aws

用户指南

AWS CloudFormation Guard

AWS CloudFormation Guard: 用户指南

Table of Contents

什么是 AWS CloudFormation Guard？ 1
你是第一次使用 Guard 吗？ 1
警卫功能 2
使用带 CloudFormation 钩子的防护 2
访问警卫 2
最佳实践 2
设置警卫 3
适用于 Linux 和 macOS 3
从预先构建的发行版二进制文件安装 Guard 3
安装货物防护 4
从 Homebrew 安装 Guard 5
对于 Windows： 5
先决条件 5
安装货物防护 4
安装来自 Chocolatey 的守卫 6
作为一个 AWS Lambda 函数 7
先决条件 7
安装 Rust 软件包管理器 7
将 Guard 作为 Lambda 函数安装 8
生成并运行 9
调用 Lambda 函数 9
使用 Guard 规则的先决条件和概述 10
先决条件 10
使用防护规则的概述 10
编写警卫规则 10
子句 11
在子句中使用查询 13
在子句中使用运算符 13
在子句中使用自定义消息 16
合并条款 17
使用带有守卫规则的方块 18
定义查询和筛选 21
在 Guard 规则中分配和引用变量 34
撰写命名规则块 41

编写用于执行情境感知评估的子句 47
测试警卫规则 59
先决条件 60
概述 60
演示 61
将输入参数与防护规则配合使用 70
如何使用 70
示例用法 71
多个输入参数 72
根据防护规则验证输入数据 72
先决条件 72
使用 validate 命令 72
针对多个数据文件验证多条规则 73
故障排除警卫 74
当选定类型的资源不存在时，子句失败 74
Guard 不评估 CloudFormation 模板 74
一般疑难解答主题 74
警卫CLI参考 75
保护CLI全局参数 75
解析树 75
语法 75
参数 75
选项 76
示例 76
规则 76
语法 76
参数 77
选项 77
示例 77
测试 77
语法 77
参数 78
选项 78
args 78
示例 78
输出 79

另请参阅 79
验证 79
语法 79
参数 79
选项 80
示例 81
输出 82
另请参阅 82
安全性 83
文档历史记录 84
AWS 词汇表 86
..... lxxxvii

什么是 AWS CloudFormation Guard？

AWS CloudFormation Guard 是一款开源的通用 policy-as-code评估工具。Guard 命令行界面 (CLI) 提供了一种 simple-to-use特定于域的声明性语言 (DSL)，可用于将策略表示为代码。此外，您还可以使用CLI命令根据这些规则验证结构化分层结构JSON或YAML数据。Guard 还提供了一个内置的单元测试框架，用于验证您的规则是否按预期运行。

Guard 不会验证 CloudFormation 模板的有效语法或允许的属性值。您可以使用 cfn-lint 工具对模板结构进行彻底检查。

Guard 不提供服务器端强制执行。你可以使用 CloudFormation Hook 来执行服务器端验证和强制执行，在那里你可以阻止或警告操作。

有关 AWS CloudFormation Guard 开发的详细信息，请参阅 Guard GitHub 存储库。

主题

- 你是第一次使用 Guard 吗？
- 警卫功能
- 使用带 CloudFormation 钩子的防护
- 访问警卫
- 最佳实践

你是第一次使用 Guard 吗？

如果您是首次使用 Guard，我们建议您先阅读以下章节：

- 设置警卫— 本节介绍如何安装 Guard。使用 Guard，您可以使用 Guard 编写策略规则，DSL并根据这些规则验证您的 JSON YAML-或-格式化结构化数据。
- 编写警卫规则— 本节提供了编写策略规则的详细演练。
- 测试警卫规则— 本节提供了详细的演练，用于测试您的规则以验证它们是否按预期运行，并根据您的规则验证您的 YAML-或 JSON-格式的结构化数据。
- 根据防护规则验证输入数据— 本节详细介绍了如何根据您的规则验证您的 YAML-或 JSON-格式化结构化数据。
- 警卫CLI参考— 本节介绍卫士中可用的命令CLI。

警卫功能

使用 Guard，您可以编写策略规则来验证任何JSON或YAML格式化的结构化数据，包括但不限于 AWS CloudFormation 模板。Guard 支持对策略检查的整个 end-to-end评估范围。规则在以下业务领域很有用：

- 预防性治理和合规性（左移测试）— 根据代表组织安全与合规最佳实践的策略规则，验证基础设施即代码 (IaC) 或基础设施和服务组合。例如，您可以验证 CloudFormation 模板、 CloudFormation 更改集、JSON基于 Terraform 的配置文件或 Kubernetes 配置。
- D@@ etective 治理和合规性 — 验证配置管理数据库 (CMDB) 资源的一致性，例如 AWS Config基于配置的项目 (CIs)。例如，开发人员可以使用 Guard 策略 AWS Config CIs来持续监控已部署资源 AWS 和非AWS 资源的状态，检测策略中的违规行为并开始修复。
- 部署安全-在部署之前确保更改是安全的。例如，根据策略规则验证 CloudFormation 变更集，以防止导致资源替换的更改，例如重命名 Amazon DynamoDB 表。

使用带 CloudFormation 钩子的防护

你可以使用 CloudFormation Guard 来创作 Hook in CloudFormation Hooks。 CloudFormation Hooks 允许您在 CloudFormation 创建、更新或删除操作以及 AWS 云端控制 API 创建或更新操作之前主动强制执行 Guard 规则。Hooks 可确保您的资源配置符合组织的安全、运营和成本优化最佳实践。

有关如何使用 Guard 创作 Gu CloudFormation ard Hook 的详细信息，请参阅 H ooks 用户指南中的编写防护规则来评估防护AWS CloudFormation 挂钩的资源。

访问警卫

要访问警卫DSL和命令，必须安装警卫CLI。有关安装 Guard 的信息CLI，请参阅设置警卫。

最佳实践

编写简单的规则，并使用命名规则在其他规则中引用它们。复杂的规则可能难以维护和测试。

设置 AWS CloudFormation Guard

AWS CloudFormation Guard 是一个开源命令行界面 (CLI)。它为您提供了一种简单的、特定于域的语言，用于编写策略规则并根据这些规则验证其结构化分层结构JSON和YAML数据。这些规则可以代表与安全、合规性等相关的公司政策指导方针。结构化分层数据可以表示描述为代码的云基础架构。例如，您可以创建规则来确保他们始终在其模板中对加密的亚马逊简单存储服务 (Amazon S3) 存储桶进行建模。 CloudFormation

