Guia do usuário

AWS CloudFormation Guard

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS CloudFormation Guard: Guia do usuário

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens comerciais da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que AWS CloudFormation Guardé 1
Você é usuário do Guard pela primeira vez? 1
Características do Guard 2
Usando o Guard com CloudFormation ganchos 2
Acessando o Guard 3
Práticas recomendadas 3
Configurando o Guard 4
Para Linux e macOS 4
Instale o Guard a partir de um binário de versão pré-construído 4
Instale o Guard do Cargo 5
Instale o Guard do Homebrew 6
No Windows 6
Pré-requisitos 6
Instale o Guard do Cargo 5
Instale o Guard da Chocolatey 8
Como uma AWS Lambda função 8
Pré-requisitos 8
Instale o gerenciador de pacotes Rust 9
Para instalar o Guard como uma função Lambda 9
Para criar e executar 10
Chamando a função Lambda 11
Pré-requisitos e visão geral do uso das regras do Guard 12
Pré-requisitos 12
Visão geral do uso das regras do Guard 12
Regras do Writing Guard 13
Cláusulas 13
Usando consultas em cláusulas 16
Usando operadores em cláusulas 16
Usando mensagens personalizadas em cláusulas 19
Combinando cláusulas 20
Usando blocos com as regras do Guard 21
Definição de consultas e filtragem 25
Atribuição e referência de variáveis nas regras do Guard 39
Composição de blocos de regras nomeadas 46

Escrevendo cláusulas para realizar avaliações contextuais ... 52
Testando as regras do Guard ... 65
Pré-requisitos ... 66
Visão geral ... 66
Demonstração ... 67
Usando parâmetros de entrada com regras do Guard ... 77
Como usar ... 77
Exemplo de uso ... 78
Vários parâmetros de entrada ... 79
Validando os dados de entrada de acordo com as regras do Guard ... 79
Pré-requisitos ... 79
Usar o comando `validate` ... 80
Validando várias regras em relação a vários arquivos de dados ... 80
Proteção de solução de problemas ... 82
A cláusula falha quando nenhum recurso do tipo selecionado está presente ... 82
O Guard não avalia o CloudFormation modelo ... 82
Tópicos gerais de solução de problemas ... 83
CLIReferência de proteção ... 84
Proteja os parâmetros CLI globais ... 84
árvore de análise ... 84
Sintaxe ... 84
Parâmetros ... 85
Opções ... 85
Exemplos ... 85
régua ... 85
Sintaxe ... 86
Parâmetros ... 86
Opções ... 86
Exemplos ... 86
teste ... 86
Sintaxe ... 86
Parâmetros ... 87
Opções ... 87
args ... 87
Exemplos ... 88
Saída ... 88

Consulte também 88
validar 88
Sintaxe 88
Parâmetros 88
Opções 90
Exemplos 91
Saída 91
Consulte também 91
Segurança 92
Histórico do documentos 93
AWS Glossário 95
..... xcvi

O que AWS CloudFormation Guardé

AWS CloudFormation Guard é uma ferramenta de avaliação de código aberto e de uso geral. policy-as-code A interface de linha de comando do Guard (CLI) fornece uma simple-to-use linguagem declarativa específica de domínio (DSL) que você pode usar para expressar políticas como código. Além disso, você pode usar CLI comandos para validar YAML dados JSON ou hierárquicos estruturados em relação a essas regras. O Guard também fornece uma estrutura de teste de unidade integrada para verificar se suas regras funcionam conforme o esperado.

O Guard não valida CloudFormation modelos para sintaxe válida ou valores de propriedades permitidos. Você pode usar a ferramenta cfn-lint para realizar uma inspeção completa da estrutura do modelo.

O Guard não fornece fiscalização pelo lado do servidor. Você pode usar os CloudFormation Hooks para realizar a validação e fiscalização do lado do servidor, onde você pode bloquear ou avisar uma operação.

Para obter informações detalhadas sobre AWS CloudFormation Guard desenvolvimento, consulte o GitHub repositório Guard.

Tópicos

- Você é usuário do Guard pela primeira vez?
- Características do Guard
- Usando o Guard com CloudFormation ganchos
- Acessando o Guard
- Práticas recomendadas

Você é usuário do Guard pela primeira vez?

Se você é um usuário iniciante do Guard, recomendamos que comece lendo as seguintes seções:

- Configurando o Guard— Esta seção descreve como instalar o Guard. Com o Guard, você pode escrever regras de política usando o Guard DSL e validar seus JSON dados estruturados ou YAML formatados em relação a essas regras.
- Regras do Writing Guard— Esta seção fornece orientações detalhadas para redigir regras de política.

- Testando as regras do Guard— Esta seção fornece um passo a passo detalhado para testar suas regras e verificar se elas funcionam conforme o esperado e validar seus JSON dados estruturados ou YAML formatados de acordo com suas regras.
- Validando os dados de entrada de acordo com as regras do Guard— Esta seção fornece uma explicação detalhada para validar seus JSON dados estruturados ou YAML formatados de acordo com suas regras.
- CLIReferência de proteção— Esta seção descreve os comandos que estão disponíveis no GuardCLI.

Características do Guard

Usando o Guard, você pode criar regras de política para validar qualquer JSON dado estruturado ou YAML formatado, incluindo, mas não se limitando a, modelos. AWS CloudFormation O Guard oferece suporte a todo o espectro de end-to-end avaliação de verificações de políticas. As regras são úteis nos seguintes domínios de negócios:

- Governança preventiva e conformidade (teste shift-left) — valide a infraestrutura como código (IaC) ou as composições de infraestrutura e serviços em relação às regras de políticas que representam as melhores práticas organizacionais de segurança e conformidade. Por exemplo, você pode validar CloudFormation modelos, conjuntos de CloudFormation alterações, arquivos de configuração JSON baseados no Terraform ou configurações do Kubernetes.
- Detective a governança e a conformidade — valide a conformidade dos recursos do Configuration Management Database (CMDB), como itens de configuração AWS Config baseados (). CIs Por exemplo, os desenvolvedores podem usar as políticas do Guard AWS Config CIs para monitorar continuamente o estado dos recursos implantados AWS e dos não AWS recursos, detectar violações de políticas e iniciar a remediação.
- Segurança na implantação — garanta que as alterações sejam seguras antes da implantação. Por exemplo, valide conjuntos de CloudFormation alterações em relação às regras de política para evitar alterações que resultem na substituição de recursos, como renomear uma tabela do Amazon DynamoDB.

Usando o Guard com CloudFormation ganchos

Você pode usar o CloudFormation Guard para criar um Hook in CloudFormation Hooks. CloudFormation O Hooks permite que você aplique proativamente suas regras do Guard antes de

CloudFormation criar, atualizar ou excluir operações e AWS API Cloud Control criar ou atualizar operações. Os ganchos garantem que suas configurações de recursos estejam em conformidade com as melhores práticas de segurança, operação e otimização de custos de sua organização.

Para obter detalhes sobre como usar o Guard para criar CloudFormation Guard Hooks, consulte Write Guard rules para avaliar recursos para Guard Hooks no Guia do usuário do AWS CloudFormation Hooks.

Acessando o Guard

Para acessar o Guard DSL e os comandos, você deve instalar o GuardCLI. Para obter informações sobre a instalação do GuardCLI, consulteConfigurando o Guard.

Práticas recomendadas

Escreva regras simples e use regras nomeadas para referenciá-las em outras regras. Regras complexas podem ser difíceis de manter e testar.

Conf AWS CloudFormation Guard iguração

AWS CloudFormation Guard é uma interface de linha de comando de código aberto (CLI). Ele fornece uma linguagem simples e específica de domínio para escrever regras de política e validar suas JSON hierarquias e dados estruturados em relação a essas regras. YAML As regras podem representar as diretrizes da política da empresa relacionadas à segurança, conformidade e muito mais. Os dados hierárquicos estruturados podem representar a infraestrutura em nuvem descrita como código. Por exemplo, você pode criar regras para garantir que elas sempre modelem buckets criptografados do Amazon Simple Storage Service (Amazon S3) em seus modelos. CloudFormation

Os tópicos a seguir fornecem informações sobre como instalar o Guard usando o sistema operacional escolhido ou como uma AWS Lambda função.

Tópicos

- Instalando o Guard para Linux e macOS
- Instalando o Guard para Windows
- Instalando o Guard como uma AWS Lambda função

Instalando o Guard para Linux e macOS

Você pode instalar AWS CloudFormation Guard para Linux e macOS usando o binário de versão pré-construído, Cargo, ou por meio do Homebrew.

Instale o Guard a partir de um binário de versão pré-construído

Use o procedimento a seguir para instalar o Guard a partir de um binário pré-criado.

1. Abra um terminal e execute o comando a seguir.

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://raw.githubusercontent.com/aws-
   cloudformation/cloudformation-guard/main/install-guard.sh | sh
   ```

2. Execute o comando a seguir para definir sua `PATH` variável.

   ```
   export PATH=~/.guard/bin:$PATH
   ```

   Resultados: Você instalou o Guard com sucesso e definiu a `PATH` variável.

- (Opcional) Para confirmar a instalação do Guard, execute o comando a seguir.

```
cfn-guard --version
```

O comando retorna a seguinte saída.

```
cfn-guard 3.0.0
```

Instale o Guard do Cargo

Cargo é o gerenciador de pacotes Rust. Conclua as etapas a seguir para instalar o Rust, que inclui o Cargo. Em seguida, instale o Guard from Cargo.

1. Execute o comando a seguir em um terminal e siga as instruções na tela para instalar o Rust.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- (Opcional) Para ambientes Ubuntu, execute o comando a seguir.

```
sudo apt-get update; sudo apt install build-essential
```

2. Configure sua variável de `PATH` ambiente e execute o comando a seguir.

```
source $HOME/.cargo/env
```

3. Com o Cargo instalado, execute o comando a seguir para instalar o Guard.

```
cargo install cfn-guard
```

Resultados: Você instalou o Guard com sucesso.

- (Opcional) Para confirmar a instalação do Guard, execute o comando a seguir.

```
cfn-guard --version
```

O comando retorna a seguinte saída.

```
cfn-guard 3.0.0
```

Instale o Guard do Homebrew

O Homebrew é um gerenciador de pacotes para macOS e Linux. Conclua as etapas a seguir para instalar o Homebrew. Em seguida, instale o Guard do Homebrew.

1. Execute o comando a seguir em um terminal e siga as instruções na tela para instalar o Homebrew.

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```

2. Com o Homebrew instalado, execute o comando a seguir para instalar o Guard.

   ```
   brew install cloudformation-guard
   ```

   Resultados: Você instalou o Guard com sucesso.

   - (Opcional) Para confirmar a instalação do Guard, execute o comando a seguir.

     ```
     cfn-guard --version
     ```

     O comando retorna a seguinte saída.

     ```
     cfn-guard 3.0.0
     ```

Instalando o Guard para Windows

Você pode instalar AWS CloudFormation Guard para Windows através do Cargo ou através do Chocolatey.

Pré-requisitos

Para criar o Guard a partir da interface de linha de comando, você deve instalar as Ferramentas de Compilação do Visual Studio 2019.

1. Baixe as ferramentas de compilação do Microsoft Visual C++ no site Build Tools for Visual Studio 2019.
2. Execute o instalador e selecione os padrões.

Instale o Guard do Cargo

Cargo é o gerenciador de pacotes Rust. Conclua as etapas a seguir para instalar o Rust, que inclui o Cargo. Em seguida, instale o Guard from Cargo.

1. Baixe o Rust e execute o rustup-init.exe.
2. No prompt de comando, escolha 1, que é a opção padrão.

   O comando retorna a seguinte saída.

   ```
   Rust is installed now. Great!

       To get started you may need to restart your current shell.
       This would reload its PATH environment variable to include
       Cargo's bin directory (%USERPROFILE%\.cargo\bin).

       Press the Enter key to continue.
   ```

3. Para finalizar a instalação, pressione a tecla Enter.
4. Com o Cargo instalado, execute o comando a seguir para instalar o Guard.

   ```
   cargo install cfn-guard
   ```

   Resultados: Você instalou o Guard com sucesso.

   - (Opcional) Para confirmar a instalação do Guard, execute o comando a seguir.

     ```
     cfn-guard --version
     ```

     O comando retorna a seguinte saída.

     ```
     cfn-guard 3.0.0
     ```

Instale o Guard da Chocolatey

Chocolatey é um gerenciador de pacotes para Windows. Conclua as etapas a seguir para instalar o Chocolatey. Em seguida, instale o Guard da Chocolatey.

1. Siga este guia para instalar o Chocolatey
2. Com o Chocolatey instalado, execute o comando a seguir para instalar o Guard.

   ```
   choco install cloudformation-guard
   ```

   Resultados: Você instalou o Guard com sucesso.

   - (Opcional) Para confirmar a instalação do Guard, execute o comando a seguir.

     ```
     cfn-guard --version
     ```

     O comando retorna a seguinte saída.

     ```
     cfn-guard 3.0.0
     ```

Instalando o Guard como uma AWS Lambda função

Você pode instalar AWS CloudFormation Guard por meio do Cargo, o gerenciador de pacotes Rust. Guard as an AWS Lambda function (`cfn-guard-lambda`) é um invólucro leve em torno de Guard (`cfn-guard`) que pode ser usado como uma função Lambda.

Pré-requisitos

Antes de instalar o Guard como uma função Lambda, você deve atender aos seguintes pré-requisitos:

- AWS Command Line Interface (AWS CLI) configurado com permissões para implantar e invocar funções Lambda. Para obter mais informações, consulte Configurar a AWS CLI.
- Uma função de AWS Lambda execução em AWS Identity and Access Management (IAM). Para obter mais informações, consulte função AWS Lambda de execução.
- Em RHEL ambientes CentOS/, adicione o repositório de `musl-libc` pacotes à sua configuração do yum. Para obter mais informações, consulte ngompa/musl-libc.

Instale o gerenciador de pacotes Rust

Cargo é o gerenciador de pacotes Rust. Conclua as etapas a seguir para instalar o Rust, que inclui o Cargo.

1. Execute o comando a seguir em um terminal e siga as instruções na tela para instalar o Rust.

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
   ```

   - (Opcional) Para ambientes Ubuntu, execute o comando a seguir.

     ```
     sudo apt-get update; sudo apt install build-essential
     ```

2. Configure sua variável de `PATH` ambiente e execute o comando a seguir.

   ```
   source $HOME/.cargo/env
   ```

Instale o Guard como uma função Lambda (Linux, macOS ou Unix)

Para instalar o Guard como uma função Lambda, conclua as etapas a seguir.

1. No seu terminal de comando, execute o comando a seguir.

   ```
   cargo install cfn-guard-lambda
   ```

   - (Opcional) Para confirmar a instalação do Guard como uma função Lambda, execute o comando a seguir.

     ```
     cfn-guard-lambda --version
     ```

     O comando retorna a seguinte saída.

     ```
     cfn-guard-lambda 3.0.0
     ```

2. Para instalar o `musl` suporte, execute o comando a seguir.

   ```
   rustup target add x86_64-unknown-linux-musl
   ```

3. Crie com `emusl`, em seguida, execute o seguinte comando no seu terminal.

```
cargo build --release --target x86_64-unknown-linux-musl
```

Para um tempo de execução personalizado, AWS Lambda requer um executável com o nome `bootstrap` no arquivo.zip do pacote de implantação. Renomeie o `cfn-lambda` executável gerado `bootstrap` e adicione-o ao arquivo.zip.

