, então **normalmente** `after` existe, mas:

- Em planos parciais ou recursos computados, campos podem estar ausentes
- OPA pode falhar se tentar acessar `after.block_public_acls` quando `after` é `null`

📌 **Boa prática:** verificar explicitamente que `after` não é `null`.

2. Campos podem não existir (unknown / computed)

No plan:

- Valores podem estar como `"unknown"` (especialmente se vêm de variáveis, módulos ou outputs)
- Campo pode simplesmente não existir

Exemplo:

json

```
"after": {  
  "block_public_acls": true  
}
```

📄 Copiar código

Se você acessar diretamente:

rego

```
after.block_public_policy
```

📄 Copiar código

👉 isso **falha** se o campo não existir.

📌 **Boa prática:** usar `object.get(after, "campo", true)`
(assumir `true` como default seguro).

3. actions pode conter mais de um valor

Casos reais:

json


```
"actions": ["create"]  
"actions": ["create", "delete"] # replace
```

📄 Copiar código

Seu código:

rego

```
rc.change.actions[_] == "create"
```

 Copiar código

👉 Isso **também pega replace**, o que pode ser desejado ou não.

📌 Decisão de segurança:

- Se quiser **bloquear replace também**, ok.
- Se quiser **apenas create puro**, precisa checar o array inteiro.

Vou assumir **create e replace devem ser negados**, pois ambos criam recurso.

🔴 4. Lógica “qualquer false” está correta, mas frágil

```
rego
```

```
public_access_flags[_] == false
```

 Copiar código

Funciona, porém:

- Se algum campo for `null` ou inexistente → erro
- Se campo for `"unknown"` → não é `false`, mas também não é seguro

📌 Boa prática: avaliar campo a campo, com default seguro.

🔴 5. `rc.name` pode não existir

No Terraform plan:

- `name` **nem sempre existe**
- `address` é sempre mais confiável

📌 Melhor usar:

```
rego
```

```
rc.address
```

 Copiar código

2. Código Rego corrigido e mais robusto

✅ Versão recomendada (compatível com OPA 1.11 / Rego v1)

rego

```
package terraform
```

```
# Negar criação (ou replace) de aws_s3_bucket_public_access_block  
# quando qualquer flag de proteção estiver false
```

```
deny[msg] if {  
    rc := input.resource_changes[_]  
  
    # Tipo do recurso  
    rc.type == "aws_s3_bucket_public_access_block"  
  
    # Create ou Replace (create + delete)  
    some a  
    a := rc.change.actions[_]  
    a == "create"  
  
    # Garante que há estado final  
    rc.change.after != null  
    after := rc.change.after  
  
    # Qualquer flag explicitamente false  
    insecure_public_access(after)  
  
    msg := sprintf(  
        "Criação negada: o recurso %s possui configurações inseguras de acesso pú  
        [rc.address]  
    )  
}
```

```
#####
```

```
# Funções auxiliares
```

```
#####
```

```
# Retorna true se qualquer flag obrigatória estiver false
```

```
insecure_public_access(after) {  
    object.get(after, "block_public_acls", true) == false  
}
```

```
insecure_public_access(after) {  
    object.get(after, "block_public_policy", true) == false  
}
```

```
insecure_public_access(after) {
    object.get(after, "ignore_public_acls", true) == false
}






insecure_public_access(after) {
    object.get(after, "restrict_public_buckets", true) == false
}
```

 Copiar código

3. Por que esta versão é melhor

- ✓ Não quebra se `after` for `null`
- ✓ Não quebra se algum campo não existir
- ✓ Trata corretamente `create` e `replace`
- ✓ Assume **default seguro** (`true`) para campos ausentes
- ✓ Usa `rc.address` , que sempre existe
- ✓ Lógica explícita e fácil de auditar (importante em DevSecOps)

4. Se quiser, posso ainda:

-  Tornar a política **fail-safe** (negar se valor for `unknown`)
-  Criar **testes OPA** (`opa test`)
-  Adaptar para **Conftest**
-  Estender para **updates**
-  Retornar mensagens detalhadas por atributo inválido

Se quiser seguir algum desses caminhos, é só dizer.



O código gerado apresentou erro de sintaxe ao ser validado pelo OPA.

Erro: 4 errors occurred: s3_policy.rego:35: rego_parse_error: if keyword is required before function body
 s3_policy.rego:39: rego_parse_error: if keyword is required before function body s3_policy.rego:43: rego_parse_error: if keyword is required before function

body s3_policy.rego:47: rego_parse_error: if keyword is required before function body

Ótimo retorno — esse erro é exatamente o tipo de problema que aparece na migração para Rego v1 👍

O OPA está certo, e o diagnóstico é claro.

📌 Causa do erro (raiz do problema)

Você está usando OPA 1.11.0 (Rego v1), onde:

- **TODA regra** (inclusive "funções auxiliares") **PRECISA do** `if`
- A sintaxe antiga (Rego v0) **não é mais aceita**

❌ O que causou o erro

Estas regras no código anterior:

```
rego

insecure_public_access(after) {
    object.get(after, "block_public_acls", true) == false
}
```

 Copiar código

No **Rego v1**, isso é inválido.

✅ Forma correta no Rego v1

A regra precisa ser escrita assim:

```
rego

insecure_public_access(after) if {
    ...
}
```

 Copiar código

✅ Código FINAL corrigido (100% compatível com OPA 1.11)

Este código:

- ✓ Compila no OPA 1.11
 - ✓ Trata after = null
 - ✓ Trata campos ausentes
 - ✓ Bloqueia create e replace
 - ✓ Usa defaults seguros
 - ✓ Não acessa campos inexistentes
-

s3_policy.rego

rego

package terraform

Nega criação (ou replace) de aws_s3_bucket_public_access_block
quando qualquer flag de proteção estiver false

deny[msg] if {

 rc := input.resource_changes[_]

 # Tipo do recurso

 rc.type == "aws_s3_bucket_public_access_block"

 # Create ou Replace (create aparece em ambos os casos)

 rc.change.actions[_] == "create"

 # Garante que existe estado final

 rc.change.after != null

 after := rc.change.after

 # Verifica configurações inseguras

 insecure_public_access(after)

 msg := sprintf(

 "Criação negada: o recurso %s possui configurações inseguras de acesso público em %s"

)

}

#####

Regras auxiliares (Rego v1)

#####


```
insecure_public_access(after) if {  
    object.get(after, "block_public_acls", true) == false  
}
```

```
insecure_public_access(after) if {  
    object.get(after, "block public policy", true) == false
```

≡ 📁 > ChatGPT ▾ 📺 Promoção gratuita ×  ...

```
insecure_public_access(after) if {  
    object.get(after, "ignore_public_acls", true) == false  
}
```

 Copiar código

```
insecure_public_access(after) if {  
    object.get(after, "restrict_public_buckets", true) == false  
}
```

Validação (exemplo)

bash

 Copiar código

```
opa check s3_policy.rego
```

✓ Nenhum erro de sintaxe

Ou com Conftest:

bash

 Copiar código

```
conftest test plan.json
```

Observação importante de segurança (opcional)

Atualmente:

- Campo ausente → assume `true` (default seguro)
- Campo `unknown` → não dispara deny

Se quiser **fail-closed** (negar se valor for `unknown` ou ausente), posso te entregar uma versão **mais restritiva**, ideal para ambientes regulados.

Se quiser, no próximo passo posso:

- Criar **testes OPA**
- Adaptar para **Conftest**
- Fazer versão **multi-policy**
- Ou explicar **linha por linha** como isso funciona no Terraform Plan JSON

É só dizer.