以下主题提供有关如何使用所选操作系统或作为 AWS Lambda 功能安装 Guard 的信息。

主题

- 安装适用于 Linux 和 macOS 的 Guard
- 安装适用于 Windows 的防护
- 将 Guard 作为 AWS Lambda 功能安装

安装适用于 Linux 和 macOS 的 Guard

你可以使用预先构建 AWS CloudFormation Guard 的发行版二进制文件 Cargo 或 Homebrew 在 Linux 和 macOS 上安装。

从预先构建的发行版二进制文件安装 Guard

使用以下步骤从预先构建的二进制文件安装 Guard。

1. 打开终端，然后运行以下命令。

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

2. 运行以下命令来设置您的PATH变量。

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

结果：您已成功安装了 Guard 并设置了PATH变量。

- （可选）要确认 Guard 的安装，请运行以下命令。

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

该命令将返回以下输出。

```
cfn-guard 3.0.0
```

安装货物防护

Cargo 是 Rust 包管理器。完成以下步骤来安装 Rust，其中包括 Cargo。然后，安装 Guard from Cargo。

1. 从终端运行以下命令，然后按照屏幕上的说明安装 Rust。

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

   - （可选）对于 Ubuntu 环境，请运行以下命令。

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

2. 配置您的PATH环境变量，然后运行以下命令。

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

3. 安装了 Cargo 后，运行以下命令来安装 Guard。

   ```
   cargo install cfn-guard
   ```

   结果：你已经成功安装了 Guard。

   - （可选）要确认 Guard 的安装，请运行以下命令。

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

     该命令将返回以下输出。

     ```
     cfn-guard 3.0.0
     ```

从 Homebrew 安装 Guard

Homebrew 是一款适用于 macOS 和 Linux 的软件包管理器。完成以下步骤来安装 Homebrew。然后，从 Homebrew 中安装 Guard。

1. 从终端运行以下命令，然后按照屏幕上的说明安装 Homebrew。

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

2. 安装了 Homebrew 后，运行以下命令来安装 Guard。

```
brew install cloudformation-guard
```

结果：你已经成功安装了 Guard。

- （可选）要确认 Guard 的安装，请运行以下命令。

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

该命令将返回以下输出。

```
cfn-guard 3.0.0
```

安装适用于 Windows 的防护

你可以通过 Cargo 或 Ch AWS CloudFormation Guard ocolatey 在 Windows 上安装。

先决条件

要通过命令行界面构建 Guard，必须安装适用于 Visual Studio 2019 的构建工具。

1. 从 Visual Studio 2019 版构建工具网站上下载微软 Visual C++ 构建工具。
2. 运行安装程序，然后选择默认值。

安装货物防护

Cargo 是 Rust 包管理器。完成以下步骤来安装 Rust，其中包括 Cargo。然后，安装 Guard from Cargo。

1. 下载 Rust 然后运行 rustup-init.exe。
2. 在命令提示符下，选择 1，这是默认选项。

   该命令将返回以下输出。

   ```
   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. 要完成安装，请按 E nter 键。
4. 安装了 Cargo 后，运行以下命令来安装 Guard。

   ```
   cargo install cfn-guard
   ```

   结果：你已经成功安装了 Guard。

   - （可选）要确认 Guard 的安装，请运行以下命令。

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

     该命令将返回以下输出。

     ```
     cfn-guard 3.0.0
     ```

安装来自 Chocolatey 的守卫

Chocolatey 是一款适用于 Windows 的软件包管理器。完成以下步骤安装 Chocolatey。然后，安装来自 Chocolatey 的 Guard。

1. 按照本指南安装 Chocolat ey
2. 安装了 Chocolatey 后，运行以下命令来安装 Guard。

```
choco install cloudformation-guard
```

结果：你已经成功安装了 Guard。

- （可选）要确认 Guard 的安装，请运行以下命令。

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

该命令将返回以下输出。

```
cfn-guard 3.0.0
```

将 Guard 作为 AWS Lambda 功能安装

你可以 AWS CloudFormation Guard 通过 Rust 软件包管理器 Cargo 进行安装。Guard as a function (`cfn-guard-lambda`) 是围绕 Guard (`cfn-guard`) 的轻量级封装器，可用作 Lambda AWS Lambda 函数。

先决条件

在将 Guard 作为 Lambda 函数安装之前，必须满足以下先决条件：

- AWS Command Line Interface (AWS CLI) 配置了部署和调用 Lambda 函数的权限。有关更多信息，请参阅配置 AWS CLI。
- AWS Identity and Access Management (IAM) 中的 AWS Lambda 执行角色。有关更多信息，请参阅AWS Lambda 执行角色。
- 在 CentOS/ RHEL 环境中，将`musl-libc`软件包存储库添加到你的 yum 配置中。欲了解更多信息，请参阅 ngompa /musl-libc。

安装 Rust 软件包管理器

Cargo 是 Rust 包管理器。完成以下步骤来安装 Rust，其中包括 Cargo。

1. 从终端运行以下命令，然后按照屏幕上的说明安装 Rust。

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

- （可选）对于 Ubuntu 环境，请运行以下命令。

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

2. 配置您的PATH环境变量，然后运行以下命令。

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

将 Guard 安装为 Lambda 函数（Linux、macOS 或 Unix）

要将 Guard 安装为 Lambda 函数，请完成以下步骤。

1. 在您的命令终端上运行以下命令。

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

   - （可选）要确认将 Guard 安装为 Lambda 函数，请运行以下命令。

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

     该命令将返回以下输出。

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

2. 要安装musl支持，请运行以下命令。

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

3. 使用构建musl，然后在终端中运行以下命令。

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

   对于自定义运行时， AWS Lambda 需要一个名称为部署包.zip 文件bootstrap中的可执行文件。将生成的cfn-lambda可执行文件重命名为，bootstrap然后将其添加到.zip 存档中。

- 对于 macOS 环境，请在 Rust 项目的根目录或中创建您的货物配置文件。~/.cargo/config

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

4. 切换到cfn-guard-lambda根目录。

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

5. 在终端中运行以下命令。

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

6. 运行以下命令以cfn-guard作为 Lambda 函数提交到您的账户。