- Para ambientes macOS, crie seu arquivo de configuração de carga na raiz do projeto Rust ou em. `~/.cargo/config`

```
[target.x86_64-unknown-linux-musl]
linker = "x86_64-linux-musl-gcc"
```

4. Mude para o diretório `cfn-guard-lambda` raiz.

```
cd ~/.cargo/bin/cfn-guard-lambda
```

5. Execute o comando a seguir no seu terminal.

```
cp ./../target/x86_64-unknown-linux-musl/release/cfn-guard-lambda ./bootstrap &&
 zip lambda.zip bootstrap && rm bootstrap
```

6. Execute o comando a seguir para enviar `cfn-guard` como uma função Lambda para sua conta.

```
aws lambda create-function --function-name cfnGuard \
 --handler guard.handler \
 --zip-file fileb://./lambda.zip \
 --runtime provided \
 --role arn:aws:iam::444455556666:role/your_lambda_execution_role \
 --environment Variables={RUST_BACKTRACE=1} \
 --tracing-config Mode=Active
```

Para criar e executar o Guard como uma função Lambda

Para invocar o enviado `cfn-guard-lambda` como uma função Lambda, execute o comando a seguir.

```
aws lambda invoke --function-name cfnGuard \
```

```
  --payload '{"data":"input data","rules":["rule1","rule2"]}' \
  output.json
```

Para chamar a estrutura de solicitação da função Lambda

Solicita a `cfn-guard-lambda` exigência dos seguintes campos:

- `data`— A versão em cadeia de caracteres do JSON modelo YAML ou
- `rules`— A versão em cadeia de caracteres do arquivo do conjunto de regras

Pré-requisitos e visão geral do uso das regras do Guard

Esta seção demonstra como você pode concluir as principais tarefas do Guard de escrever, testar e validar regras em relação a dados formatados em JSON ou YAML. Além disso, ele contém orientações detalhadas que demonstram como escrever regras que respondem a casos de uso específicos.

Tópicos

- Pré-requisitos
- Visão geral do uso das regras do Guard
- AWS CloudFormation Guard Regras de redação
- AWS CloudFormation Guard Regras de teste
- Usando parâmetros de entrada com AWS CloudFormation Guard regras
- Validando dados de entrada em relação às regras AWS CloudFormation Guard

Pré-requisitos

Antes de escrever regras de política usando a linguagem específica de domínio (DSL) do Guard, você deve instalar a interface de linha de comando (CLI) do Guard. Para obter mais informações, consulte Configurando o Guard.

Visão geral do uso das regras do Guard

Ao usar o Guard, você normalmente executa as seguintes etapas:

1. Grave dados em formato JSON ou YAML para validar.
2. Escreva as regras da política do Guard. Para obter mais informações, consulte Regras do Writing Guard.
3. Verifique se suas regras funcionam conforme o esperado usando o `test` comando Guard. Para obter mais informações sobre testes unitários, consulteTestando as regras do Guard.
4. Use o `validate` comando Guard para validar seus dados formatados em JSON ou YAML em relação às suas regras. Para obter mais informações, consulte Validando os dados de entrada de acordo com as regras do Guard.

AWS CloudFormation Guard Regras de redação

Dentro AWS CloudFormation Guard, regras são policy-as-code regras. Você escreve regras na linguagem específica do domínio do Guard (DSL) com as quais você pode validar seus dados JSON - ou YAML formatados. As regras são compostas por cláusulas.

Você pode salvar regras escritas usando o Guard DSL em arquivos de texto simples que usam qualquer extensão de arquivo.

Você pode criar vários arquivos de regras e categorizá-los como um conjunto de regras. Os conjuntos de regras permitem que você valide seus dados JSON YAML - ou formatados em relação a vários arquivos de regras ao mesmo tempo.

Tópicos

- Cláusulas
- Usando consultas em cláusulas
- Usando operadores em cláusulas
- Usando mensagens personalizadas em cláusulas
- Combinando cláusulas
- Usando blocos com as regras do Guard
- Definindo consultas e filtragem do Guard
- Atribuição e referência de variáveis nas regras do Guard
- Composição de blocos de regras nomeadas em AWS CloudFormation Guard
- Escrevendo cláusulas para realizar avaliações contextuais

Cláusulas

As cláusulas são expressões booleanas que são avaliadas como verdadeiras (`PASS`) ou falsas (`FAIL`). As cláusulas usam operadores binários para comparar dois valores ou operadores unários que operam em um único valor.

Exemplos de cláusulas unárias

A cláusula unária a seguir avalia se a coleção `TcpBlockedPorts` está vazia.

```
InputParameters.TcpBlockedPorts not empty
```

A cláusula unária a seguir avalia se a `ExecutionRoleArn` propriedade é uma string.

```
Properties.ExecutionRoleArn is_string
```

Exemplos de cláusulas binárias

A cláusula binária a seguir avalia se a `BucketName` propriedade contém a string`encrypted`, independentemente da maiúscula e minúscula.

```
Properties.BucketName != /(?i)encrypted/
```

A cláusula binária a seguir avalia se a `ReadCapacityUnits` propriedade é menor ou igual a 5.000.

```
Properties.ProvisionedThroughput.ReadCapacityUnits <= 5000
```

Sintaxe para escrever cláusulas de regras do Guard

```
<query> <operator> [query|value literal] [custom message]
```

Propriedades das cláusulas da regra de Guarda

`query`

Uma expressão separada por ponto (`.`) escrita para atravessar dados hierárquicos. As expressões de consulta podem incluir expressões de filtro para atingir um subconjunto de valores. As consultas podem ser atribuídas a variáveis para que você possa escrevê-las uma vez e referenciá-las em outro lugar em um conjunto de regras, o que permitirá acessar os resultados da consulta.

Para obter mais informações sobre como escrever consultas e filtrar, consulte. Definição de consultas e filtragem

Obrigatório: Sim

`operator`

Um operador unário ou binário que ajuda a verificar o estado da consulta. O lado esquerdo (LHS) de um operador binário deve ser uma consulta e o lado direito (RHS) deve ser uma consulta ou um valor literal.

Operadores binários suportados: == (Igual) | `!` = (Diferente) | > (Maior que) | >= (Maior que ou igual a) | < (Menor que) | <= (Menor que ou igual a) | `IN` (Em uma lista no formato [x, y, z]

Operadores unários suportados: `exists` | | `empty` | `is_string` | | `is_list` | `is_struct` `not(!)`

Obrigatório: Sim

`query|value literal`

Uma consulta ou um valor literal compatível, como `string` ou`integer(64)`.

Literais de valor suportados:

- Todos os tipos primitivos:`string`,`integer(64)`,,`float(64)`,`bool`, `char` `regex`
- Todos os tipos de intervalos especializados para expressão `integer(64)float(64)`, ou `char` intervalos expressos como:
  - `r[<lower_limit>, <upper_limit>]`, que se traduz em qualquer valor k que satisfaça a seguinte expressão: `lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`, que se traduz em qualquer valor k que satisfaça a seguinte expressão: `lower_limit <= k < upper_limit`
  - `r(<lower_limit>, <upper_limit>]`, que se traduz em qualquer valor k que satisfaça a seguinte expressão: `lower_limit < k <= upper_limit`
  - `r(<lower_limit>, <upper_limit>)`,que se traduz em qualquer valor k que satisfaça a seguinte expressão: `lower_limit < k < upper_limit`
- Matrizes associativas (mapas) para dados de estrutura de valores-chave aninhados. Por exemplo:

  ```
  { "my-map": { "nested-maps": [ { "key": 10, "value": 20 } ] } }
  ```
- Matrizes de tipos primitivos ou tipos de matrizes associativas

Obrigatório: Condicional; obrigatório quando um operador binário é usado.

`custom message`

Uma string que fornece informações sobre a cláusula. A mensagem é exibida nas saídas detalhadas dos `test` comandos `validate` and e pode ser útil para entender ou depurar a avaliação de regras em dados hierárquicos.

Obrigatório: não

Usando consultas em cláusulas

Para obter informações sobre como escrever consultas, consulte Definição de consultas e filtragem e. Atribuição e referência de variáveis nas regras do Guard

Usando operadores em cláusulas

A seguir estão exemplos CloudFormation de modelos `Template-1` `Template-2` e. Para demonstrar o uso de operadores compatíveis, os exemplos de consultas e cláusulas nesta seção se referem a esses modelos de exemplo.

Modelo-1

```
Resources:
 S3Bucket:
   Type: "AWS::S3::Bucket"
   Properties:
     BucketName: "MyServiceS3Bucket"
     BucketEncryption:
       ServerSideEncryptionConfiguration:
         - ServerSideEncryptionByDefault:
             SSEAlgorithm: 'aws:kms'
             KMSMasterKeyID: 'arn:aws:kms:us-
east-1:123456789:key/056ea50b-1013-3907-8617-c93e474e400'
     Tags:
       - Key: "stage"
         Value: "prod"
       - Key: "service"
         Value: "myService"
```

Modelo-2

```
Resources:
 NewVolume:
   Type: AWS::EC2::Volume
   Properties:
     Size: 100
     VolumeType: io1
     Iops: 100
     AvailabilityZone:
       Fn::Select:
```

```
        - 0
        - Fn::GetAZs: us-east-1
    Tags:
      - Key: environment
        Value: test
  DeletionPolicy: Snapshot
```

Exemplos de cláusulas que usam operadores unários

- `empty`— Verifica se uma coleção está vazia. Você também pode usá-lo para verificar se uma consulta tem valores em dados hierárquicos porque as consultas resultam em uma coleção. Você não pode usá-lo para verificar se as consultas de valor de string têm uma string (`""`) vazia definida. Para obter mais informações, consulte Definição de consultas e filtragem.

  A cláusula a seguir verifica se o modelo tem um ou mais recursos definidos. É avaliado `PASS` porque um recurso com o ID lógico `S3Bucket` está definido em`Template-1`.

  ```
  Resources !empty
  ```

  A cláusula a seguir verifica se uma ou mais tags estão definidas para o `S3Bucket` recurso. É avaliado `PASS` porque `S3Bucket` tem duas tags definidas para a `Tags` propriedade em`Template-1`.

  ```
  Resources.S3Bucket.Properties.Tags !empty
  ```

- `exists`— Verifica se cada ocorrência da consulta tem um valor e pode ser usada no lugar de`!= null`.

  A cláusula a seguir verifica se a `BucketEncryption` propriedade está definida para o. `S3Bucket` É avaliado como `PASS` porque `BucketEncryption` está definido para `S3Bucket` em`Template-1`.

  ```
  Resources.S3Bucket.Properties.BucketEncryption exists
  ```

> **Note**
>
> As `not exists` verificações `empty` e avaliam a ausência `true` de chaves de propriedade ao percorrer os dados de entrada. Por exemplo, se a `Properties`

seção não estiver definida no modelo para o`S3Bucket`, a cláusula será `Resources.S3Bucket.Properties.Tag empty` avaliada como. `true` As `empty` verificações `exists` e não exibem o caminho do JSON ponteiro dentro do documento nas mensagens de erro. Ambas as cláusulas geralmente têm erros de recuperação que não mantêm essas informações de travessia.

- `is_string`— Verifica se cada ocorrência da consulta é do `string` tipo.

  A cláusula a seguir verifica se um valor de string foi especificado para a `BucketName` propriedade do `S3Bucket` recurso. É avaliado como `PASS` porque o valor da string `"MyServiceS3Bucket"` é especificado para `BucketName` in`Template-1`.

  ```
  Resources.S3Bucket.Properties.BucketName is_string
  ```

- `is_list`— Verifica se cada ocorrência da consulta é do `list` tipo.

  A cláusula a seguir verifica se uma lista foi especificada para a `Tags` propriedade do `S3Bucket` recurso. É avaliado como `PASS` porque dois pares de valores-chave são especificados em. `Tags` `Template-1`

  ```
  Resources.S3Bucket.Properties.Tags is_list
  ```

- `is_struct`— Verifica se cada ocorrência da consulta é um dado estruturado.

  A cláusula a seguir verifica se os dados estruturados estão especificados para a `BucketEncryption` propriedade do `S3Bucket` recurso. É avaliado como `PASS` porque `BucketEncryption` é especificado usando o tipo de `ServerSideEncryptionConfiguration` propriedade *`(object)`* em`Template-1`.

> **Note**
>
> Para verificar o estado inverso, você pode usar o operador (`not !`) com os `is_struct` operadores `is_string`, `is_list`, e.

Exemplos de cláusulas que usam operadores binários

A cláusula a seguir verifica se o valor especificado para a `BucketName` propriedade do `S3Bucket` recurso em `Template-1` contém a string`encrypt`, independentemente da maiúscula e minúscula. Isso acontece `PASS` porque o nome do bucket especificado `"MyServiceS3Bucket"` não contém a string`encrypt`.

```
Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
```

A cláusula a seguir verifica se o valor especificado para a `Size` propriedade do `NewVolume` recurso em `Template-2` está dentro de um intervalo específico: 50 <= `Size` <= 200. É avaliado como `PASS` porque `100` está especificado para`Size`.

```
Resources.NewVolume.Properties.Size IN r[50,200]
```

A cláusula a seguir verifica se o valor especificado para a `VolumeType` propriedade do `NewVolume` recurso em `Template-2` é `io1``io2`, ou`gp3`. É avaliado como `PASS` porque `io1` está especificado para`NewVolume`.

```
Resources.NewVolume.Properties.NewVolume.VolumeType IN [ 'io1','io2','gp3' ]
```

> **Note**
>
> Os exemplos de consultas nesta seção demonstram o uso de operadores usando os recursos com lógica IDs `S3Bucket` e. `NewVolume` Os nomes dos recursos geralmente são definidos pelo usuário e podem ser nomeados arbitrariamente em um modelo de infraestrutura como código (IaC). Para escrever uma regra que seja genérica e se aplique a todos os `AWS::S3::Bucket` recursos definidos no modelo, a forma mais comum de consulta usada é`Resources.*[ Type == 'AWS::S3::Bucket' ]`. Para obter mais informações, consulte Definição de consultas e filtragem para obter detalhes sobre o uso e explore o diretório de exemplos no `cloudformation-guard` GitHub repositório.

Usando mensagens personalizadas em cláusulas

No exemplo a seguir, cláusulas para `Template-2` incluir uma mensagem personalizada.

```
Resources.NewVolume.Properties.Size IN r[50,200]
```

```
<<
    EC2Volume size must be between 50 and 200,
    not including 50 and 200
>>
Resources.NewVolume.Properties.VolumeType IN [ 'io1','io2','gp3' ] <<Allowed Volume
 Types are io1, io2, and gp3>>
```

Combinando cláusulas

No Guard, cada cláusula escrita em uma nova linha é combinada implicitamente com a próxima cláusula usando conjunção (lógica booleana). `and` Veja o exemplo a seguir.

```
# clause_A ^ clause_B ^ clause_C
clause_A
clause_B
clause_C
```

Você também pode usar a disjunção para combinar uma cláusula com a próxima cláusula especificando `or|OR` no final da primeira cláusula.

```
<query> <operator> [query|value literal] [custom message] [or|OR]
```

Em uma cláusula de Guarda, as disjunções são avaliadas primeiro, seguidas pelas conjunções. As regras de proteção podem ser definidas como uma conjunção de disjunção de cláusulas (e `and|AND` de `or|OR` s) que são avaliadas como () ou `true` (PASS). `false` `FAIL` Isso é semelhante à forma normal conjuntiva.

Os exemplos a seguir demonstram a ordem das avaliações das cláusulas.

```
# (clause_E v clause_F) ^ clause_G
clause_E OR clause_F
clause_G

# (clause_H v clause_I) ^ (clause_J v clause_K)
clause_H OR
clause_I
clause_J OR
clause_K

# (clause_L v clause_M v clause_N) ^ clause_O
```

```
clause_L OR
clause_M OR
clause_N
clause_O
```

Todas as cláusulas baseadas no exemplo `Template-1` podem ser combinadas usando a conjunção. Veja o exemplo a seguir.

```
Resources.S3Bucket.Properties.BucketName is_string
Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
Resources.S3Bucket.Properties.BucketEncryption exists
Resources.S3Bucket.Properties.BucketEncryption is_struct
Resources.S3Bucket.Properties.Tags is_list
Resources.S3Bucket.Properties.Tags !empty
```

Usando blocos com as regras do Guard

Blocos são composições que removem a verbosidade e a repetição de um conjunto de cláusulas, condições ou regras relacionadas. Existem três tipos de blocos:

- Blocos de consulta
- `when`blocos
- Blocos de regras nomeadas

Blocos de consulta

A seguir estão as cláusulas baseadas no exemplo`Template-1`. A conjunção foi usada para combinar as cláusulas.

```
Resources.S3Bucket.Properties.BucketName is_string
Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
Resources.S3Bucket.Properties.BucketEncryption exists
Resources.S3Bucket.Properties.BucketEncryption is_struct
Resources.S3Bucket.Properties.Tags is_list
Resources.S3Bucket.Properties.Tags !empty
```

Partes da expressão de consulta em cada cláusula são repetidas. Você pode melhorar a composição e remover a verbosidade e a repetição de um conjunto de cláusulas relacionadas com o mesmo

caminho de consulta inicial usando um bloco de consulta. O mesmo conjunto de cláusulas pode ser escrito conforme mostrado no exemplo a seguir.

```
Resources.S3Bucket.Properties {
    BucketName is_string
    BucketName != /(?i)encrypt/
    BucketEncryption exists
    BucketEncryption is_struct
    Tags is_list
    Tags !empty
}
```

Em um bloco de consulta, a consulta anterior ao bloco define o contexto das cláusulas dentro do bloco.

Para obter mais informações sobre o uso de blocos, consulte[Composição de blocos de regras nomeadas](#).

`when`blocos

Você pode avaliar blocos condicionalmente usando `when` blocos, que assumem o seguinte formato.

```
  when <condition> {
       Guard_rule_1
       Guard_rule_2
       ...
   }
```

A `when` palavra-chave designa o início do `when` bloco. `condition`é uma regra da Guarda. O bloco só é avaliado se a avaliação da condição resultar em `true` (`PASS`).

A seguir está um exemplo de `when` bloco baseado em`Template-1`.

```
when Resources.S3Bucket.Properties.BucketName is_string {
     Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
 }
```

A cláusula dentro do `when` bloco só é avaliada se o valor especificado `BucketName` for uma string. Se o valor especificado para `BucketName` for referenciado na `Parameters` seção do modelo, conforme mostrado no exemplo a seguir, a cláusula dentro do `when` bloco não será avaliada.

```
Parameters:
   S3BucketName:
     Type: String

 Resources:
   S3Bucket:
     Type: "AWS::S3::Bucket"
     Properties:
       BucketName:
         Ref: S3BucketName
     ...
```

Blocos de regras nomeadas

Você pode atribuir um nome a um conjunto de regras (conjunto de regras) e, em seguida, referenciar esses blocos de validação modulares, chamados de blocos de regras nomeadas, em outras regras. Os blocos de regras nomeadas assumem o seguinte formato.

```
  rule <rule name> [when <condition>] {
    Guard_rule_1
    Guard_rule_2
    ...
    }
```

A `rule` palavra-chave designa o início do bloco de regras nomeadas.

`rule name`é uma string legível por humanos que identifica de forma exclusiva um bloco de regras nomeadas. É um rótulo para o conjunto de regras do Guard que ele encapsula. Nesse uso, o termo regra de proteção inclui cláusulas, blocos de consulta, blocos e `when` blocos de regras nomeadas. O nome da regra pode ser usado para se referir ao resultado da avaliação do conjunto de regras que ela encapsula, o que torna os blocos de regras nomeadas reutilizáveis. O nome da regra também fornece contexto sobre falhas de regras nas saídas do `test` comando `validate` e. O nome da regra é exibido junto com o status de avaliação do bloco (`PASS``FAIL`, ou`SKIP`) na saída de avaliação do arquivo de regras. Veja o exemplo a seguir.

```
# Sample output of an evaluation where check1, check2, and check3 are rule names.
_Summary__ __Report_ Overall File Status = **FAIL**
**PASS/****SKIP** **rules**
check1 **SKIP**
check2 **PASS**
```

```
**FAILED rules**
check3 **FAIL**
```

Você também pode avaliar blocos de regras nomeadas condicionalmente especificando a `when` palavra-chave seguida por uma condição após o nome da regra.

A seguir está o `when` bloco de exemplo que foi discutido anteriormente neste tópico.

```
rule checkBucketNameStringValue when Resources.S3Bucket.Properties.BucketName is_string
 {
    Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
}
```

Usando blocos de regras nomeadas, o precedente também pode ser escrito da seguinte forma.

```
rule checkBucketNameIsString {
    Resources.S3Bucket.Properties.BucketName is_string
}
rule checkBucketNameStringValue when checkBucketNameIsString {
    Resources.S3Bucket.Properties.BucketName != /(?i)encrypt/
}
```

Você pode reutilizar e agrupar blocos de regras nomeadas com outras regras do Guard. A seguir estão alguns exemplos.

```
rule rule_name_A {
    Guard_rule_1 OR
    Guard_rule_2
    ...
}

rule rule_name_B {
    Guard_rule_3
    Guard_rule_4
    ...
}

rule rule_name_C {
    rule_name_A OR rule_name_B
}

rule rule_name_D {
```

```
    rule_name_A
    rule_name_B
}

rule rule_name_E when rule_name_D {
    Guard_rule_5
    Guard_rule_6
    ...
}
```

Definindo consultas e filtragem do Guard

Este tópico aborda como escrever consultas e usar a filtragem ao escrever cláusulas de regras do Guard.

Pré-requisitos

A filtragem é um AWS CloudFormation Guard conceito avançado. Recomendamos que você analise os seguintes tópicos fundamentais antes de aprender sobre filtragem:

- O que AWS CloudFormation Guardé
- Regras de redação, cláusulas

Definindo consultas

As expressões de consulta são expressões simples separadas por ponto (`.`) escritas para atravessar dados hierárquicos. As expressões de consulta podem incluir expressões de filtro para direcionar um subconjunto de valores. Quando as consultas são avaliadas, elas resultam em uma coleção de valores, semelhante a um conjunto de resultados retornado de uma consulta SQL.

O exemplo de consulta a seguir pesquisa `AWS::IAM::Role` recursos em um AWS CloudFormation modelo.

```
Resources.*[ Type == 'AWS::IAM::Role' ]
```

As consultas seguem estes princípios básicos:

- Cada parte dot (`.`) da consulta percorre a hierarquia quando um termo-chave explícito é usado, como `Resources` ou `Properties.Encrypted.` Se alguma parte da consulta não corresponder ao datum de entrada, o Guard gerará um erro de recuperação.

- Uma parte dot (`.`) da consulta que usa um curinga `*` percorre todos os valores da estrutura nesse nível.
- Uma parte dot (`.`) da consulta que usa um curinga de matriz `[*]` percorre todos os índices dessa matriz.
- Todas as coleções podem ser filtradas especificando filtros dentro de colchetes. `[]` As coleções podem ser encontradas das seguintes maneiras:
  - As matrizes que ocorrem naturalmente no datum são coleções. A seguir estão exemplos da :

    Portas: `[20, 21, 110, 190]`

    Etiquetas: `[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - Ao percorrer todos os valores de uma estrutura como `Resources.*`
  - Qualquer resultado de consulta é, em si, uma coleção da qual os valores podem ser filtrados posteriormente. Veja o exemplo a seguir.

```
# Query all resources
let all_resources = Resource.*

# Filter IAM resources from query results
let iam_resources = %resources[ Type == /IAM/ ]

# Further refine to get managed policies
let managed_policies = %iam_resources[ Type == /ManagedPolicy/ ]

# Traverse each managed policy
%managed_policies {
    # Do something with each policy
}
```

Veja a seguir um exemplo de trecho CloudFormation de modelo.

```
Resources:
  SampleRole:
    Type: AWS::IAM::Role
    ...
  SampleInstance:
    Type: AWS::EC2::Instance
    ...
```

```
  SampleVPC:
     Type: AWS::EC2::VPC
    ...
  SampleSubnet1:
    Type: AWS::EC2::Subnet
    ...
  SampleSubnet2:
    Type: AWS::EC2::Subnet
    ...
```

Com base nesse modelo, o caminho percorrido é `SampleRole` e o valor final selecionado é. `Type: AWS::IAM::Role`

```
Resources:
  SampleRole:
    Type: AWS::IAM::Role
    ...
```

O valor resultante da consulta `Resources.*[ Type == 'AWS::IAM::Role' ]` no formato YAML é mostrado no exemplo a seguir.

```
- Type: AWS::IAM::Role
  ...
```

Algumas das maneiras pelas quais você pode usar consultas são as seguintes:

- Atribua uma consulta às variáveis para que os resultados da consulta possam ser acessados referenciando essas variáveis.
- Siga a consulta com um bloco que testa cada um dos valores selecionados.
- Compare uma consulta diretamente com uma cláusula básica.

Atribuição de consultas a variáveis

O Guard suporta atribuições de variáveis únicas dentro de um determinado escopo. Para obter mais informações sobre variáveis nas regras do Guard, consulteAtribuição e referência de variáveis nas regras do Guard.

Você pode atribuir consultas a variáveis para poder escrever consultas uma vez e depois referenciá-las em outro lugar nas regras do Guard. Veja o exemplo a seguir de atribuições de variáveis que demonstram os princípios de consulta discutidos posteriormente nesta seção.

```
#
# Simple query assignment
#
let resources = Resources.* # All resources

#
# A more complex query here (this will be explained below)
#
let iam_policies_allowing_log_creates = Resources.*[
    Type in [/IAM::Policy/, /IAM::ManagedPolicy/]
    some Properties.PolicyDocument.Statement[*] {
         some Action[*] == 'cloudwatch:CreateLogGroup'
         Effect == 'Allow'
    }
]
```

Percorrendo diretamente os valores de uma variável atribuída a uma consulta

O Guard suporta a execução direta dos resultados de uma consulta. No exemplo a seguir, o `when` bloco testa a `AvailabilityZone` propriedade `EncryptedVolumeType`, e para cada `AWS::EC2::Volume` recurso encontrado em um CloudFormation modelo.

```
let ec2_volumes = Resources.*[ Type == 'AWS::EC2::Volume' ]

when %ec2_volumes !empty {
    %ec2_volumes {
        Properties {
            Encrypted == true
            VolumeType in ['gp2', 'gp3']
            AvailabilityZone in ['us-west-2b', 'us-west-2c']
        }
    }
}
```

Comparações diretas em nível de cláusula

O Guard também oferece suporte a consultas como parte das comparações diretas. Por exemplo, veja o seguinte:

```
let resources = Resources.*

    some %resources.Properties.Tags[*].Key == /PROD$/
```

```
    some %resources.Properties.Tags[*].Value == /^App/
```

No exemplo anterior, as duas cláusulas (começando com a `some` palavra-chave) expressas na forma mostrada são consideradas cláusulas independentes e são avaliadas separadamente.

Formulário de cláusula única e cláusula de bloco

Juntas, as duas cláusulas de exemplo mostradas na seção anterior não são equivalentes ao bloco a seguir.

```
let resources = Resources.*

some %resources.Properties.Tags[*] {
    Key == /PROD$/
    Value == /^App/
}
```

Esse bloco consulta cada `Tag` valor na coleção e compara seus valores de propriedade com os valores de propriedade esperados. A forma combinada das cláusulas na seção anterior avalia as duas cláusulas de forma independente. Considere a seguinte entrada.

```
Resources:
  ...
  MyResource:
    ...
    Properties:
      Tags:
        - Key: EndPROD
          Value: NotAppStart
        - Key: NotPRODEnd
          Value: AppStart
```

As cláusulas na primeira forma são avaliadas como. `PASS` Ao validar a primeira cláusula na primeira forma, o caminho a seguir através de `ResourcesProperties`,,`Tags`, e `Key` corresponde ao valor `NotPRODEnd` e não corresponde ao valor esperado. `PROD`

```
Resources:
  ...
  MyResource:
    ...
    Properties:
      Tags:
```

```
        - Key: EndPROD
          Value: NotAppStart
        - Key: NotPRODEnd
          Value: AppStart
```

O mesmo acontece com a segunda cláusula do primeiro formulário. O caminho através de `ResourcesProperties`,`Tags`,, e `Value` corresponde ao valor`AppStart`. Como resultado, a segunda cláusula de forma independente.

O resultado geral é um`PASS`.

No entanto, o formulário de bloqueio é avaliado da seguinte forma. Para cada `Tags` valor, ele compara se o `Key` e `Value` corresponde; `NotAppStart` e `NotPRODEnd` os valores não são correspondidos no exemplo a seguir.

```
Resources:
  ...
  MyResource:
    ...
    Properties:
      Tags:
        - Key: EndPROD
          Value: NotAppStart
        - Key: NotPRODEnd
          Value: AppStart
```

Porque as avaliações verificam ambos e `Key == /PROD$/``Value == /^App/`, a partida não está completa. Portanto, o resultado é`FAIL`.

> **Note**
>
> Ao trabalhar com coleções, recomendamos que você use o formulário de cláusula de bloco quando quiser comparar vários valores para cada elemento na coleção. Use o formulário de cláusula única quando a coleção for um conjunto de valores escalares ou quando você pretende comparar apenas um único atributo.

Resultados da consulta e cláusulas associadas

Todas as consultas retornam uma lista de valores. Qualquer parte de uma travessia, como uma chave ausente, valores vazios para um array (`Tags: []`) ao acessar todos os índices ou valores

ausentes para um mapa ao encontrar um map (`Resources: {}`) vazio, pode levar a erros de recuperação.

Todos os erros de recuperação são considerados falhas ao avaliar as cláusulas em relação a essas consultas. A única exceção é quando filtros explícitos são usados na consulta. Quando os filtros são usados, as cláusulas associadas são ignoradas.

As seguintes falhas de bloco estão associadas à execução de consultas.

- Se um modelo não contiver recursos, a consulta será avaliada como`FAIL`, e as cláusulas de nível de bloco associadas também serão avaliadas como. `FAIL`
- Quando um modelo contém um bloco de recursos vazio`{ "Resources": {} }`, como, a consulta é avaliada como`FAIL`, e as cláusulas de nível de bloco associadas também são avaliadas como. `FAIL`
- Se um modelo contiver recursos, mas nenhum corresponder à consulta, a consulta retornará resultados vazios e as cláusulas de nível de bloco serão ignoradas.

Usando filtros em consultas

Os filtros nas consultas são efetivamente cláusulas do Guard que são usadas como critérios de seleção. A seguir está a estrutura de uma cláusula.

```
<query> <operator> [query|value literal] [message] [or|OR]
```

Lembre-se dos seguintes pontos-chave AWS CloudFormation Guard Regras de redação ao trabalhar com filtros:

- Combine cláusulas usando a Forma Normal Conjuntiva (CNF).
- Especifique cada cláusula de conjunção (`and`) em uma nova linha.
- Especifique disjunções (`or`) usando a `or` palavra-chave entre duas cláusulas.

O exemplo a seguir demonstra as cláusulas conjuntivas e disjuntivas.

```
resourceType == 'AWS::EC2::SecurityGroup'
InputParameters.TcpBlockedPorts not empty

InputParameters.TcpBlockedPorts[*] {
    this in r(100, 400] or
```

```
    this in r(4000, 65535]
}
```

Usando cláusulas para critérios de seleção

Você pode aplicar a filtragem a qualquer coleção. A filtragem pode ser aplicada diretamente em atributos na entrada que já são como `securityGroups: [....]` uma coleção. Você também pode aplicar a filtragem em uma consulta, que é sempre uma coleção de valores. Você pode usar todos os recursos das cláusulas, incluindo a forma normal conjuntiva, para filtragem.

A consulta comum a seguir é frequentemente usada ao selecionar recursos por tipo em um CloudFormation modelo.

```
Resources.*[ Type == 'AWS::IAM::Role' ]
```

A consulta `Resources.*` retorna todos os valores presentes na `Resources` seção da entrada. Para o exemplo de entrada do modelo emDefinindo consultas, a consulta retorna o seguinte.

```
- Type: AWS::IAM::Role
  ...
- Type: AWS::EC2::Instance
  ...
- Type: AWS::EC2::VPC
  ...
- Type: AWS::EC2::Subnet
  ...
- Type: AWS::EC2::Subnet
  ...
```

Agora, aplique o filtro nessa coleção. O critério de correspondência é`Type == AWS::IAM::Role`. A seguir está a saída da consulta após a aplicação do filtro.

```
- Type: AWS::IAM::Role
  ...
```

Em seguida, verifique várias cláusulas para obter `AWS::IAM::Role` recursos.

```
let all_resources = Resources.*
let all_iam_roles = %all_resources[ Type == 'AWS::IAM::Role' ]
```

Veja a seguir um exemplo de consulta de filtragem que seleciona todos `AWS::IAM::Policy` os `AWS::IAM::ManagedPolicy` recursos.

```
Resources.*[
    Type in [ /IAM::Policy/,
              /IAM::ManagedPolicy/ ]
]
```

O exemplo a seguir verifica se esses recursos de política têm um `PolicyDocument` especificado.

```
Resources.*[
    Type in [ /IAM::Policy/,
              /IAM::ManagedPolicy/ ]
    Properties.PolicyDocument exists
]
```

Criando necessidades de filtragem mais complexas

Considere o exemplo a seguir de um item de AWS Config configuração para informações de grupos de segurança de entrada e saída.

```
---
resourceType: 'AWS::EC2::SecurityGroup'
configuration:
  ipPermissions:
    - fromPort: 172
      ipProtocol: tcp
      toPort: 172
      ipv4Ranges:
        - cidrIp: 10.0.0.0/24
        - cidrIp: 0.0.0.0/0
    - fromPort: 89
      ipProtocol: tcp
      ipv6Ranges:
        - cidrIpv6: '::/0'
      toPort: 189
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 1.1.1.1/32
    - fromPort: 89
      ipProtocol: '-1'
      toPort: 189
```

```
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 1.1.1.1/32
  ipPermissionsEgress:
    - ipProtocol: '-1'
      ipv6Ranges: []
      prefixListIds: []
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 0.0.0.0/0
      ipRanges:
        - 0.0.0.0/0
  tags:
    - key: Name
      value: good-sg-delete-me
  vpcId: vpc-0123abcd
InputParameter:
  TcpBlockedPorts:
    - 3389
    - 20
    - 21
    - 110
    - 143
```

Observe o seguinte:

- `ipPermissions`(regras de entrada) é uma coleção de regras dentro de um bloco de configuração.
- Cada estrutura de regra contém atributos como `ipv4Ranges` e `ipv6Ranges` para especificar uma coleção de blocos CIDR.