```
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
```

构建 Guard 并将其作为 Lambda 函数运行

要调用作cfn-guard-lambda为 Lambda 函数提交的，请运行以下命令。

```
aws lambda invoke --function-name cfnGuard \
  --payload '{"data":"input data","rules":["rule1","rule2"]}' \
  output.json
```

调用 Lambda 函数请求结构

cfn-guard-lambda要求填写以下字段：

- data— YAML 或JSON模板的字符串版本
- rules— 规则集文件的字符串版本

使用 Guard 规则的先决条件和概述

本节演示如何完成 Guard 核心任务，即针对 JSON 或 YAML 格式的数据编写、测试和验证规则。此外，它还包含详细的演练，演示了响应特定用例的编写规则。

主题

- 先决条件
- 使用防护规则的概述
- 写作 AWS CloudFormation Guard 规则
- 测试 AWS CloudFormation Guard 规则
- 将输入参数与 AWS CloudFormation Guard 规则配合使用
- 根据 AWS CloudFormation Guard 规则验证输入数据

先决条件

必须先安装 Guard 命令行界面 (CLI)，然后才能使用 Guard 域特定语言 (DSL) 编写策略规则。有关更多信息，请参阅 设置警卫。

使用防护规则的概述

使用 Guard 时，通常要执行以下步骤：

1. 写入 JSON 或 YAML 格式的数据进行验证。
2. 编写警卫策略规则。有关更多信息，请参阅 编写警卫规则。
3. 使用 Guard `test` 命令验证您的规则是否按预期运行。有关单元测试的更多信息，请参阅测试警卫规则。
4. 使用 Guard `validate` 命令根据您的规则验证 JSON 或 YAML 格式的数据。有关更多信息，请参阅 根据防护规则验证输入数据。

写作 AWS CloudFormation Guard 规则

在中 AWS CloudFormation Guard，规则就是 policy-as-code规则。您使用 Guard 域特定语言 (DSL) 编写规则，您可以根据这些语言来验证您的 YAML-或 JSON-格式的数据。规则由条款组成。

您可以将使用 Guard 编写的规则保存DSL到使用任何文件扩展名的纯文本文件中。

您可以创建多个规则文件并将其归类为一个规则集。规则集允许您同时根据多个规则文件验证 JSON YAML-或-格式的数据。

主题

- 子句
- 在子句中使用查询
- 在子句中使用运算符
- 在子句中使用自定义消息
- 合并条款
- 使用带有守卫规则的方块
- 定义 Guard 查询和过滤
- 在 Guard 规则中分配和引用变量
- 在中撰写命名规则块 AWS CloudFormation Guard
- 编写用于执行情境感知评估的子句

子句

子句是计算结果为 true (PASS) 或 false (FAIL) 的布尔表达式。子句要么使用二元运算符来比较两个值，要么使用对单个值进行运算的一元运算符。

一元子句的示例

以下一元子句评估集合`TcpBlockedPorts`是否为空。

```
InputParameters.TcpBlockedPorts not empty
```

以下一元子句评估该`ExecutionRoleArn`属性是否为字符串。

```
Properties.ExecutionRoleArn is_string
```

二进制子句的示例

以下二进制子句评估该`BucketName`属性是否包含字符串`encrypted`，无论大小写如何。

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

以下二进制子句评估该`ReadCapacityUnits`属性是否小于或等于 5,000。

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

编写 Guard 规则子句的语法

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

守卫规则子句的属性

`query`

写入以点 (`.`) 分隔的表达式，用于遍历分层数据。查询表达式可以包括筛选表达式来定位值的子集。可以将查询分配给变量，这样您就可以编写一次查询，然后在规则集的其他地方引用它们，这样您就可以访问查询结果。

有关编写查询和筛选的更多信息，请参阅定义查询和筛选。

必需：是

`operator`

一元运算符或二进制运算符，可帮助检查查询的状态。二元运算符的左边 (LHS) 必须是查询，右边 (RHS) 必须是查询或值文字。

支持的二元运算符：`==` `!=`（等于）`|>`（不等于）`|>=`（大于）`|`（大于或等于）`|<=`（小于）`|`（小于或等于）`|IN`（在 [x、y、z 格式列表中] <

支持的一元运算符：`exists`| `empty` | `is_string` | | `is_list` | `is_struct` `not(!)`

必需：是

`query|value literal`

查询或支持的值文字，例如`string`或`integer(64)`。

支持的值文字：

- 所有原始类型：`string`、、`integer(64)`、`float(64)`、`bool`、`char` `regex`
- 所有用于表达`integer(64)``float(64)`、或`char`范围的专用范围类型，表示为：
  - `r[<lower_limit>, <upper_limit>]`，它转换为满足以下表达式`k`的任何值：`lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`，它转换为满足以下表达式`k`的任何值：`lower_limit <= k < upper_limit`

  - r(<lower_limit>, <upper_limit>]，它转换为满足以下表达式k的任何值：lower_limit < k <= upper_limit
  - r(<lower_limit>, <upper_limit>),这表示任何满足以下表达式k的值：lower_limit < k < upper_limit
- 嵌套键值结构数据的关联数组（映射）。例如：

  { "my-map": { "nested-maps": [{ "key": 10, "value": 20 }] } }
- 原始类型或关联数组类型的数组

必需：有条件的；使用二元运算符时为必填项。

custom message

提供有关子句的信息的字符串。该消息显示在validate和test命令的详细输出中，可用于理解或调试分层数据的规则评估。

必需：否

在子句中使用查询

有关编写查询的信息，请参见定义查询和筛选和在 Guard 规则中分配和引用变量。

在子句中使用运算符

以下是示例 CloudFormation 模板，Template-1和Template-2。为了演示支持的运算符的使用，本节中的示例查询和子句参考了这些示例模板。

模板-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"
```

模板-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
```

使用一元运算符的子句示例

- empty— 检查集合是否为空。您还可以使用它来检查查询是否在分层数据中包含值，因为查询会生成集合。你不能用它来检查字符串值查询是否定义了空字符串 ("")。有关更多信息，请参阅 定义查询和筛选。

  以下子句检查模板是否定义了一个或多个资源。它的计算结果PASS为，因为中定义了具有逻辑 ID S3Bucket 的Template-1资源。