Vamos escrever uma regra que seleciona todas as regras de entrada que permitem conexões de qualquer endereço IP e verifica se as regras não permitem que portas bloqueadas por TCP sejam expostas.

Comece com a parte da consulta que abrange IPv4, conforme mostrado no exemplo a seguir.

```
configuration.ipPermissions[
    #
    # at least one ipv4Ranges equals ANY IPv4
    #
```

```
    some ipv4Ranges[*].cidrIp == '0.0.0.0/0'
]
```

A `some` palavra-chave é útil nesse contexto. Todas as consultas retornam uma coleção de valores que correspondem à consulta. Por padrão, o Guard avalia se todos os valores retornados como resultado da consulta são comparados com as verificações. No entanto, esse comportamento nem sempre é o que você precisa para verificações. Considere a seguinte parte da entrada do item de configuração.

```
ipv4Ranges:
  - cidrIp: 10.0.0.0/24
  - cidrIp: 0.0.0.0/0 # any IP allowed
```

Há dois valores presentes para`ipv4Ranges`. Nem todos os `ipv4Ranges` valores são iguais a um endereço IP indicado por. `0.0.0.0/0` Você quer ver se pelo menos um valor corresponde`0.0.0.0/0`. Você diz ao Guard que nem todos os resultados retornados de uma consulta precisam corresponder, mas pelo menos um resultado deve corresponder. A `some` palavra-chave diz ao Guard que garanta que um ou mais valores da consulta resultante correspondam à verificação. Se nenhum valor do resultado da consulta corresponder, o Guard gerará um erro.

Em seguida, adicione IPv6, conforme mostrado no exemplo a seguir.

```
configuration.ipPermissions[
    #
    # at-least-one ipv4Ranges equals ANY IPv4
    #
    some ipv4Ranges[*].cidrIp == '0.0.0.0/0' or
    #
    # at-least-one ipv6Ranges contains ANY IPv6
    #
    some ipv6Ranges[*].cidrIpv6 == '::/0'
]
```

Por fim, no exemplo a seguir, confirme se o protocolo não udp é.

```
configuration.ipPermissions[
    #
    # at-least-one ipv4Ranges equals ANY IPv4
    #
    some ipv4Ranges[*].cidrIp == '0.0.0.0/0' or
```

```
    #
    # at-least-one ipv6Ranges contains ANY IPv6
    #
    some ipv6Ranges[*].cidrIpv6 == '::/0'

    #
    # and ipProtocol is not udp
    #
    ipProtocol != 'udp' ]
]
```

A seguir está a regra completa.

```
rule any_ip_ingress_checks
{

    let ports = InputParameter.TcpBlockedPorts[*]

    let targets = configuration.ipPermissions[
        #
        # if either ipv4 or ipv6 that allows access from any address
        #
        some ipv4Ranges[*].cidrIp == '0.0.0.0/0' or
        some ipv6Ranges[*].cidrIpv6 == '::/0'

        #
        # the ipProtocol is not UDP
        #
        ipProtocol != 'udp' ]

    when %targets !empty
    {
        %targets {
            ipProtocol != '-1'
            <<
              result: NON_COMPLIANT
              check_id: HUB_ID_2334
              message: Any IP Protocol is allowed
            >>

            when fromPort exists
                 toPort exists
            {
```

```
                let each_target = this
                %ports {
                    this < %each_target.fromPort or
                    this > %each_target.toPort
                    <<
                        result: NON_COMPLIANT
                        check_id: HUB_ID_2340
                        message: Blocked TCP port was allowed in range
                    >>
                }
            }

        }
    }
}
```

Separando coleções com base em seus tipos contidos

Ao usar modelos de configuração de infraestrutura como código (IaC), você pode encontrar uma coleção que contém referências a outras entidades dentro do modelo de configuração. Veja a seguir um exemplo de CloudFormation modelo que descreve as tarefas do Amazon Elastic Container Service (Amazon ECS) com uma referência local, uma referência `TaskRoleArn` a e uma referência `TaskArn` direta de string.

```
Parameters:
  TaskArn:
    Type: String
Resources:
  ecsTask:
    Type: 'AWS::ECS::TaskDefinition'
    Metadata:
      SharedExectionRole: allowed
    Properties:
      TaskRoleArn: 'arn:aws:....'
      ExecutionRoleArn: 'arn:aws:...'
  ecsTask2:
    Type: 'AWS::ECS::TaskDefinition'
    Metadata:
      SharedExectionRole: allowed
    Properties:
      TaskRoleArn:
        'Fn::GetAtt':
          - iamRole
```

```
          - Arn
      ExecutionRoleArn: 'arn:aws:...2'
  ecsTask3:
    Type: 'AWS::ECS::TaskDefinition'
    Metadata:
      SharedExectionRole: allowed
    Properties:
      TaskRoleArn:
        Ref: TaskArn
      ExecutionRoleArn: 'arn:aws:...2'
  iamRole:
    Type: 'AWS::IAM::Role'
    Properties:
      PermissionsBoundary: 'arn:aws:...3'
```

Considere a seguinte consulta.

```
let ecs_tasks = Resources.*[ Type == 'AWS::ECS::TaskDefinition' ]
```

Essa consulta retorna uma coleção de valores que contém todos os três `AWS::ECS::TaskDefinition` recursos mostrados no modelo de exemplo. Separe `ecs_tasks` os que contêm referências `TaskRoleArn` locais dos outros, conforme mostrado no exemplo a seguir.

```
let ecs_tasks = Resources.*[ Type == 'AWS::ECS::TaskDefinition' ]

let ecs_tasks_role_direct_strings = %ecs_tasks[
    Properties.TaskRoleArn is_string ]

let ecs_tasks_param_reference = %ecs_tasks[
    Properties.TaskRoleArn.'Ref' exists ]

rule task_role_from_parameter_or_string {
    %ecs_tasks_role_direct_strings !empty or
    %ecs_tasks_param_reference !empty
}

rule disallow_non_local_references {
    # Known issue for rule access: Custom message must start on the same line
    not task_role_from_parameter_or_string
    <<
        result: NON_COMPLIANT
        message: Task roles are not local to stack definition
```

```
    >>
}
```

Atribuição e referência de variáveis nas regras do Guard

Você pode atribuir variáveis em seus arquivos de AWS CloudFormation Guard regras para armazenar informações que você deseja referenciar nas regras do Guard. O Guard suporta a atribuição de variáveis com um único disparo. As variáveis são avaliadas lentamente, o que significa que o Guard só avalia as variáveis quando as regras são executadas.

Tópicos

- Atribuição de variáveis
- Variáveis de referência
- Escopo variável
- Exemplos de variáveis nos arquivos de regras do Guard

Atribuição de variáveis

Use a `let` palavra-chave para inicializar e atribuir uma variável. Como prática recomendada, use snake case para nomes de variáveis. As variáveis podem armazenar literais estáticos ou propriedades dinâmicas resultantes de consultas. No exemplo a seguir, a variável `ecs_task_definition_task_role_arn` armazena o valor da string estática`arn:aws:iam:123456789012:role/my-role-name`.

```
let ecs_task_definition_task_role_arn = 'arn:aws:iam::123456789012:role/my-role-name'
```

No exemplo a seguir, a variável `ecs_tasks` armazena os resultados de uma consulta que pesquisa todos os `AWS::ECS::TaskDefinition` recursos em um AWS CloudFormation modelo. Você pode consultar `ecs_tasks` para acessar informações sobre esses recursos ao escrever regras.

```
let ecs_tasks = Resources.*[
    Type == 'AWS::ECS::TaskDefinition'
]
```

Variáveis de referência

Use o `%` prefixo para referenciar uma variável.

Com base no exemplo da `ecs_task_definition_task_role_arn` variável emAtribuição de variáveis, você pode fazer referência `ecs_task_definition_task_role_arn` na `query|value literal` seção de uma cláusula de regra do Guard. O uso dessa referência garante que o valor especificado para a `TaskDefinitionArn` propriedade de qualquer `AWS::ECS::TaskDefinition` recurso em um CloudFormation modelo seja o valor da string estáticaarn`:aws:iam:123456789012:role/my-role-name`.

```
Resources.*.Properties.TaskDefinitionArn == %ecs_task_definition_role_arn
```

Com base no exemplo da `ecs_tasks` variável emAtribuição de variáveis, você pode fazer referência `ecs_tasks` em uma consulta (por exemplo, %ECS_tasks.properties). Primeiro, o Guard avalia a variável `ecs_tasks` e depois usa os valores retornados para percorrer a hierarquia. Se a variável for `ecs_tasks` resolvida para valores que não sejam de string, o Guard gerará um erro.

> **Note**
>
> Atualmente, o Guard não oferece suporte a variáveis de referência em mensagens de erro personalizadas.

Escopo variável

O escopo se refere à visibilidade das variáveis definidas em um arquivo de regras. Um nome de variável só pode ser usado uma vez dentro de um escopo. Há três níveis em que uma variável pode ser declarada ou três escopos de variáveis possíveis:

- Nível de arquivo — Normalmente declaradas na parte superior do arquivo de regras, você pode usar variáveis em nível de arquivo em todas as regras do arquivo de regras. Eles são visíveis em todo o arquivo.

  No arquivo de regras de exemplo a seguir, as variáveis `ecs_task_definition_task_role_arn` e `ecs_task_definition_execution_role_arn` são inicializadas no nível do arquivo.

  ```
  let ecs_task_definition_task_role_arn = 'arn:aws:iam::123456789012:role/my-task-role-name'
  let ecs_task_definition_execution_role_arn = 'arn:aws:iam::123456789012:role/my-execution-role-name'
  ```

```
rule check_ecs_task_definition_task_role_arn
{
    Resources.*.Properties.TaskRoleArn == %ecs_task_definition_task_role_arn
}

rule check_ecs_task_definition_execution_role_arn
{
    Resources.*.Properties.ExecutionRoleArn ==
 %ecs_task_definition_execution_role_arn
}
```

- Nível de regra — Declaradas em uma regra, as variáveis de nível de regra só são visíveis para essa regra específica. Qualquer referência fora da regra resultará em um erro.

  No arquivo de regras de exemplo a seguir, as variáveis `ecs_task_definition_task_role_arn` e e `ecs_task_definition_execution_role_arn` são inicializadas no nível da regra. O só `ecs_task_definition_task_role_arn` pode ser referenciado dentro da regra `check_ecs_task_definition_task_role_arn` nomeada. Você só pode referenciar a `ecs_task_definition_execution_role_arn` variável dentro da regra `check_ecs_task_definition_execution_role_arn` nomeada.

```
rule check_ecs_task_definition_task_role_arn
{
    let ecs_task_definition_task_role_arn = 'arn:aws:iam::123456789012:role/my-task-
role-name'
    Resources.*.Properties.TaskRoleArn == %ecs_task_definition_task_role_arn
}

rule check_ecs_task_definition_execution_role_arn
{
    let ecs_task_definition_execution_role_arn = 'arn:aws:iam::123456789012:role/my-
execution-role-name'
    Resources.*.Properties.ExecutionRoleArn ==
 %ecs_task_definition_execution_role_arn
}
```

- Nível do bloco — Declaradas dentro de um bloco, como uma `when` cláusula, as variáveis do nível do bloco só são visíveis para aquele bloco específico. Qualquer referência fora do bloco resultará em um erro.

No arquivo de regras de exemplo a seguir, as variáveis `ecs_task_definition_task_role_arn` e `ecs_task_definition_execution_role_arn` são inicializadas no nível do bloco dentro do `AWS::ECS::TaskDefinition` bloco de tipos. Você só pode referenciar as `ecs_task_definition_execution_role_arn` variáveis `ecs_task_definition_task_role_arn` e dentro dos blocos `AWS::ECS::TaskDefinition` de tipos para suas respectivas regras.

```
rule check_ecs_task_definition_task_role_arn
{
    AWS::ECS::TaskDefinition
    {
        let ecs_task_definition_task_role_arn = 'arn:aws:iam::123456789012:role/my-
task-role-name'
        Properties.TaskRoleArn == %ecs_task_definition_task_role_arn
    }
}

rule check_ecs_task_definition_execution_role_arn
{
    AWS::ECS::TaskDefinition
    {
        let ecs_task_definition_execution_role_arn = 'arn:aws:iam::123456789012:role/
my-execution-role-name'
        Properties.ExecutionRoleArn == %ecs_task_definition_execution_role_arn
    }
}
```

Exemplos de variáveis nos arquivos de regras do Guard

As seções a seguir fornecem exemplos de atribuição estática e dinâmica de variáveis.

Atribuição estática

Veja a seguir um exemplo CloudFormation de modelo.

```
Resources:
  EcsTask:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      TaskRoleArn: 'arn:aws:iam::123456789012:role/my-role-name'
```

Com base nesse modelo, você pode escrever uma regra chamada `check_ecs_task_definition_task_role_arn` que garante que a `TaskRoleArn` propriedade de todos os recursos do `AWS::ECS::TaskDefinition` modelo seja`arn:aws:iam::123456789012:role/my-role-name`.

```
rule check_ecs_task_definition_task_role_arn
{
    let ecs_task_definition_task_role_arn = 'arn:aws:iam::123456789012:role/my-role-
name'
    Resources.*.Properties.TaskRoleArn == %ecs_task_definition_task_role_arn
}
```

Dentro do escopo da regra, você pode inicializar uma variável chamada `ecs_task_definition_task_role_arn` e atribuir a ela o valor `'arn:aws:iam::123456789012:role/my-role-name'` da string estática. A cláusula de regra verifica se o valor especificado para a `TaskRoleArn` propriedade do `EcsTask` recurso é `arn:aws:iam::123456789012:role/my-role-name` referenciando a `ecs_task_definition_task_role_arn` variável na `query|value literal` seção.

Atribuição dinâmica

Veja a seguir um exemplo CloudFormation de modelo.

```
Resources:
  EcsTask:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      TaskRoleArn: 'arn:aws:iam::123456789012:role/my-role-name'
```

Com base nesse modelo, você pode inicializar uma variável chamada `ecs_tasks` dentro do escopo do arquivo e atribuir a ela a consulta`Resources.*[ Type == 'AWS::ECS::TaskDefinition'`. O Guard consulta todos os recursos no modelo de entrada e armazena informações sobre eles em`ecs_tasks`. Você também pode escrever uma regra chamada `check_ecs_task_definition_task_role_arn` que garanta que a `TaskRoleArn` propriedade de todos os recursos do `AWS::ECS::TaskDefinition` modelo seja `arn:aws:iam::123456789012:role/my-role-name`

```
let ecs_tasks = Resources.*[
    Type == 'AWS::ECS::TaskDefinition'
]
```

```
rule check_ecs_task_definition_task_role_arn
{
    %ecs_tasks.Properties.TaskRoleArn == 'arn:aws:iam::123456789012:role/my-role-name'
}
```

A cláusula de regra verifica se o valor especificado para a `TaskRoleArn` propriedade do `EcsTask` recurso é `arn:aws:iam::123456789012:role/my-role-name` referenciando a `ecs_task_definition_task_role_arn` variável na `query` seção.

Impondo a configuração AWS CloudFormation do modelo

Vamos dar uma olhada em um exemplo mais complexo de um caso de uso de produção. Neste exemplo, escrevemos regras do Guard para garantir controles mais rígidos sobre como ECS as tarefas da Amazon são definidas.

Veja a seguir um exemplo CloudFormation de modelo.

```
Resources:
  EcsTask:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      TaskRoleArn:
        'Fn::GetAtt': [TaskIamRole, Arn]
      ExecutionRoleArn:
        'Fn::GetAtt': [ExecutionIamRole, Arn]

  TaskIamRole:
    Type: 'AWS::IAM::Role'
    Properties:
      PermissionsBoundary: 'arn:aws:iam::123456789012:policy/MyExamplePolicy'

  ExecutionIamRole:
    Type: 'AWS::IAM::Role'
    Properties:
      PermissionsBoundary: 'arn:aws:iam::123456789012:policy/MyExamplePolicy'
```

Com base nesse modelo, escrevemos as seguintes regras para garantir que esses requisitos sejam atendidos:

- Cada `AWS::ECS::TaskDefinition` recurso no modelo tem uma função de tarefa e uma função de execução anexadas.

- As funções da tarefa e as funções de execução são AWS Identity and Access Management (IAM) funções.
- As funções são definidas no modelo.
- A `PermissionsBoundary` propriedade é especificada para cada função.

```
# Select all Amazon ECS task definition resources from the template
let ecs_tasks = Resources.*[
    Type == 'AWS::ECS::TaskDefinition'
]

# Select a subset of task definitions whose specified value for the TaskRoleArn
 property is an Fn::Gett-retrievable attribute
let task_role_refs = some %ecs_tasks.Properties.TaskRoleArn.'Fn::GetAtt'[0]

# Select a subset of TaskDefinitions whose specified value for the ExecutionRoleArn
 property is an Fn::Gett-retrievable attribute
let execution_role_refs = some %ecs_tasks.Properties.ExecutionRoleArn.'Fn::GetAtt'[0]

# Verify requirement #1
rule all_ecs_tasks_must_have_task_end_execution_roles
    when %ecs_tasks !empty
{
    %ecs_tasks.Properties {
        TaskRoleArn exists
        ExecutionRoleArn exists
    }
}

# Verify requirements #2 and #3
rule all_roles_are_local_and_type_IAM
    when all_ecs_tasks_must_have_task_end_execution_roles
{
    let task_iam_references = Resources.%task_role_refs
    let execution_iam_reference = Resources.%execution_role_refs

    when %task_iam_references !empty {
        %task_iam_references.Type == 'AWS::IAM::Role'
    }

    when %execution_iam_reference !empty {
        %execution_iam_reference.Type == 'AWS::IAM::Role'
```

```
    }
}

# Verify requirement #4
rule check_role_have_permissions_boundary
    when all_ecs_tasks_must_have_task_end_execution_roles
{
    let task_iam_references = Resources.%task_role_refs
    let execution_iam_reference = Resources.%execution_role_refs

    when %task_iam_references !empty {
        %task_iam_references.Properties.PermissionsBoundary exists
    }

    when %execution_iam_reference !empty {
        %execution_iam_reference.Properties.PermissionsBoundary exists
    }
}
```

Composição de blocos de regras nomeadas em AWS CloudFormation Guard

Ao escrever blocos de regras nomeadas usando AWS CloudFormation Guard, você pode usar os dois estilos de composição a seguir:

- Dependência condicional
- Dependência correlacional

Usar qualquer um desses estilos de composição de dependências ajuda a promover a reutilização e reduz a verbosidade e a repetição em blocos de regras nomeadas.

Tópicos

- Pré-requisitos
- Composição de dependência condicional
- Composição de dependência correlacional

Pré-requisitos

Saiba mais sobre blocos de regras nomeadas em Como escrever regras.

Composição de dependência condicional

Nesse estilo de composição, a avaliação de um `when` bloco ou bloco de regras nomeadas depende condicionalmente do resultado da avaliação de um ou mais outros blocos ou cláusulas de regras nomeadas. O exemplo de arquivo de regras do Guard a seguir contém blocos de regras nomeadas que demonstram dependências condicionais.

```
# Named-rule block, rule_name_A
rule rule_name_A {
    Guard_rule_1
    Guard_rule_2
    ...
}

# Example-1, Named-rule block, rule_name_B, takes a conditional dependency on
 rule_name_A
rule rule_name_B when rule_name_A {
    Guard_rule_3
    Guard_rule_4
    ...
}

# Example-2, when block takes a conditional dependency on rule_name_A
when rule_name_A {
    Guard_rule_3
    Guard_rule_4
    ...
}

# Example-3, Named-rule block, rule_name_C, takes a conditional dependency on
 rule_name_A ^ rule_name_B
rule rule_name_C when rule_name_A
                      rule_name_B {
    Guard_rule_3
    Guard_rule_4
    ...
}

# Example-4, Named-rule block, rule_name_D, takes a conditional dependency on
 (rule_name_A v clause_A) ^ clause_B ^ rule_name_B
rule rule_name_D when rule_name_A OR
                      clause_A
                      clause_B
```

```
                        rule_name_B {
    Guard_rule_3
    Guard_rule_4
    ...
}
```

No arquivo de regras de exemplo anterior, `Example-1` tem os seguintes resultados possíveis:

- Se for `rule_name_A` avaliado como`PASS`, as regras do Guard encapsuladas por `rule_name_B` serão avaliadas.
- Se for `rule_name_A` avaliado como`FAIL`, as regras do Guard encapsuladas por não `rule_name_B` serão avaliadas. `rule_name_B`avalia a. `SKIP`
- Se for `rule_name_A` avaliado como`SKIP`, as regras do Guard encapsuladas por não `rule_name_B` serão avaliadas. `rule_name_B`avalia a. `SKIP`

> **Note**
>
> Esse caso acontece se depender `rule_name_A` condicionalmente de uma regra que avalia `FAIL` e resulta na `rule_name_A` avaliação a. `SKIP`

A seguir está um exemplo de um item de configuração do banco de dados de gerenciamento de configuração (CMDB) de um AWS Config item para informações de grupos de segurança de entrada e saída. Este exemplo demonstra a composição da dependência condicional.

```
rule check_resource_type_and_parameter {
    resourceType == /AWS::EC2::SecurityGroup/
    InputParameters.TcpBlockedPorts NOT EMPTY
}

rule check_parameter_validity when check_resource_type_and_parameter {
    InputParameters.TcpBlockedPorts[*] {
        this in r[0,65535]
    }
}

rule check_ip_procotol_and_port_range_validity when check_parameter_validity {
    let ports = InputParameters.TcpBlockedPorts[*]

    #
```

```
    # select all ipPermission instances that can be reached by ANY IP address
    # IPv4 or IPv6 and not UDP
    #
    let configuration = configuration.ipPermissions[
        some ipv4Ranges[*].cidrIp == "0.0.0.0/0" or
        some ipv6Ranges[*].cidrIpv6 == "::/0"
        ipProtocol != 'udp' ]
    when %configuration !empty {
        %configuration {
            ipProtocol != '-1'

            when fromPort exists
                toPort exists {
                let ip_perm_block = this
                %ports {
                    this < %ip_perm_block.fromPort or
                    this > %ip_perm_block.toPort
                }
            }
        }
    }
}
```

No exemplo anterior, `check_parameter_validity` depende condicionalmente `check_resource_type_and_parameter` e depende `check_ip_procotol_and_port_range_validity` condicionalmente de. `check_parameter_validity` A seguir está um item de configuração do banco de dados de gerenciamento de configuração (CMDB) que está em conformidade com as regras anteriores.

```
---
version: '1.3'
resourceType: 'AWS::EC2::SecurityGroup'
resourceId: sg-12345678abcdefghi
configuration:
  description: Delete-me-after-testing
  groupName: good-sg-test-delete-me
  ipPermissions:
    - fromPort: 172
      ipProtocol: tcp
      ipv6Ranges: []
      prefixListIds: []
      toPort: 172
      userIdGroupPairs: []
```

```
      ipv4Ranges:
        - cidrIp: 0.0.0.0/0
      ipRanges:
        - 0.0.0.0/0
    - fromPort: 89
      ipProtocol: tcp
      ipv6Ranges:
        - cidrIpv6: '::/0'
      prefixListIds: []
      toPort: 89
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 0.0.0.0/0
      ipRanges:
        - 0.0.0.0/0
  ipPermissionsEgress:
    - ipProtocol: '-1'
      ipv6Ranges: []
      prefixListIds: []
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 0.0.0.0/0
      ipRanges:
        - 0.0.0.0/0
  tags:
    - key: Name
      value: good-sg-delete-me
  vpcId: vpc-0123abcd
InputParameters:
  TcpBlockedPorts:
    - 3389
    - 20
    - 110
    - 142
    - 1434
    - 5500
supplementaryConfiguration: {}
resourceTransitionStatus: None
```

Composição de dependência correlacional

Nesse estilo de composição, a avaliação de um `when` bloco ou bloco de regras nomeadas tem uma dependência correlacional do resultado da avaliação de uma ou mais outras regras do Guard. A dependência correlacional pode ser alcançada da seguinte forma.

```
# Named-rule block, rule_name_A, takes a correlational dependency on all of the Guard
 rules encapsulated by the named-rule block
rule rule_name_A {
    Guard_rule_1
    Guard_rule_2
    ...
}

# when block takes a correlational dependency on all of the Guard rules encapsulated by
 the when block
when condition {
    Guard_rule_1
    Guard_rule_2
    ...
}
```

Para ajudá-lo a entender a composição de dependências correlacionais, revise o exemplo a seguir de um arquivo de regras do Guard.

```
#
# Allowed valid protocols for AWS::ElasticLoadBalancingV2::Listener resources
#
let allowed_protocols = [ "HTTPS", "TLS" ]

let elbs = Resources.*[ Type == 'AWS::ElasticLoadBalancingV2::Listener' ]

#
# If there are AWS::ElasticLoadBalancingV2::Listener resources present, ensure that
 they have protocols specified from the
# list of allowed protocols and that the Certificates property is not empty
#
rule ensure_all_elbs_are_secure when %elbs !empty {
    %elbs.Properties {
        Protocol in %allowed_protocols
        Certificates !empty
    }
```

```
}

#
# In addition to secure settings, ensure that AWS::ElasticLoadBalancingV2::Listener
 resources are private
#
rule ensure_elbs_are_internal_and_secure when %elbs !empty {
    ensure_all_elbs_are_secure
    %elbs.Properties.Scheme == 'internal'
}
```

No arquivo de regras anterior, `ensure_elbs_are_internal_and_secure` tem uma dependência correlacional de. `ensure_all_elbs_are_secure` Veja a seguir um exemplo de CloudFormation modelo que está em conformidade com as regras anteriores.

```
Resources:
  ServiceLBPublicListener46709EAA:
    Type: 'AWS::ElasticLoadBalancingV2::Listener'
    Properties:
      Scheme: internal
      Protocol: HTTPS
      Certificates:
        - CertificateArn: 'arn:aws:acm...'
  ServiceLBPublicListener4670GGG:
    Type: 'AWS::ElasticLoadBalancingV2::Listener'
    Properties:
      Scheme: internal
      Protocol: HTTPS
      Certificates:
        - CertificateArn: 'arn:aws:acm...'
```

Escrevendo cláusulas para realizar avaliações contextuais

AWS CloudFormation Guard as cláusulas são avaliadas em relação aos dados hierárquicos. O mecanismo de avaliação do Guard resolve consultas em relação aos dados recebidos seguindo os dados hierárquicos conforme especificado, usando uma notação pontilhada simples. Freqüentemente, várias cláusulas são necessárias para avaliar em relação a um mapa de dados ou a uma coleção. O Guard fornece uma sintaxe conveniente para escrever essas cláusulas. O mecanismo está ciente do contexto e usa os dados correspondentes associados às avaliações.

Veja a seguir um exemplo de uma configuração do Kubernetes Pod com contêineres, na qual você pode aplicar avaliações contextuais.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: app
      image: 'images.my-company.example/app:v4'
      resources:
        requests:
          memory: 64Mi
          cpu: 0.25
        limits:
          memory: 128Mi
          cpu: 0.5
    - name: log-aggregator
      image: 'images.my-company.example/log-aggregator:v6'
      resources:
        requests:
          memory: 64Mi
          cpu: 0.25
        limits:
          memory: 128Mi
          cpu: 0.75
```

Você pode criar cláusulas do Guard para avaliar esses dados. Ao avaliar um arquivo de regras, o contexto é todo o documento de entrada. Veja a seguir exemplos de cláusulas que validam a aplicação de limites para contêineres especificados em um pod.

```
#
# At this level, the root document is available for evaluation
#

#
# Our rule only evaluates for apiVersion == v1 and K8s kind is Pod
#
rule ensure_container_limits_are_enforced
    when apiVersion == 'v1'
        kind == 'Pod'
{
```

```
    spec.containers[*] {
        resources.limits {
            #
            # Ensure that cpu attribute is set
            #
            cpu exists
            <<
                Id: K8S_REC_18
                Description: CPU limit must be set for the container
            >>

            #
            # Ensure that memory attribute is set
            #
            memory exists
            <<
                Id: K8S_REC_22
                Description: Memory limit must be set for the container
            >>
        }
    }
}
```

Compreensão **`context`** nas avaliações

No nível do bloco de regras, o contexto de entrada é o documento completo. As avaliações da `when` condição ocorrem nesse contexto raiz de entrada em que os `kind` atributos `apiVersion` e estão localizados. No exemplo anterior, essas condições são avaliadas como`true`.

Agora, percorra a hierarquia `spec.containers[*]` mostrada no exemplo anterior. Para cada travessia da hierarquia, o valor do contexto muda de acordo. Depois que a travessia do `spec` bloco é concluída, o contexto muda, conforme mostrado no exemplo a seguir.

```
containers:
  - name: app
    image: 'images.my-company.example/app:v4'
    resources:
      requests:
        memory: 64Mi
        cpu: 0.25
      limits:
        memory: 128Mi
```

```
        cpu: 0.5
  - name: log-aggregator
    image: 'images.my-company.example/log-aggregator:v6'
    resources:
      requests:
        memory: 64Mi
        cpu: 0.25
      limits:
        memory: 128Mi
        cpu: 0.75
```

Depois de percorrer o `containers` atributo, o contexto é mostrado no exemplo a seguir.

```
- name: app
  image: 'images.my-company.example/app:v4'
  resources:
    requests:
      memory: 64Mi
      cpu: 0.25
    limits:
      memory: 128Mi
      cpu: 0.5
- name: log-aggregator
  image: 'images.my-company.example/log-aggregator:v6'
  resources:
    requests:
      memory: 64Mi
      cpu: 0.25
    limits:
      memory: 128Mi
      cpu: 0.75
```

Entendendo os loops

Você pode usar a expressão `[*]` para definir um loop para todos os valores contidos na matriz do `containers` atributo. O bloco é avaliado para cada elemento interno`containers`. No trecho de regra do exemplo anterior, as cláusulas contidas no bloco definem as verificações a serem validadas em relação a uma definição de contêiner. O bloco de cláusulas contido nele é avaliado duas vezes, uma para cada definição de contêiner.

```
{
    spec.containers[*] {
```

```
        ...
    }
}
```

Para cada iteração, o valor do contexto é o valor no índice correspondente.

> **Note**
>
> O único formato de acesso ao índice suportado é `[<integer>]` ou`[*]`. Atualmente, o Guard não suporta faixas como`[2..4]`.

Matrizes

Geralmente, em locais onde uma matriz é aceita, valores únicos também são aceitos. Por exemplo, se houver apenas um contêiner, a matriz poderá ser descartada e a entrada a seguir será aceita.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: 0.25
      limits:
        memory: "128Mi"
        cpu: 0.5
```

Se um atributo puder aceitar uma matriz, certifique-se de que sua regra use o formato de matriz. No exemplo anterior, você usa `containers[*]` e não`containers`. O Guard avalia corretamente ao percorrer os dados quando encontra somente o formulário de valor único.

> **Note**
>
> Sempre use o formulário de matriz ao expressar acesso a uma cláusula de regra quando um atributo aceita uma matriz. O Guard avalia corretamente mesmo no caso de um único valor ser usado.