  ```
  Resources !empty
  ```

  以下子句检查是否为S3Bucket资源定义了一个或多个标签。它的计算结果PASS为，因为在中Template-1为该Tags属性定义S3Bucket了两个标签。

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

- exists— 检查每次出现的查询是否都有值并且可以用来代替!= null。

以下子句检查是否为定义了该BucketEncryption属性S3Bucket。它的计算结果PASS为，因为BucketEncryption是在S3Bucket中定义的。Template-1

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

> **Note**
>
> 遍历输入数据时，empty和not exists检查true的计算结果是否缺少属性键。例如，如果未在的模板中定义该Properties部分S3Bucket，则该子句的Resources.S3Bucket.Properties.Tag empty计算结果为。trueexists和empty勾选不会在错误消息中显示文档内的JSON指针路径。这两个子句通常都有检索错误，无法保留这些遍历信息。

- is_string— 检查每次出现的查询是否为string类型。

  以下子句检查是否为S3Bucket资源的BucketName属性指定了字符串值。它的计算结果PASS为，因为在BucketName中Template-1指定了字符串值"MyServiceS3Bucket"。

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

- is_list— 检查每次出现的查询是否为list类型。

  以下子句检查是否为S3Bucket资源的Tags属性指定了列表。它的计算结果PASS为，因为在中指定了两个键值对。Tags Template-1

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

- is_struct— 检查每次出现的查询是否都是结构化数据。

  以下子句检查是否为S3Bucket资源的BucketEncryption属性指定了结构化数据。它的计算结果PASS为，因为BucketEncryption是使用*(object)*中的ServerSideEncryptionConfigurationTemplate-1属性类型指定的。

> **Note**
>
> 要检查反向状态，可以将 (not !) 运算符与is_stringis_list、和运is_struct算符一起使用。

使用二元运算符的子句示例

以下子句检查为中的S3Bucket资源BucketName属性指定的值是否Template-1包含字符串encrypt，无论大小写如何。其计算结果PASS为，因为指定的存储桶名称"MyServiceS3Bucket"不包含字符串encrypt。

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

以下子句检查为中的NewVolume资源Size属性指定的值Template-2是否在特定范围内：50 <= Size <= 200。它的计算结果PASS为，因为100是指定的。Size

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

以下子句检查中为NewVolume资源VolumeType属性指定的值是否Template-2为io1io2、或gp3。它的计算结果PASS为，因为io1是指定的。NewVolume

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

> **Note**
>
> 本节中的示例查询演示了如何使用带有逻辑IDsS3Bucket和的资源来使用运算符NewVolume。资源名称通常是用户定义的，可以在基础设施即代码 (IaC) 模板中任意命名。要编写适用于模板中定义的所有AWS::S3::Bucket资源的通用规则，最常用的查询形式是Resources.*[Type == 'AWS::S3::Bucket']。有关更多信息，请参阅定义查询和筛选，了解有关用法的详细信息，并浏览cloudformation-guard GitHub 存储库中的示例目录。

在子句中使用自定义消息

在以下示例中，的子句Template-2包括一条自定义消息。

```
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>>
```

合并条款

在 Guard 中，写在新行上的每个子句都使用连词（布尔and逻辑）与下一个子句隐式组合。请参阅以下示例。

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

也可以使用 disjunction 将子句与下一个子句合并，方法是在第一个子句的or|OR末尾指定。

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

在 Guard 子句中，首先评估分离词，然后评估连词。保护规则可以定义为计算结果为 () 或 false () and|AND 的子句（一个 of or|OR sPASS）的组合。true FAIL这类似于连词法线形式。

以下示例演示了子句的求值顺序。

```
# (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
```

所有基于该示例的子句均Template-1可使用连词进行组合。请参阅以下示例。

```
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
```

使用带有守卫规则的方块

块是从一组相关的子句、条件或规则中去除冗长和重复内容的组合。有三种类型的方块：

- 查询区块
- when方块
- 命名规则块

查询区块

以下是基于该示例的子句Template-1。连词用于合并子句。

```
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
```

每个子句中的部分查询表达式都是重复的。通过使用查询块，您可以提高组合性，并消除一组具有相同初始查询路径的相关子句的冗长性和重复性。可以编写相同的子句集，如以下示例所示。

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

```
    Tags is_list
    Tags !empty
}
```

在查询块中，块之前的查询为块内的子句设置上下文。

有关使用方块的更多信息，请参阅撰写命名规则块。

when方块

你可以使用方块有条件地评估when方块，其形式如下。

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

when关键字表示when区块的起点。 condition是守卫规则。仅当对条件的评估结果为 true (PASS) 时，才会对方块进行评估。

以下是基于的示例when块Template-1。

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

仅当为指定的值为字符串时，才会评估when块中的子句。BucketName如果在模板的Parameters部分中引用了BucketName为指定的值，如以下示例所示，则不会评估该when块中的子句。

```
Parameters:
   S3BucketName:
     Type: String

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

命名规则块

您可以为一组规则（规则集）指定名称，然后在其他规则中引用这些模块化验证块，称为命名规则块。命名规则块采用以下形式。

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

rule关键字表示命名规则块的开头。

rule name是一个人类可读的字符串，用于唯一标识命名规则块。这是它封装的 Guard 规则集的标签。在这种用法中，“保护规则” 一词包括子句、查询块、when块和命名规则块。规则名称可用于指代其封装的规则集的评估结果，这使得命名规则块可重复使用。规则名称还提供了有关validate和test命令输出中规则失败的上下文。规则名称与区块的评估状态（PASSFAIL、或SKIP）一起显示在规则文件的评估输出中。请参阅以下示例。

```
# 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**
```

您还可以通过在规则名称后指定when关键字和条件来有条件地评估命名规则块。

以下是本主题前面讨论的示例when块。

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

使用命名规则块，也可以将前面的内容写成以下内容。

```
rule checkBucketNameIsString {
    Resources.S3Bucket.Properties.BucketName is_string
```

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

您可以重复使用命名规则块并将其与其他 Guard 规则分组。以下是几个例子。

```
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
    ...
}
```

定义 Guard 查询和过滤

本主题介绍编写查询以及在编写 Guard 规则子句时使用筛选。

先决条件

过滤是一个高级 AWS CloudFormation Guard 概念。我们建议您在学习筛选之前先阅读以下基础主题：

- 什么是 AWS CloudFormation Guard？
- 写作规则、条款

定义查询

查询表达式是为遍历分层数据而编写的简单点 (.) 分隔表达式。查询表达式可以包括筛选表达式来定位值的子集。对查询进行评估时，它们会生成一组值，类似于 SQL 查询返回的结果集。