Usando o formulário `spec.containers[*]` em vez de `spec.containers`

As consultas de proteção retornam uma coleção de valores resolvidos. Quando você usa o formulário`spec.containers`, os valores resolvidos para a consulta contêm a matriz referida por`containers`, não os elementos dentro dela. Ao usar o formulário`spec.containers[*]`, você se refere a cada elemento individual contido. Lembre-se de usar o `[*]` formulário sempre que quiser avaliar cada elemento contido na matriz.

Usando `this` para referenciar o valor do contexto atual

Ao criar uma regra do Guard, você pode referenciar o valor do contexto usando`this`. Muitas vezes, `this` está implícito porque está vinculado ao valor do contexto. Por exemplo,`this.apiVersion`,`this.kind`, e `this.spec` estão vinculados à raiz ou ao documento. Por outro lado, `this.resources` está vinculado a cada valor para`containers`, como `/spec/containers/0/` `/spec/containers/1` e. Da mesma forma, `this.cpu` e `this.memory` mapeie os limites, especificamente `/spec/containers/0/resources/limits` `/spec/containers/1/resources/limits` e.

No próximo exemplo, a regra anterior para a configuração do Kubernetes Pod foi reescrita para ser usada explicitamente. `this`

```
rule ensure_container_limits_are_enforced
    when this.apiVersion == 'v1'
         this.kind == 'Pod'
{
    this.spec.containers[*] {
        this.resources.limits {
            #
            # Ensure that cpu attribute is set
            #
            this.cpu exists
            <<
                Id: K8S_REC_18
```

```
                Description: CPU limit must be set for the container
            >>

            #
            # Ensure that memory attribute is set
            #
            this.memory exists
            <<
                Id: K8S_REC_22
                Description: Memory limit must be set for the container
            >>
        }
    }
}
```

Você não precisa usar `this` explicitamente. No entanto, a `this` referência pode ser útil ao trabalhar com escalares, conforme mostrado no exemplo a seguir.

```
InputParameters.TcpBlockedPorts[*] {
    this in r[0, 65535)
    <<
        result: NON_COMPLIANT
        message: TcpBlockedPort not in range (0, 65535)
    >>
}
```

No exemplo anterior, `this` é usado para se referir a cada número de porta.

Possíveis erros com o uso de implícito **`this`**

Ao criar regras e cláusulas, há alguns erros comuns ao referenciar elementos do valor de contexto implícito`this`. Por exemplo, considere o seguinte dado de entrada para avaliar (isso deve ser aprovado).

```
resourceType: 'AWS::EC2::SecurityGroup'
InputParameters:
  TcpBlockedPorts: [21, 22, 110]
configuration:
  ipPermissions:
  - fromPort: 172
    ipProtocol: tcp
    ipv6Ranges: []
```

```
    prefixListIds: []
    toPort: 172
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: "0.0.0.0/0"
  - fromPort: 89
    ipProtocol: tcp
    ipv6Ranges:
      - cidrIpv6: "::/0"
    prefixListIds: []
    toPort: 109
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 10.2.0.0/24
```

Quando testada em relação ao modelo anterior, a regra a seguir resulta em um erro porque faz uma suposição incorreta de aproveitar o implícito. `this`

```
rule check_ip_procotol_and_port_range_validity
{
    #
    # select all ipPermission instances that can be reached by ANY IP address
    # IPv4 or IPv6 and not UDP
    #
    let any_ip_permissions = configuration.ipPermissions[
        some ipv4Ranges[*].cidrIp == "0.0.0.0/0" or
        some ipv6Ranges[*].cidrIpv6 == "::/0"

        ipProtocol != 'udp' ]

    when %any_ip_permissions !empty
    {
        %any_ip_permissions {
            ipProtocol != '-1' # this here refers to each ipPermission instance
            InputParameters.TcpBlockedPorts[*] {
                fromPort > this or
                toPort   < this
                <<
                    result: NON_COMPLIANT
                    message: Blocked TCP port was allowed in range
                >>
            }
        }
```

```
    }
}
```

Para ver esse exemplo, salve o arquivo de regras anterior com o nome `any_ip_ingress_check.guard` e os dados com o nome `ip_ingress.yaml` do arquivo. Em seguida, execute o `validate` comando a seguir com esses arquivos.

```
cfn-guard validate -r any_ip_ingress_check.guard -d ip_ingress.yaml --show-clause-
failures
```

Na saída a seguir, o mecanismo indica que sua tentativa de recuperar uma propriedade `InputParameters.TcpBlockedPorts[*]` no valor `/configuration/ipPermissions/0` / `configuration/ipPermissions/1` falhou.

```
Clause #2     FAIL(Block[Location[file:any_ip_ingress_check.guard, line:17,
 column:13]])

              Attempting to retrieve array index or key from map at Path = /
configuration/ipPermissions/0, Type was not an array/object map, Remaining Query =
 InputParameters.TcpBlockedPorts[*]

Clause #3     FAIL(Block[Location[file:any_ip_ingress_check.guard, line:17,
 column:13]])

              Attempting to retrieve array index or key from map at Path = /
configuration/ipPermissions/1, Type was not an array/object map, Remaining Query =
 InputParameters.TcpBlockedPorts[*]
```

Para ajudar a entender esse resultado, reescreva a regra usando referências `this` explícitas.

```
rule check_ip_procotol_and_port_range_validity
{
    #
    # select all ipPermission instances that can be reached by ANY IP address
    # IPv4 or IPv6 and not UDP
    #
    let any_ip_permissions = this.configuration.ipPermissions[
        some ipv4Ranges[*].cidrIp == "0.0.0.0/0" or
        some ipv6Ranges[*].cidrIpv6 == "::/0"

        ipProtocol != 'udp' ]
```

```
    when %any_ip_permissions !empty
    {
        %any_ip_permissions {
            this.ipProtocol != '-1' # this here refers to each ipPermission instance
            this.InputParameters.TcpBlockedPorts[*] {
                this.fromPort > this or
                this.toPort   < this
                <<
                    result: NON_COMPLIANT
                    message: Blocked TCP port was allowed in range
                >>
            }
        }
    }
}
```

`this.InputParametersfaz` referência a cada valor contido na variável`any_ip_permissions`. A consulta atribuída à variável seleciona `configuration.ipPermissions` valores que correspondem. O erro indica uma tentativa de recuperação `InputParamaters` nesse contexto, mas `InputParameters` estava no contexto raiz.

O bloco interno também faz referência a variáveis que estão fora do escopo, conforme mostrado no exemplo a seguir.

```
{
    this.ipProtocol != '-1' # this here refers to each ipPermission instance
    this.InputParameter.TcpBlockedPorts[*] { # ERROR referencing InputParameter off /
configuration/ipPermissions[*]
        this.fromPort > this or # ERROR: implicit this refers to values inside /
InputParameter/TcpBlockedPorts[*]
        this.toPort   < this
        <<
            result: NON_COMPLIANT
            message: Blocked TCP port was allowed in range
        >>
    }
}
```

`this`se refere a cada valor de porta em`[21, 22, 110]`, mas também se refere a `fromPort` `toPort` e. Ambos pertencem ao escopo do bloco externo.

Resolvendo erros com o uso implícito de **`this`**

Use variáveis para atribuir e referenciar valores explicitamente. Primeiro, `InputParameter.TcpBlockedPorts` faz parte do contexto de entrada (raiz). `InputParameter.TcpBlockedPortsSaia` do bloco interno e atribua-o explicitamente, conforme mostrado no exemplo a seguir.

```
rule check_ip_procotol_and_port_range_validity
{
     let ports = InputParameters.TcpBlockedPorts[*]
    # ... cut off for illustrating change
}
```

Em seguida, consulte essa variável explicitamente.

```
rule check_ip_procotol_and_port_range_validity
{
    #
    # Important: Assigning InputParameters.TcpBlockedPorts results in an ERROR.
    # We need to extract each port inside the array. The difference is the query
    # InputParameters.TcpBlockedPorts returns [[21, 20, 110]] whereas the query
    # InputParameters.TcpBlockedPorts[*] returns [21, 20, 110].
    #
    let ports = InputParameters.TcpBlockedPorts[*]

    #
    # select all ipPermission instances that can be reached by ANY IP address
    # IPv4 or IPv6 and not UDP
    #
    let any_ip_permissions = configuration.ipPermissions[
        some ipv4Ranges[*].cidrIp == "0.0.0.0/0" or
        some ipv6Ranges[*].cidrIpv6 == "::/0"

        ipProtocol != 'udp' ]

    when %any_ip_permissions !empty
    {
        %any_ip_permissions {
            this.ipProtocol != '-1' # this here refers to each ipPermission instance
            %ports {
                this.fromPort > this or
                this.toPort   < this
```

```
                <<
                    result: NON_COMPLIANT
                    message: Blocked TCP port was allowed in range
                >>
            }
        }
    }
}
```

Faça o mesmo com `this` as referências internas internas`%ports`.

No entanto, todos os erros ainda não foram corrigidos porque o loop interno `ports` ainda tem uma referência incorreta. O exemplo a seguir mostra a remoção da referência incorreta.

```
rule check_ip_procotol_and_port_range_validity
{
    #
    # Important: Assigning InputParameters.TcpBlockedPorts results in an ERROR.
    # We need to extract each port inside the array. The difference is the query
    # InputParameters.TcpBlockedPorts returns [[21, 20, 110]] whereas the query
    # InputParameters.TcpBlockedPorts[*] returns [21, 20, 110].
    #
    let ports = InputParameters.TcpBlockedPorts[*]

    #
    # select all ipPermission instances that can be reached by ANY IP address
    # IPv4 or IPv6 and not UDP
    #
    let any_ip_permissions = configuration.ipPermissions[
        #
        # if either ipv4 or ipv6 that allows access from any address
        #
        some ipv4Ranges[*].cidrIp == '0.0.0.0/0' or
        some ipv6Ranges[*].cidrIpv6 == '::/0'

        #
        # the ipProtocol is not UDP
        #
        ipProtocol != 'udp' ]

    when %any_ip_permissions !empty
    {
        %any_ip_permissions {
```

```
            ipProtocol != '-1'
            <<
              result: NON_COMPLIANT
              check_id: HUB_ID_2334
              message: Any IP Protocol is allowed
            >>

            when fromPort exists
                 toPort exists
            {
                let each_any_ip_perm = this
                %ports {
                    this < %each_any_ip_perm.fromPort or
                    this > %each_any_ip_perm.toPort
                    <<
                        result: NON_COMPLIANT
                        check_id: HUB_ID_2340
                        message: Blocked TCP port was allowed in range
                    >>
                }
            }
        }
    }
}
```

Em seguida, execute o `validate` comando novamente. Desta vez, passa.

```
cfn-guard validate -r any_ip_ingress_check.guard -d ip_ingress.yaml --show-clause-
failures
```

A seguir está a saída do `validate` comando.

```
Summary Report Overall File Status = PASS
PASS/SKIP rules
check_ip_procotol_and_port_range_validity    PASS
```

Para testar essa abordagem em busca de falhas, o exemplo a seguir usa uma alteração na carga útil.

```
resourceType: 'AWS::EC2::SecurityGroup'
InputParameters:
  TcpBlockedPorts: [21, 22, 90, 110]
```

```
configuration:
  ipPermissions:
    - fromPort: 172
      ipProtocol: tcp
      ipv6Ranges: []
      prefixListIds: []
      toPort: 172
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: "0.0.0.0/0"
    - fromPort: 89
      ipProtocol: tcp
      ipv6Ranges:
        - cidrIpv6: "::/0"
      prefixListIds: []
      toPort: 109
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 10.2.0.0/24
```

90 está dentro do intervalo de 89 a 109 que tem qualquer IPv6 endereço permitido. A seguir está a saída do `validate` comando depois de executá-lo novamente.

```
Clause #3           FAIL(Clause(Location[file:any_ip_ingress_check.guard, line:43,
 column:21], Check: _  LESS THAN %each_any_ip_perm.fromPort))
                    Comparing Int((Path("/InputParameters/TcpBlockedPorts/2"), 90))
 with Int((Path("/configuration/ipPermissions/1/fromPort"), 89)) failed
                    (DEFAULT: NO_MESSAGE)
Clause #4           FAIL(Clause(Location[file:any_ip_ingress_check.guard, line:44,
 column:21], Check: _  GREATER THAN %each_any_ip_perm.toPort))
                    Comparing Int((Path("/InputParameters/TcpBlockedPorts/2"), 90))
 with Int((Path("/configuration/ipPermissions/1/toPort"), 109)) failed

                                                 result: NON_COMPLIANT
                                                 check_id: HUB_ID_2340
                                                 message: Blocked TCP port was allowed in
 range
```

AWS CloudFormation Guard Regras de teste

Você pode usar a estrutura de teste de unidade AWS CloudFormation Guard integrada para verificar se as regras do Guard funcionam conforme o esperado. Esta seção fornece uma explicação passo

a passo sobre como escrever um arquivo de teste unitário e como usá-lo para testar seu arquivo de regras com o `test` comando.

Seu arquivo de teste de unidade deve ter uma das seguintes extensões: `.json`, `.JSON`, `.jsn`, `.yaml`, `.YAML`, ou `.yml`.

Tópicos

- Pré-requisitos
- Visão geral dos arquivos de teste unitário do Guard
- Passo a passo da criação de um arquivo de teste unitário de regras do Guard

Pré-requisitos

Escreva as regras do Guard para avaliar seus dados de entrada. Para obter mais informações, consulte Regras do Writing Guard.

Visão geral dos arquivos de teste unitário do Guard

Os arquivos de teste de unidade do Guard são JSON arquivos YAML formatados ou que contêm várias entradas e os resultados esperados para regras escritas dentro de um arquivo de regras do Guard. Pode haver várias amostras para avaliar diferentes expectativas. Recomendamos que você comece testando entradas vazias e, em seguida, adicione progressivamente informações para avaliar várias regras e cláusulas.

Além disso, recomendamos que você nomeie os arquivos de teste de unidade usando o sufixo `_test.json` ou `_tests.yaml`. Por exemplo, se você tiver um arquivo de regras chamado `my_rules.guard`, nomeie seu arquivo de teste de unidade `my_rules_tests.yaml`.

Sintaxe

O seguinte mostra a sintaxe de um arquivo de teste de unidade em YAML formato.

```
---
- name: <TEST NAME>
  input:
     <SAMPLE INPUT>
   expectations:
     rules:
       <RULE NAME>: [PASS|FAIL|SKIP]
```

Propriedades

A seguir estão as propriedades de um arquivo de teste do Guard.

`input`

Dados para testar suas regras. Recomendamos que seu primeiro teste use uma entrada vazia, conforme mostrado no exemplo a seguir.

```
---
- name: MyTest1
  input {}
```

Para testes subsequentes, adicione dados de entrada para testar.

Obrigatório: Sim

`expectations`

O resultado esperado quando regras específicas são avaliadas em relação aos dados de entrada. Especifique uma ou várias regras que você deseja testar além do resultado esperado para cada regra. O resultado esperado deve ser um dos seguintes:

- `PASS`— Quando executadas com base em seus dados de entrada, as regras são avaliadas como`true`.
- `FAIL`— Quando executadas com base em seus dados de entrada, as regras são avaliadas como`false`.
- `SKIP`— Quando executada com base nos dados de entrada, a regra não é acionada.

```
expectations:
    rules:
      check_rest_api_is_private: PASS
```

Obrigatório: Sim

Passo a passo da criação de um arquivo de teste unitário de regras do Guard

A seguir está um arquivo de regras chamado`api_gateway_private.guard`. A intenção dessa regra é verificar se todos os tipos de recursos do Amazon API Gateway definidos em um

CloudFormation modelo são implantados somente para acesso privado. Ele também verifica se pelo menos uma declaração de política permite acesso a partir de uma nuvem privada virtual (VPC).

```
#
# Select all AWS::ApiGateway::RestApi resources
#     present in the Resources section of the template.
#
let api_gws = Resources.*[ Type == 'AWS::ApiGateway::RestApi']

#
# Rule intent:
# 1) All AWS::ApiGateway::RestApi resources deployed must be private.

# 2) All AWS::ApiGateway::RestApi resources deployed must have at least one AWS
 Identity and Access Management (IAM) policy condition key to allow access from a VPC.
#
# Expectations:
# 1) SKIP when there are no AWS::ApiGateway::RestApi resources in the template.
# 2) PASS when:
#     ALL AWS::ApiGateway::RestApi resources in the template have
 the EndpointConfiguration property set to Type: PRIVATE.
#     ALL AWS::ApiGateway::RestApi resources in the template have one IAM condition key
 specified in the Policy property with aws:sourceVpc or :SourceVpc.
# 3) FAIL otherwise.

#
#

rule check_rest_api_is_private when %api_gws !empty {
    %api_gws {
        Properties.EndpointConfiguration.Types[*] == "PRIVATE"

    }
}

rule check_rest_api_has_vpc_access when check_rest_api_is_private {
    %api_gws {
        Properties {
            #
            # ALL AWS::ApiGateway::RestApi resources in the template have one IAM
 condition key specified in the Policy property with
            #     aws:sourceVpc or :SourceVpc
            #
```

```
            some Policy.Statement[*] {
                Condition.*[ keys == /aws:[sS]ource(Vpc|VPC|Vpce|VPCE)/ ] !empty
            }
        }
    }
}
```

Este passo a passo testa a intenção da primeira regra: todos os `AWS::ApiGateway::RestApi` recursos implantados devem ser privados.

1. Crie um arquivo de teste unitário chamado `api_gateway_private_tests.yaml` que contenha o seguinte teste inicial. Com o teste inicial, adicione uma entrada vazia e espere que a regra `check_rest_api_is_private` seja ignorada porque não há `AWS::ApiGateway::RestApi` recursos como entradas.

   ```
   ---
   - name: MyTest1
     input: {}
     expectations:
       rules:
         check_rest_api_is_private: SKIP
   ```

2. Execute o primeiro teste em seu terminal usando o `test` comando. Para o `--rules-file` parâmetro, especifique seu arquivo de regras. Para o `--test-data` parâmetro, especifique seu arquivo de teste de unidade.

   ```
   cfn-guard test \
    --rules-file api_gateway_private.guard \
    --test-data api_gateway_private_tests.yaml \
   ```

   O resultado do primeiro teste é`PASS`.

   ```
   Test Case #1
   Name: "MyTest1"
     PASS Rules:
       check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP
   ```

3. Adicione outro teste ao seu arquivo de teste unitário. Agora, estenda o teste para incluir recursos vazios. A seguir está o `api_gateway_private_tests.yaml` arquivo atualizado.

   ```
   ---
   ```

```
- name: MyTest1
  input: {}
  expectations:
    rules:
      check_rest_api_is_private: SKIP
- name: MyTest2
  input:
     Resources: {}
  expectations:
    rules:
      check_rest_api_is_private: SKIP
```

4. Execute `test` com o arquivo de teste de unidade atualizado.

```
cfn-guard test \
 --rules-file api_gateway_private.guard \
 --test-data api_gateway_private_tests.yaml \
```

O resultado do segundo teste é`PASS`.

```
Test Case #1
Name: "MyTest1"
  PASS Rules:
    check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP
Test Case #2
Name: "MyTest2"
  PASS Rules:
    check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP
```

5. Adicione mais dois testes ao seu arquivo de teste unitário. Estenda o teste para incluir o seguinte:

   - Um `AWS::ApiGateway::RestApi` recurso sem propriedades especificadas.

> **Note**
>
> Esse não é um CloudFormation modelo válido, mas é útil testar se a regra funciona corretamente mesmo para entradas malformadas.

Espere que esse teste falhe porque a `EndpointConfiguration` propriedade não foi especificada e, portanto, não está definida como`PRIVATE`.

- Um `AWS::ApiGateway::RestApi` recurso que satisfaz a primeira intenção com a `EndpointConfiguration` propriedade definida como`PRIVATE`, mas não satisfaz a segunda intenção porque não tem declarações de política definidas. Espere que esse teste passe.

A seguir está o arquivo de teste unitário atualizado.

```
---
- name: MyTest1
  input: {}
  expectations:
    rules:
      check_rest_api_is_private: SKIP
- name: MyTest2
  input:
     Resources: {}
  expectations:
    rules:
      check_rest_api_is_private: SKIP
- name: MyTest3
  input:
    Resources:
      apiGw:
        Type: AWS::ApiGateway::RestApi
  expectations:
    rules:
      check_rest_api_is_private: FAIL
- name: MyTest4
  input:
    Resources:
      apiGw:
        Type: AWS::ApiGateway::RestApi
        Properties:
          EndpointConfiguration:
            Types: "PRIVATE"
  expectations:
    rules:
      check_rest_api_is_private: PASS
```

6. Execute `test` com o arquivo de teste de unidade atualizado.

```
cfn-guard test \
 --rules-file api_gateway_private.guard \
 --test-data api_gateway_private_tests.yaml \
```

O terceiro resultado é`FAIL`, e o quarto resultado é`PASS`.

```
Test Case #1
Name: "MyTest1"
  PASS Rules:
    check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP

Test Case #2
Name: "MyTest2"
  PASS Rules:
    check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP

Test Case #3
Name: "MyTest3"
  PASS Rules:
    check_rest_api_is_private: Expected = FAIL, Evaluated = FAIL

Test Case #4
Name: "MyTest4"
  PASS Rules:
    check_rest_api_is_private: Expected = PASS, Evaluated = PASS
```

7. Comente os testes 1—3 em seu arquivo de teste unitário. Acesse o contexto detalhado somente para o quarto teste. A seguir está o arquivo de teste unitário atualizado.

```
---
#- name: MyTest1
#  input: {}
#  expectations:
#    rules:
#      check_rest_api_is_private_and_has_access: SKIP
#- name: MyTest2
#  input:
#     Resources: {}
#  expectations:
#    rules:
```

```
#      check_rest_api_is_private_and_has_access: SKIP
#- name: MyTest3
#  input:
#    Resources:
#      apiGw:
#        Type: AWS::ApiGateway::RestApi
#  expectations:
#    rules:
#      check_rest_api_is_private_and_has_access: FAIL
- name: MyTest4
  input:
    Resources:
      apiGw:
        Type: AWS::ApiGateway::RestApi
        Properties:
          EndpointConfiguration:
            Types: "PRIVATE"
  expectations:
    rules:
      check_rest_api_is_private: PASS
```

8. Inspecione os resultados da avaliação executando o `test` comando em seu terminal, usando o `--verbose` sinalizador. O contexto detalhado é útil para entender as avaliações. Nesse caso, ele fornece informações detalhadas sobre por que o quarto teste teve sucesso com um `PASS` resultado.

```
cfn-guard test \
  --rules-file api_gateway_private.guard \
  --test-data api_gateway_private_tests.yaml \
  --verbose
```

Aqui está a saída dessa execução.

```
Test Case #1
Name: "MyTest4"
  PASS Rules:
    check_rest_api_is_private: Expected = PASS, Evaluated = PASS
Rule(check_rest_api_is_private, PASS)
    |  Message: DEFAULT MESSAGE(PASS)
    Condition(check_rest_api_is_private, PASS)
        |  Message: DEFAULT MESSAGE(PASS)
```

```
        Clause(Clause(Location[file:api_gateway_private.guard, line:20, column:37],
 Check: %api_gws NOT EMPTY ), PASS)
            |  From: Map((Path("/Resources/apiGw"), MapValue { keys:
 [String((Path("/Resources/apiGw/Type"), "Type")), String((Path("/Resources/
apiGw/Properties"), "Properties"))], values: {"Type": String((Path("/Resources/
apiGw/Type"), "AWS::ApiGateway::RestApi")), "Properties": Map((Path("/
Resources/apiGw/Properties"), MapValue { keys: [String((Path("/Resources/
apiGw/Properties/EndpointConfiguration"), "EndpointConfiguration"))],
 values: {"EndpointConfiguration": Map((Path("/Resources/apiGw/Properties/
EndpointConfiguration"), MapValue { keys: [String((Path("/Resources/apiGw/
Properties/EndpointConfiguration/Types"), "Types"))], values: {"Types":
 String((Path("/Resources/apiGw/Properties/EndpointConfiguration/Types"),
 "PRIVATE"))} }))} }))} }))
            |  Message: (DEFAULT: NO_MESSAGE)
    Conjunction(cfn_guard::rules::exprs::GuardClause, PASS)
        |  Message: DEFAULT MESSAGE(PASS)
        Clause(Clause(Location[file:api_gateway_private.guard, line:22, column:5],
 Check: Properties.EndpointConfiguration.Types[*]  EQUALS String("PRIVATE")), PASS)
            |  Message: (DEFAULT: NO_MESSAGE)
```

A principal observação da saída é a linha`Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`, que indica que a verificação foi aprovada. O exemplo também mostrou o caso em que se esperava que `Types` fosse uma matriz, mas um único valor foi fornecido. Nesse caso, a Guard continuou avaliando e fornecendo um resultado correto.

9. Adicione um caso de teste como o quarto caso de teste ao seu arquivo de teste de unidade para um `AWS::ApiGateway::RestApi` recurso com a `EndpointConfiguration` propriedade especificada. O caso de teste falhará em vez de ser aprovado. A seguir está o arquivo de teste unitário atualizado.

```
---
#- name: MyTest1
#  input: {}
#  expectations:
#    rules:
#      check_rest_api_is_private_and_has_access: SKIP
#- name: MyTest2
#  input:
#     Resources: {}
#  expectations:
```

```
#    rules:
#      check_rest_api_is_private_and_has_access: SKIP
#- name: MyTest3
#  input:
#    Resources:
#      apiGw:
#        Type: AWS::ApiGateway::RestApi
#  expectations:
#    rules:
#      check_rest_api_is_private_and_has_access: FAIL
#- name: MyTest4
#  input:
#    Resources:
#      apiGw:
#        Type: AWS::ApiGateway::RestApi
#        Properties:
#          EndpointConfiguration:
#            Types: "PRIVATE"
#  expectations:
#    rules:
#      check_rest_api_is_private: PASS
- name: MyTest5
  input:
    Resources:
      apiGw:
        Type: AWS::ApiGateway::RestApi
        Properties:
          EndpointConfiguration:
            Types: [PRIVATE, REGIONAL]
  expectations:
    rules:
      check_rest_api_is_private: FAIL
```

10. Execute o `test` comando com o arquivo de teste de unidade atualizado usando o `--verbose` sinalizador.

```
cfn-guard test \
 --rules-file api_gateway_private.guard \
 --test-data api_gateway_private_tests.yaml \
 --verbose
```

O `FAIL` resultado é o esperado porque `REGIONAL` está especificado`EndpointConfiguration`, mas não é esperado.

```
Test Case #1
Name: "MyTest5"
  PASS Rules:
    check_rest_api_is_private: Expected = FAIL, Evaluated = FAIL
Rule(check_rest_api_is_private, FAIL)
    |  Message: DEFAULT MESSAGE(FAIL)
    Condition(check_rest_api_is_private, PASS)
        |  Message: DEFAULT MESSAGE(PASS)
        Clause(Clause(Location[file:api_gateway_private.guard, line:20, column:37],
 Check: %api_gws NOT EMPTY ), PASS)
            |  From: Map((Path("/Resources/apiGw"), MapValue { keys:
 [String((Path("/Resources/apiGw/Type"), "Type")), String((Path("/Resources/
apiGw/Properties"), "Properties"))], values: {"Type": String((Path("/Resources/
apiGw/Type"), "AWS::ApiGateway::RestApi")), "Properties": Map((Path("/
Resources/apiGw/Properties"), MapValue { keys: [String((Path("/Resources/
apiGw/Properties/EndpointConfiguration"), "EndpointConfiguration"))],
 values: {"EndpointConfiguration": Map((Path("/Resources/apiGw/Properties/
EndpointConfiguration"), MapValue { keys: [String((Path("/Resources/apiGw/
Properties/EndpointConfiguration/Types"), "Types"))], values: {"Types":
 List((Path("/Resources/apiGw/Properties/EndpointConfiguration/Types"),
 [String((Path("/Resources/apiGw/Properties/EndpointConfiguration/Types/0"),
 "PRIVATE")), String((Path("/Resources/apiGw/Properties/EndpointConfiguration/
Types/1"), "REGIONAL"))]))} }))} }))} }))
            |  Message: DEFAULT MESSAGE(PASS)
    BlockClause(Block[Location[file:api_gateway_private.guard, line:21, column:3]],
 FAIL)
        |  Message: DEFAULT MESSAGE(FAIL)
        Conjunction(cfn_guard::rules::exprs::GuardClause, FAIL)
            |  Message: DEFAULT MESSAGE(FAIL)
            Clause(Clause(Location[file:api_gateway_private.guard, line:22,
 column:5], Check: Properties.EndpointConfiguration.Types[*]  EQUALS
 String("PRIVATE")), FAIL)
                |  From: String((Path("/Resources/apiGw/Properties/
EndpointConfiguration/Types/1"), "REGIONAL"))
                |  To: String((Path("api_gateway_private.guard/22/5/Clause/"),
 "PRIVATE"))
                |  Message: (DEFAULT: NO_MESSAGE)
```

A saída detalhada do `test` comando segue a estrutura do arquivo de regras. Cada bloco no arquivo de regras é um bloco na saída detalhada. O bloco mais alto é cada regra. Se houver

when condições contrárias à regra, elas aparecerão em um bloco de condições de irmãos. No exemplo a seguir, a condição `%api_gws !empty` é testada e aprovada.

```
rule check_rest_api_is_private when %api_gws !empty {
```

Depois que a condição for aprovada, testamos as cláusulas da regra.

```
%api_gws {
    Properties.EndpointConfiguration.Types[*] == "PRIVATE"
}
```

`%api_gws`é uma regra de bloqueio que corresponde ao `BlockClause` nível na saída (linha: 21). A cláusula de regra é um conjunto de cláusulas de conjunção (AND), em que cada cláusula de conjunção é um conjunto de disjunções. `OR` A conjunção tem uma única cláusula,. `Properties.EndpointConfiguration.Types[*] == "PRIVATE"` Portanto, a saída detalhada mostra uma única cláusula. O caminho `/Resources/apiGw/Properties/EndpointConfiguration/Types/1` mostra quais valores na entrada são comparados, que nesse caso é o elemento `Types` indexado em 1.

EmValidando os dados de entrada de acordo com as regras do Guard, você pode usar os exemplos desta seção para usar o `validate` comando para avaliar os dados de entrada em relação às regras.

Usando parâmetros de entrada com AWS CloudFormation Guard regras

AWS CloudFormation Guard permite que você use parâmetros de entrada para pesquisas dinâmicas de dados durante a validação. Esse recurso é particularmente útil quando você precisa referenciar dados externos em suas regras. No entanto, ao especificar as chaves dos parâmetros de entrada, o Guard exige que não haja caminhos conflitantes.

Como usar

1. Use o `-i` sinalizador `--input-parameters` ou para especificar arquivos contendo parâmetros de entrada. Vários arquivos de parâmetros de entrada podem ser especificados e combinados para formar um contexto comum. As teclas de parâmetros de entrada não podem ter caminhos conflitantes.

2. Use o `-d` sinalizador `--data` ou para especificar o arquivo de modelo real a ser validado.

Exemplo de uso

1. Crie um arquivo de parâmetros de entrada (por exemplo,`network.yaml`):

```
NETWORK:
  allowed_security_groups: ["sg-282850", "sg-292040"]
  allowed_prefix_lists: ["pl-63a5400a", "pl-02cd2c6b"]
```

2. Faça referência a esses parâmetros em seu arquivo de regras de proteção (por exemplo,`security_groups.guard`):

```
let groups = Resources.*[ Type == 'AWS::EC2::SecurityGroup' ]

let permitted_sgs = NETWORK.allowed_security_groups
let permitted_pls = NETWORK.allowed_prefix_lists
rule check_permitted_security_groups_or_prefix_lists(groups) {
    %groups {
        this in %permitted_sgs or
        this in %permitted_pls
    }
}

rule CHECK_PERMITTED_GROUPS when %groups !empty {
    check_permitted_security_groups_or_prefix_lists(
       %groups.Properties.GroupName
    )
}
```

3. Crie um modelo de dados com falha (por exemplo,`security_groups_fail.yaml`):

```
# ---
# AWSTemplateFormatVersion: 2010-09-09
# Description: CloudFormation - EC2 Security Group

Resources:
  mySecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupName: "wrong"
```

4. Execute o comando validate:

```
cfn-guard validate -r security_groups.guard -i network.yaml -d
 security_groups_fail.yaml
```

Neste comando:

- `-r`especifica o arquivo de regras
- `-i`especifica o arquivo de parâmetros de entrada
- `-d`especifica o arquivo de dados (modelo) a ser validado

Vários parâmetros de entrada

Você pode especificar vários arquivos de parâmetros de entrada:

```
cfn-guard validate -r rules.guard -i params1.yaml -i params2.yaml -d template.yaml
```

Todos os arquivos especificados com `-i` serão combinados para formar um único contexto para pesquisa de parâmetros.