以下示例查询在 AWS CloudFormation 模板中搜索`AWS::IAM::Role`资源。

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

查询遵循以下基本原则：

- 当使用显式关键词时，查询的每个点 (.) 部分都会向下遍历层次结构，例如`Resources`或`Properties.Encrypted`.如果查询的任何部分与传入的数据不匹配，Guard 将引发检索错误。
- 查询中使用通配符的 dot (.) 部分`*`会遍历该级别结构的所有值。
- 查询中使用数组通配符的 dot (.) 部分会`[*]`遍历该数组的所有索引。
- 可以通过在方括号内指定过滤器来筛选所有集合`[]`。可以通过以下方式遇到集合：
  - datum 中自然出现的数组是集合。下面是一些 示例：

    端口：`[20, 21, 110, 190]`

    标签：`[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - 遍历结构的所有值时 `Resources.*`
  - 任何查询结果本身都是一个集合，可以从中进一步筛选值。请参阅以下示例。

    ```
    # 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
}
```

以下是示例 CloudFormation 模板片段。

```
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
    ...
```

基于此模板，遍历的路径为`SampleRole`，选择的最终值为。`Type: AWS::IAM::Role`

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

YAML 格式`Resources.*[ Type == 'AWS::IAM::Role' ]`的查询结果值如下例所示。

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

您可以使用查询的一些方法如下：

- 为变量分配查询，以便可以通过引用这些变量来访问查询结果。

- 在查询之后使用一个针对每个选定值进行测试的块。
- 直接将查询与基本子句进行比较。

为变量分配查询

Guard 支持在给定范围内的一次性变量赋值。有关 Guard 规则中变量的更多信息，请参阅在 Guard 规则中分配和引用变量。

您可以将查询分配给变量，这样您就可以编写一次查询，然后在 Guard 规则的其他地方引用它们。参见以下变量赋值示例，这些变量赋值演示了本节后面讨论的查询原理。

```
#
# 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'
    }
]
```

直接遍历分配给查询的变量中的值

Guard 支持直接根据查询结果运行。在以下示例中，when模块针对 CloudFormation 模板中找到的每个AWS::EC2::Volume资源的EncryptedVolumeType、和AvailabilityZone属性进行测试。

```
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']
```

```
        }
    }
}
```

直接进行子句级别的比较

Guard 还支持将查询作为直接比较的一部分。例如，请参阅以下内容。

```
let resources = Resources.*

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

在前面的示例中，以所示形式表示的两个子句（以some关键字开头）被视为独立子句，并且是分开计算的。

单一条款和区块条款形式

总而言之，前一节中显示的两个示例子句并不等同于以下块。

```
let resources = Resources.*

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

此块查询集合中的每个Tag值，并将其属性值与预期的属性值进行比较。前一节中子句的组合形式对这两个子句进行独立评估。考虑以下输入。

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

第一种形式的子句的计算结果为PASS。验证第一种形式的第一个子句时，、ResourcesPropertiesTags、和的以下路径与值Key相匹配但NotPRODEnd与预期值PROD不匹配。

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

第一种形式的第二个子句也是如此。Resources、PropertiesTags、和的路径与值Value相匹配AppStart。结果，第二个子句是独立的。

总体结果是PASS.

但是，方块形态的计算结果如下。对于每个Tags值，它会比较Key和Value是否匹配；NotAppStart以下示例中的NotPRODEnd值不匹配。

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

由于评估会同时检查和 `Key == /PROD$/Value == /^App/`，因此匹配未完成。因此，结果是FAIL。

> **Note**
>
> 使用集合时，我们建议您在要比较集合中每个元素的多个值时使用块子句形式。如果集合是一组标量值，或者只想比较单个属性，则使用单子句形式。

查询结果和相关条款

所有查询都会返回一个值列表。遍历的任何部分，例如缺少键、访问所有索引时数组 (Tags: []) 的空值，或者遇到空地图 (Resources: {}) 时地图的缺失值，都可能导致检索错误。

在根据此类查询评估子句时，所有检索错误都被视为失败。唯一的例外是在查询中使用显式过滤器时。使用筛选器时，会跳过关联的子句。

以下区块失败与正在运行的查询有关。

- 如果模板不包含资源，则查询的计算结果为FAIL，关联的块级子句也计算为FAIL。
- 当模板包含类似的空资源块时{ "Resources": {} }，查询的计算结果为FAIL，关联的块级子句也计算为FAIL。
- 如果模板包含资源但没有与查询相匹配的资源，则查询将返回空结果，并跳过区块级子句。

在查询中使用过滤器

查询中的过滤器实际上是用作选择标准的保护子句。以下是条款的结构。

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

使用筛选器写作 AWS CloudFormation Guard 规则时，请记住以下要点：

- 使用连词普通形式 (CNF) 合并子句。
- 在新行上指定每个连词 (and) 子句。
- 通过在两个子句之间使用or关键字来指定分离 (or)。

以下示例演示了连词和分离子句。

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

InputParameters.TcpBlockedPorts[*] {
    this in r(100, 400] or
    this in r(4000, 65535]
}
```

使用子句作为选择标准

您可以对任何集合应用筛选。筛选可以直接应用于输入中已经是类似集合的属性`securityGroups: [....]`。您也可以对查询应用筛选，查询始终是值的集合。您可以使用子句的所有功能（包括连词普通形式）进行筛选。

从 CloudFormation 模板中按类型选择资源时，通常使用以下常用查询。

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

该查询`Resources.*`返回输入`Resources`部分中存在的所有值。对于中的示例模板输入定义查询，查询返回以下内容。

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

现在，对这个集合应用过滤器。匹配的标准是`Type == AWS::IAM::Role`。以下是应用筛选器后的查询输出。

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

接下来，检查各种`AWS::IAM::Role`资源条款。

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

以下是选择所有`AWS::IAM::ManagedPolicy`资源`AWS::IAM::Policy`和资源的筛选查询示例。

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

以下示例检查这些策略资源是否已指PolicyDocument定。

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

构建更复杂的过滤需求

考虑以下入口和出口安全组信息的 AWS Config 配置项目示例。

```
---
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
```

请注意以下几点：

- ipPermissions（入口规则）是配置块内的规则集合。
- 每个规则结构都包含诸如ipv4Ranges和之类的属性ipv6Ranges，用于指定 CIDR 块的集合。

让我们编写一条规则，选择允许来自任何 IP 地址的连接的所有入口规则，并验证这些规则是否不允许泄露 TCP 阻塞的端口。

从涵盖的查询部分开始 IPv4，如以下示例所示。

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

在此上下文中，some关键字很有用。所有查询都会返回与查询匹配的值的集合。默认情况下，Guard 会评估查询结果返回的所有值都与校验相匹配。但是，这种行为可能并不总是你需要进行检查的。考虑配置项目输入的以下部分。

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

存在两个值ipv4Ranges。并非所有ipv4Ranges值都等于用表示的 0.0.0.0/0 IP 地址。你想看看是否至少有一个值匹配0.0.0.0/0。你告诉 Guard，并非所有从查询返回的结果都需要匹配，但至少

有一个结果必须匹配。some关键字告诉 Guard 确保结果查询中的一个或多个值与校验相匹配。如果没有匹配的查询结果值，Guard 将引发错误。

接下来 IPv6，添加，如以下示例所示。

```
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'
]
```

最后，在以下示例中，验证协议是否不是udp。

```
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' ]
]
```

以下是完整的规则。

```
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
                    >>
                }
            }

        }
     }
}
```

根据集合包含的类型分隔集合

使用基础设施即代码 (IaC) 配置模板时，您可能会遇到一个集合，其中包含对配置模板中其他实体的引用。以下是描述亚马逊弹性容器服务 (Amazon ECS) 任务的示例 CloudFormation 模板，其中包含本地引用`TaskRoleArn`、引用`TaskArn`和直接字符串引用。

```
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'
```

请考虑以下查询。

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

此查询返回的值集合包含示例模板中显示的所有三个AWS::ECS::TaskDefinition资源。将ecs_tasks包含TaskRoleArn本地引用的内容与其他引用分开，如以下示例所示。

```
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
    >>
}
```

在 Guard 规则中分配和引用变量

您可以在 AWS CloudFormation Guard 规则文件中分配变量，以存储要在 Guard 规则中引用的信息。Guard 支持一次性变量赋值。变量是延迟计算的，这意味着 Guard 仅在规则运行时才计算变量。

主题

- 分配变量
- 引用变量
- 变量作用域
- 守卫规则文件中的变量示例

分配变量

使用let关键字初始化变量并分配变量。最佳做法是使用蛇形大小写变量名。变量可以存储查询产生的静态文字或动态属性。在以下示例中，变量ecs_task_definition_task_role_arn存储静态字符串值arn:aws:iam:123456789012:role/my-role-name。

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

在以下示例中，变量ecs_tasks存储了在 AWS CloudFormation 模板中搜索所有AWS::ECS::TaskDefinition资源的查询结果。在编写规则时ecs_tasks，您可以参考有关这些资源的访问信息。

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

引用变量

使用前%缀来引用变量。

根据中的ecs_task_definition_task_role_arn变量示例分配变量，您可以在 Guard 规则子句的query|value literal部分ecs_task_definition_task_role_arn中进行引用。使用该引用可确保为 CloudFormation 模板中任何AWS::ECS::TaskDefinition资源的TaskDefinitionArn属性指定的值都是静态字符串值arn:aws:iam:123456789012:role/my-role-name。

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

根据中的ecs_tasks变量示例分配变量，您可以在查询中进行引用ecs_tasks（例如，%ecs_tasks.Properties）。首先，Guard 评估变量，ecs_tasks然后使用返回的值遍历层次结构。如果变量ecs_tasks解析为非字符串值，则 Guard 会引发错误。

> **Note**
>
> 目前，Guard 不支持在自定义错误消息中引用变量。

变量作用域

范围是指规则文件中定义的变量的可见性。一个变量名只能在一个作用域内使用一次。有三个级别可以声明变量，或者有三个可能的变量范围：

- 文件级 — 通常在规则文件顶部声明，可以在规则文件的所有规则中使用文件级变量。它们对整个文件都是可见的。

  在以下示例规则文件中，变量ecs_task_definition_task_role_arn和ecs_task_definition_execution_role_arn是在文件级别初始化的。

```
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
}
```

- 规则级别 — 在规则中声明，规则级变量仅对该特定规则可见。规则之外的任何引用都会导致错误。

  在以下示例规则文件中，变量ecs_task_definition_task_role_arn和ecs_task_definition_execution_role_arn是在规则级别初始化的。ecs_task_definition_task_role_arn只能在check_ecs_task_definition_task_role_arn命名规则中引用。您只能在check_ecs_task_definition_execution_role_arn命名规则中引用该ecs_task_definition_execution_role_arn变量。

```
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
}
```

- 块级 — 在块（例如when子句）中声明，块级变量仅对该特定块可见。区块之外的任何引用都会导致错误。

  在以下示例规则文件中，变量ecs_task_definition_task_role_arn和ecs_task_definition_execution_role_arn是在类型块内的块级别初始化的AWS::ECS::TaskDefinition。您只能在AWS::ECS::TaskDefinition类型块中为其各自的规则引用ecs_task_definition_task_role_arn和ecs_task_definition_execution_role_arn变量。

```
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
    }
}
```

守卫规则文件中的变量示例

以下各节提供了变量静态和动态赋值的示例。

静态赋值

以下是一个示例 CloudFormation 模板。

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

基于此模板，您可以编写一个名为的规则，该规则可`check_ecs_task_definition_task_role_arn`确保所有`AWS::ECS::TaskDefinition`模板资源的`TaskRoleArn`属性均为`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
}
```

在规则的范围内，您可以初始化名`ecs_task_definition_task_role_arn`为的变量，并为其分配静态字符串值`'arn:aws:iam::123456789012:role/my-role-name'`。规则子句`arn:aws:iam::123456789012:role/my-role-name`通过引用`query|value literal`部分中的`ecs_task_definition_task_role_arn`变量来检查为`EcsTask`资源`TaskRoleArn`属性指定的值是否为该值。

动态分配

以下是一个示例 CloudFormation 模板。

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

基于此模板，您可以初始化在文件ecs_tasks范围内调用的变量，并将查询分配给该变量Resources.*[Type == 'AWS::ECS::TaskDefinition'。Guard 会查询输入模板中的所有资源，并将有关这些资源的信息ecs_tasks。你也可以编写一个名为的规则，check_ecs_task_definition_task_role_arn确保所有AWS::ECS::TaskDefinition模板资源的TaskRoleArn属性都是 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'
}
```