Validando dados de entrada em relação às regras AWS CloudFormation Guard

Você pode usar o AWS CloudFormation Guard `validate` comando para validar dados de acordo com as regras do Guard. Para obter mais informações sobre o `validate` comando, incluindo seus parâmetros e opções, consulte validate.

Pré-requisitos

- Escreva as regras do Guard para validar seus dados de entrada. Para obter mais informações, consulte Regras do Writing Guard.
- Teste suas regras para garantir que elas funcionem conforme o esperado. Para obter mais informações, consulte Testando as regras do Guard.

Usar o comando `validate`

Para validar seus dados de entrada em relação às regras do Guard, como um AWS CloudFormation modelo, execute o `validate` comando Guard. Para o `--rules` parâmetro, especifique o nome de um arquivo de regras. Para o `--data` parâmetro, especifique o nome do arquivo de dados de entrada.

```
cfn-guard validate \
  --rules rules.guard \
  --data template.json
```

Se o Guard validar com êxito os modelos, o `validate` comando retornará um status de saída de `0` (`$?`em bash). Se o Guard identificar uma violação de regra, o `validate` comando retornará um relatório de status das regras que falharam. Use o sinalizador de resumo (`-s all`) para ver a árvore de avaliação detalhada que mostra como o Guard avaliou cada regra.

```
template.json Status = PASS / SKIP
PASS/SKIP rules
rules.guard/rule    PASS
```

Validando várias regras em relação a vários arquivos de dados

Para ajudar a manter as regras, você pode escrever regras em vários arquivos e organizá-las como quiser. Em seguida, você pode validar vários arquivos de regras em relação a um arquivo de dados ou vários arquivos de dados. O `validate` comando pode usar um diretório de arquivos para as `--rules` opções `--data` e. Por exemplo, você pode executar o comando a seguir, onde `/path/to/dataDirectory` contém um ou mais arquivos de dados e `/path/to/ruleDirectory` contém um ou mais arquivos de regras.

```
cfn-guard validate --data /path/to/dataDirectory --rules /path/to/ruleDirectory
```

Você pode criar regras para verificar se vários recursos definidos em vários CloudFormation modelos têm as atribuições de propriedades apropriadas para garantir a criptografia em repouso. Para facilitar a pesquisa e a manutenção, você pode ter regras para verificar a criptografia em repouso em cada recurso em arquivos separados`s3_bucket_encryption.guard`, chamados `deec2_volume_encryption.guard`, e `rds_dbinstance_encrytion.guard` em um diretório com o caminho`~/GuardRules/encryption_at_rest`. Os CloudFormation modelos que você

precisa validar estão em um diretório com o caminho`~/CloudFormation/templates`. Nesse caso, execute o `validate` comando da seguinte maneira.

```
cfn-guard validate --data ~/CloudFormation/templates --rules ~/GuardRules/
encryption_at_rest
```

Solução de problemas AWS CloudFormation Guard

Se você encontrar problemas ao trabalhar com AWS CloudFormation Guard, consulte os tópicos desta seção.

Tópicos

- A cláusula falha quando nenhum recurso do tipo selecionado está presente
- O Guard não avalia o CloudFormation modelo com referências abreviadas Fn::GetAtt
- Tópicos gerais de solução de problemas

A cláusula falha quando nenhum recurso do tipo selecionado está presente

Quando uma consulta usa um filtro`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`, como, se não houver `AWS::ApiGateway::RestApi` recursos na entrada, a cláusula é avaliada como. `FAIL`

```
%api_gws.Properties.EndpointConfiguration.Types[*] == "PRIVATE"
```

Para evitar esse resultado, atribua filtros às variáveis e use a verificação de `when` condição.

```
let api_gws = Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]
    when %api_gws !empty { ...}
```

O Guard não avalia o CloudFormation modelo com referências abreviadas Fn::GetAtt

O Guard não suporta as formas abreviadas de funções intrínsecas. Por exemplo, o uso de`!Join`, `!Sub` em um AWS CloudFormation modelo YAML formatado não é suportado. Em vez disso, use as formas expandidas das funções CloudFormation intrínsecas. Por exemplo, use`Fn::Join`, `Fn::Sub` em CloudFormation modelos YAML formatados ao avaliá-los de acordo com as regras do Guard.

Para obter mais informações sobre funções intrínsecas, consulte a referência da função intrínseca no Guia do usuário.AWS CloudFormation

Tópicos gerais de solução de problemas

- Verifique se os `string` literais não contêm cadeias de caracteres de escape incorporadas. Atualmente, o Guard não oferece suporte a cadeias de escape incorporadas em `string` literais.
- Verifique se suas `!=` comparações comparam tipos de dados compatíveis. Por exemplo, a `string` e an não `int` são tipos de dados compatíveis para comparação. Ao realizar a `!=` comparação, se os valores forem incompatíveis, ocorrerá um erro internamente. Atualmente, o erro é suprimido e convertido `false` para satisfazer a PartialEqcaracterística em Rust.

AWS CloudFormation Guard CLIparâmetros e referência de comandos

Os seguintes parâmetros e comandos globais estão disponíveis por meio da interface de linha de AWS CloudFormation Guard comando (CLI).

Tópicos

- Proteja os parâmetros CLI globais
- árvore de análise
- régua
- teste
- validar

Proteja os parâmetros CLI globais

Você pode usar os parâmetros a seguir com qualquer AWS CloudFormation Guard CLI comando.

`-h`, `--help`

Imprime informações de ajuda.

`-V`, `--version`

Imprime as informações da versão.

árvore de análise

Gera uma árvore de análise para as AWS CloudFormation Guard regras definidas em um arquivo de regras.

Sintaxe

```
cfn-guard parse-tree
--output <value>
--rules <value>
```

Parâmetros

`-h`, `--help`

Imprime informações de ajuda.

`-j`, `--print-json`

Imprime a saída em JSON formato.

`-y`, `--print-yaml`

Imprime a saída em YAML formato.

`-V`, `--version`

Imprime as informações da versão.

Opções

`-o`, `--output`

Grava a árvore gerada em um arquivo de saída.

`-r`, `--rules`

Fornece um arquivo de regras.

Exemplos

```
cfn-guard parse-tree \
--output output.json \
--rules rules.guard
```

régua

Pega um arquivo de AWS CloudFormation modelo JSON - ou YAML -formatado e gera automaticamente um conjunto de AWS CloudFormation Guard regras que correspondem às propriedades dos recursos do modelo. Esse comando é uma forma útil de começar a escrever regras ou criar ready-to-use regras a partir de modelos conhecidos em boas condições.

Sintaxe

```
cfn-guard rulegen
--output <value>
--template <value>
```

Parâmetros

`-h`, `--help`

Imprime informações de ajuda.

`-V`, `--version`

Imprime as informações da versão.

Opções

`-o`, `--output`

Grava as regras geradas em um arquivo de saída. Dada a possibilidade de surgirem centenas ou até milhares de regras, recomendamos o uso dessa opção.

`-t`, `--template`

Fornece o caminho para um arquivo CloudFormation de modelo em JSON ou YAML formato.

Exemplos

```
cfn-guard rulegen \
--output output.json \
--template template.json
```

teste

Valida um arquivo de AWS CloudFormation Guard regras em relação a um arquivo de teste unitário do Guard no YAML formato JSON ou para determinar o sucesso de regras individuais.

Sintaxe

```
cfn-guard test
```

```
--rules-file <value>
--test-data <value>
```

Parâmetros

`-h`, `--help`

Imprime informações de ajuda.

`-m`, `--last-modified`

Classifica pelos horários da última modificação em um diretório

`-V`, `--version`

Imprime as informações da versão.

`-v`, `--verbose`

Aumenta a verbosidade da saída. Pode ser especificado várias vezes.

A saída detalhada segue a estrutura do arquivo de regras do Guard. Cada bloco no arquivo de regras é um bloco na saída detalhada. O bloco mais alto é cada regra. Se houver `when` condições contrárias à regra, elas aparecerão como um bloco de condições entre irmãos.

Opções

`-r`, `--rules-file`

Fornece o nome de um arquivo de regras.

`-t`, `--test-data`

Fornece o nome de um arquivo ou diretório para arquivos de dados em qualquer YAML formato JSON ou formato.

args

<alphabetical>

Classifica alfabeticamente dentro de um diretório.

Exemplos

```
cfn-guard test \
--rules rules.guard \
--test-data rules_tests.json
```

Saída

```
PASS|FAIL Expected Rule = rule_name, Status = SKIP|FAIL|PASS, Got Status = SKIP|FAIL|
PASS
```

Consulte também

Testando as regras do Guard

validar

Valida os dados em relação AWS CloudFormation Guard às regras para determinar o sucesso ou o fracasso.

Sintaxe

```
cfn-guard validate
--data <value>
--output-format <value>
--rules <value>
--show-summary <value>
--type <value>
```

Parâmetros

`-a`, `--alphabetical`

Valida arquivos em um diretório ordenado alfabeticamente.

`-h`, `--help`

Imprime informações de ajuda.

`-m`, `--last-modified`

Valida arquivos em um diretório ordenado pelos horários da última modificação.

`-P`, `--payload`

Permite que você forneça regras e dados no seguinte JSON formato por meio de `stdin`:

```
{"rules":["<rules 1>", "<rules 2>", ...], "data":["<data 1>", "<data 2>", ...]}
```

Por exemplo:

```
{"data": ["{\"Resources\":{\"NewVolume\":{\"Type\":\"AWS::EC2::Volume\",\"Properties
\":{\"Size\":500,\"Encrypted\":false,\"AvailabilityZone\":\"us-west-2b\"}},
\"NewVolume2\":{\"Type\":\"AWS::EC2::Volume\",\"Properties\":{\"Size\":50,\"Encrypted
\":false,\"AvailabilityZone\":\"us-west-2c\"}}},\"Parameters\":{\"InstanceName\":
\"TestInstance\"}}","{\"Resources\":{\"NewVolume\":{\"Type\":\"AWS::EC2::Volume\",
\"Properties\":{\"Size\":500,\"Encrypted\":false,\"AvailabilityZone\":\"us-west-2b\"}},
\"NewVolume2\":{\"Type\":\"AWS::EC2::Volume\",\"Properties\":{\"Size\":50,\"Encrypted
\":false,\"AvailabilityZone\":\"us-west-2c\"}}},\"Parameters\":{\"InstanceName
\":\"TestInstance\"}}"], "rules" : [ "Parameters.InstanceName == \"TestInstance
\"","Parameters.InstanceName == \"TestInstance\"" ]}
```

Para “regras”, especifique uma lista de strings de arquivos de regras. Para “dados”, especifique uma lista de strings de arquivos de dados.

Se você especificar o `--payload` sinalizador, não especifique as `--data` opções `--rules` ou.

`-p`, `--print-json`

Imprime a saída em JSON formato.

`-s`, `--show-clause-failures`

Mostra a falha da cláusula, incluindo um resumo.

`-V`, `--version`

Imprime as informações da versão.

`-v`, `--verbose`

Aumenta a verbosidade da saída. Pode ser especificado várias vezes.

Opções

`-d`, `--data` (sequência de caracteres)

Fornece o nome de um arquivo ou diretório para arquivos de dados em qualquer YAML formato JSON ou formato. Se você fornecer um diretório, o Guard avaliará as regras especificadas em relação a todos os arquivos de dados no diretório. O diretório deve conter somente arquivos de dados; ele não pode conter arquivos de dados e de regras.

Se você especificar o `--payload` sinalizador, não especifique a `--data` opção.

`-o`, `--output-format` (sequência de caracteres)

Grava em um arquivo de saída.

Padrão: `single-line-summary`

Valores permitidos: `json` | `yaml` | `single-line-summary`

`-r`, `--rules` (sequência de caracteres)

Fornece o nome de um arquivo de regras ou um diretório de arquivos de regras. Se você fornecer um diretório, o Guard avaliará todas as regras no diretório em relação aos dados especificados. O diretório deve conter somente arquivos de regras; ele não pode conter dados e arquivos de regras.

Se você especificar o `--payload` sinalizador, não especifique a `--rules` opção.

`--show-summary` (string)

Especifica a verbosidade do resumo da avaliação da regra do Guard. Se você especificar`all`, o Guard exibirá o resumo completo. Se você especificar`pass,fail`, o Guard exibirá somente informações resumidas das regras aprovadas ou reprovadas. Se você especificar`none`, o Guard não exibirá informações resumidas. Por padrão, `all` é especificado.

Valores permitidos: `all` | `pass,fail` | `none`

`-t`, `--type` (sequência de caracteres)

Fornece o formato dos dados de entrada. Quando você especifica o tipo de dados de entrada, o Guard exibe os nomes lógicos dos recursos do CloudFormation modelo na saída. Por padrão, o Guard exibe caminhos e valores de propriedades, como`Property [/Resources/vol2/Properties/Encrypted`.

Valores permitidos: `CFNTemplate`

Exemplos

```
cfn-guard validate \
--data file_directory_name \
--output-format yaml \
--rules rules.guard \
--show-summary pass,fail \
--type CFNtemplate
```

Saída

Se o Guard validar com êxito os modelos, o `validate` comando retornará um status de saída de `0` (`$?`em bash). Se o Guard identificar uma violação de regra, o `validate` comando retornará um relatório de status das regras que falharam. Use o sinalizador detalhado (`-v`) para ver a árvore de avaliação detalhada que mostra como o Guard avaliou cada regra.

```
Summary Report Overall File Status = PASS
PASS/SKIP rules
default PASS
```

Consulte também

Validando os dados de entrada de acordo com as regras do Guard

Segurança em AWS CloudFormation Guard

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como a segurança da nuvem e a segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam ao Guard, consulte AWS Serviços no escopo do programa de conformidade AWS .
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade dos dados, os requisitos da empresa e as leis e os regulamentos aplicáveis

A documentação a seguir ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao instalar o Guard como uma AWS Lambda função (cfn-guard-lambda):

- Segurança no Guia do AWS Command Line Interface Usuário
- Segurança no Guia do AWS Lambda Desenvolvedor
- Segurança no Guia do AWS Identity and Access Management Usuário

AWS CloudFormation Guard Histórico do documento

A tabela a seguir descreve as versões de documentação do AWS CloudFormation Guard.

- Última atualização da documentação: 30 de junho de 2023
- Última versão: 3.0.0

Alteração	Descrição	Data
Versão 3.0.0	A versão 3.0.0 apresenta os seguintes aprimoramentos: • Tópicos de introdução e instalação atualizados para a versão do Guard 3.0.0. • Instruções de instalação adicionadas para Homebrew e Chocolatey. • Informações relacionadas à migração das regras do Guard foram atualizadas para refletir as mudanças na versão 3.0.0 do Guard. • Foi adicionado um link proeminente para o AWS CloudFormation Guard GitHub repositório.	30 de junho de 2023
Lançamento da versão 2.1.3	A versão 2.1.3 apresenta os seguintes aprimoramentos: Foram adicionadas informaçõ es sobre os aprimoramentos do Guard 2.1.3. As referências	9 de junho de 2023

	ao Guard 2.0 foram atualizad as para o Guard 2.1.3.	
Lançamento da versão 2.0.4	A versão 2.0.4 apresenta os seguintes aprimoramentos: A `--payload` bandeira foi adicionada ao `validate` comando. Para obter mais informaçõ es, consulte validar na CLI referência do Guard.	19 de outubro de 2021

Lançamento da versão 2.0.3	A versão 2.0.3 apresenta os seguintes aprimoramentos:
	
	• Você pode fornecer nomes de teste para cada teste em seu arquivo de teste de unidade. Para obter mais informações, consulte Testando as regras do Guard.
	• As seguintes opções foram adicionadas ao `validate` comando:
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	
	  Para obter mais informaçõ es, consulte validar na CLI referência do Guard.
		27 de julho de 2021

Lançamento inicial	Versão inicial do Guia AWS CloudFormation Guard do usuário.	15 de julho de 2021

AWS Glossário

Para obter a AWS terminologia mais recente, consulte o AWS glossário na Glossário da AWS Referência.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.