规则子句arn:aws:iam::123456789012:role/my-role-name通过引用query部分中的ecs_task_definition_task_role_arn变量来检查为EcsTask资源TaskRoleArn属性指定的值是否为该值。

强制 AWS CloudFormation 模板配置

让我们来看一个更复杂的生产用例示例。在此示例中，我们编写了 Guard 规则，以确保更严格地控制 Amazon ECS 任务的定义方式。

以下是一个示例 CloudFormation 模板。

```
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'
```

基于此模板，我们编写了以下规则来确保满足这些要求：

- 模板中的每个AWS::ECS::TaskDefinition资源都附加了一个任务角色和一个执行角色。
- 任务角色和执行角色是 AWS Identity and Access Management (IAM) 角色。
- 角色在模板中定义。
- 该PermissionsBoundary属性是为每个角色指定的。

```
# 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
    }
}
```

在中撰写命名规则块 AWS CloudFormation Guard

使用编写命名规则块时 AWS CloudFormation Guard，可以使用以下两种构图风格：

- 条件依赖关系
- 相关依赖

使用这两种依赖关系组合方式中的任何一种都有助于提高可重用性，并减少命名规则块中的冗长和重复。

主题

- 先决条件
- 条件依赖组合
- 关联依赖关系构成

先决条件

在编写规则中了解命名规则块。

条件依赖组合

在这种组合方式中，对when块或命名规则块的评估有条件依赖于一个或多个其他命名规则块或子句的评估结果。以下示例 Guard 规则文件包含演示条件依赖关系的命名规则块。

```
# 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
    ...
}
```

在前面的示例规则文件中，可能Example-1有以下结果：

- 如果rule_name_A评估为PASS，则对封装的守卫规则进行评估。rule_name_B
- 如果rule_name_A评估为FAIL，则不评估封装的rule_name_B守卫规则。 rule_name_B评估为。SKIP
- 如果rule_name_A评估为SKIP，则不评估封装的rule_name_B守卫规则。 rule_name_B评估为。SKIP

> **Note**
>
> 如果rule_name_A有条件地依赖于计算结果为的规则，则会FAIL发生这种rule_name_A情况。SKIP

以下是来自入口和出口安全组信息项目的配置管理数据库 (CMDB) 配置 AWS Config 项目的示例。此示例演示了条件依赖项组合。

```
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
                }
            }
        }
    }
}
```

在前面的示例中，check_parameter_validity有条件地依赖于check_resource_type_and_parameter并check_ip_procotol_and_port_range_validity有条件地依赖于。check_parameter_validity以下是符合上述规则的配置管理数据库 (CMDB) 配置项目。

```
---
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
```

关联依赖关系构成

在这种组合方式中，对区块或命名规则when块的评估与一个或多个其他 Guard 规则的评估结果具有相关依赖性。相关依赖可以通过以下方式实现。

```
# 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
    ...
}
```

为了帮助您理解关联依赖关系构成，请查看以下警卫规则文件示例。

```
#
# 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'
}
```

在前面的规则文件中，ensure_elbs_are_internal_and_secure具有相关的依赖关系。ensure_all_elbs_are_secure以下是符合上述规则的示例 CloudFormation 模板。

```
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...'
```

编写用于执行情境感知评估的子句

AWS CloudFormation Guard 子句是根据分层数据进行评估的。Guard 评估引擎通过遵循指定的分层数据，使用简单的点分符号来解析针对传入数据的查询。通常，需要多个子句来根据数据或集合映射进行评估。Guard 提供了一种便捷的语法来编写此类子句。该引擎具有情境感知能力，并使用与之相关的相应数据进行评估。

以下是带有容器的 Kubernetes Pod 配置的示例，您可以对其应用上下文感知评估。

```
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
```

您可以编写 Guard 子句来评估这些数据。评估规则文件时，上下文是整个输入文档。以下是验证对 Pod 中指定的容器实施限制的示例子句。

```
#
# 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
            >>
        }
    }
}
```

评估context中的理解

在规则块级别，传入的上下文是完整的文档。对when条件的评估是针对apiVersion和kind属性所在的传入根上下文进行的。在前面的示例中，这些条件的计算结果为true。

现在，遍历前面示例所spec.containers[*]示的层次结构。对于层次结构的每次遍历，上下文值都会相应变化。spec区块遍历完成后，上下文会发生变化，如以下示例所示。

```
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
```

遍历containers属性后，上下文显示在以下示例中。

```
- 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
```

了解循环

您可以使用表达式[*]为containers属性数组中包含的所有值定义循环。对方块中的每个元素进行评估containers。在前面的示例规则片段中，块中包含的子句定义了要根据容器定义进行验证的检查。其中包含的子句块会被评估两次，每个容器定义一次。

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

对于每次迭代，上下文值是相应索引处的值。

> **Note**
>
> 唯一支持的索引访问格式是[<integer>]或[*]。目前，Guard 不支持这样的射程[2..4]。

数组

通常，在接受数组的地方，也接受单值。例如，如果只有一个容器，则可以删除数组并接受以下输入。

```
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
```

如果属性可以接受数组，请确保您的规则使用数组形式。在前面的示例中，您使用的是 and containers[*] not containers。当Guard仅遇到单值形式时，Guard在遍历数据时会正确进行评估。

> **Note**
>
> 当属性接受数组时，在表达对规则子句的访问权限时，请务必使用数组形式。即使使用单个值，Guard 也能正确评估。

使用表单spec.containers[*]代替 spec.containers

Guard 查询返回已解析值的集合。使用表单时spec.containers，查询的解析值包含引用的数组containers，而不是其中的元素。使用表单时spec.containers[*]，指的是其中包含的每个单独元素。每当你打算计算数组中包含的每个元素时，请记住使用[*]表单。

this用于引用当前上下文值

在创作 Guard 规则时，您可以使用引用上下文值this。通常，this是隐式的，因为它与上下文的值绑定。例如，this.apiVersionthis.kind、和绑this.spec定到根目录或文档。相比之下this.resources，绑定到的每个值containers，例如/spec/containers/0/和/spec/containers/1。同样，this.cpu并this.memory映射到极限，具体而言，/spec/containers/0/resources/limits和/spec/containers/1/resources/limits。

在下一个示例中，对前面的 Kubernetes Pod 配置规则进行了重写以明确使用。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
            >>
        }
    }
}
```

您无需this明确使用。但是，在使用标量时，该this引用可能很有用，如以下示例所示。

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

在前面的示例中，this用于指代每个端口号。

使用隐式可能出现的错误 **this**

在编写规则和子句时，从隐式this上下文值中引用元素时会出现一些常见错误。例如，考虑以下要计算的输入数据（必须通过）。

```
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
```

当针对前面的模板进行测试时，以下规则会导致错误，因为它错误地假设利用了隐式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
                >>
            }
        }
    }
}
```

要完成此示例，请使用名称保存前面的规则文件any_ip_ingress_check.guard，用文件名保存数据ip_ingress.yaml。然后，使用这些文件运行以下validate命令。

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

在以下输出中，引擎表示尝试检索该值InputParameters.TcpBlockedPorts[*]/configuration/ipPermissions/0的属性/configuration/ipPermissions/1失败。

```
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[*]
```

为了帮助理解此结果，请使用this显式引用重写规则。

```
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.InputParameters引用变量中包含的每个值any_ip_permissions。分配给变量的查询会选择匹配的configuration.ipPermissions值。该错误表示有人尝试在此上下文InputParamaters中进行检索，但InputParameters是在根上下文中。

内部块还引用超出作用域的变量，如以下示例所示。

```
{
    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指中的每个端口值[21, 22, 110]，但它也指fromPort和toPort。它们都属于外部块作用域。

通过隐式使用来解决错误 **this**

使用变量来显式分配和引用值。首先，InputParameter.TcpBlockedPorts是输入（根）上下文的一部分。移InputParameter.TcpBlockedPorts出内部方块并对其进行显式分配，如以下示例所示。

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

然后，明确引用此变量。

```
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
                >>
            }
        }
    }
}
```

对内部this引用执行同样的操作%ports。

但是，所有错误尚未修复，因为里面的循环ports仍然有错误的引用。以下示例显示了如何删除错误的引用。

```
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
                    >>
                }
            }
        }
    }
}
```

接下来，再次运行该validate命令。这一次，它过去了。

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

以下是该validate命令的输出。

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

为了测试这种方法是否存在故障，以下示例使用了有效载荷更改。

```
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 在允许任何IPv6地址的 89—109 范围内。以下是再次运行该validate命令后的输出。

```
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
```

测试 AWS CloudFormation Guard 规则

您可以使用 AWS CloudFormation Guard 内置的单元测试框架来验证您的 Guard 规则是否按预期运行。本节提供了有关如何编写单元测试文件以及如何使用该文件通过test命令测试规则文件的演练。

您的单元测试文件必须具有以下扩展名之一：.json、.JSON.jsn、.yaml、.YAML、或.yml。

主题

- 先决条件
- Guard 单元测试文件概述
- 编写 Guard 规则单元测试文件的演练

先决条件

编写 Guard 规则来评估您的输入数据。有关更多信息，请参阅 编写警卫规则。

Guard 单元测试文件概述

Guard 单元测试文件是 JSON-或 YAML-格式的文件，其中包含多个输入以及写在 Guard 规则文件中的规则的预期结果。可以有多个样本来评估不同的期望。我们建议您首先测试是否存在空白输入，然后逐步添加用于评估各种规则和条款的信息。

此外，我们建议您使用后缀_test.json或_tests.yaml来命名单元测试文件。例如，如果您有一个名为的规则文件my_rules.guard，请命名您的单元测试文件my_rules_tests.yaml。

语法

下面显示了YAML格式化的单元测试文件的语法。

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

属性

以下是 Guard 测试文件的属性。

`input`

用于测试您的规则的数据。我们建议您的第一次测试使用空输入，如以下示例所示。

```
---
- name: MyTest1
```

```
  input {}
```

对于后续测试，请将输入数据添加到测试中。

必需：是

`expectations`

根据您的输入数据评估特定规则时的预期结果。除了每条规则的预期结果外，还要指定一个或多个要测试的规则。预期结果必须是以下之一：

- PASS— 当针对您的输入数据运行时，规则的计算结果为`true`。
- FAIL— 当针对您的输入数据运行时，规则的计算结果为`false`。
- SKIP— 当针对您的输入数据运行时，该规则不会被触发。

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

必需：是

编写 Guard 规则单元测试文件的演练

以下是名为的规则文件`api_gateway_private.guard`。此规则的目的是检查 CloudFormation 模板中定义的所有 Amazon API Gateway 资源类型是否仅用于私有访问。它还会检查是否至少有一条策略声明允许从虚拟私有云进行访问 (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
            }
        }
    }
}
```

本演练测试了第一个规则意图：部署的所有AWS::ApiGateway::RestApi资源都必须是私有的。

1. 创建一个名为的单元测试文件api_gateway_private_tests.yaml，其中包含以下初始测试。在初始测试中，添加一个空输入，预计该规则check_rest_api_is_private会跳过，因为没有AWS::ApiGateway::RestApi资源作为输入。

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

```
      check_rest_api_is_private: SKIP
```

2. 使用test命令在终端中运行第一个测试。对于--rules-file参数，请指定您的规则文件。对于--test-data参数，请指定您的单元测试文件。

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

第一次测试的结果是PASS。

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

3. 向您的单元测试文件中添加另一个测试。现在，将测试范围扩展到包括空资源。以下是更新的api_gateway_private_tests.yaml文件。

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

4. test使用更新的单元测试文件运行。

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

第二次测试的结果是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. 在单元测试文件中再添加两个测试。将测试范围扩展到包括以下内容：

   - 未指定属性的AWS::ApiGateway::RestApi资源。

     > **Note**
     >
     > 这不是一个有效的 CloudFormation 模板，但是即使对于格式错误的输入，测试该规则是否正常工作也很有用。

     预计此测试将失败，因为未指定该EndpointConfiguration属性，因此未将其设置为PRIVATE。

   - 一种AWS::ApiGateway::RestApi资源，它满足第一个意图，其EndpointConfiguration属性设置为，PRIVATE但由于未定义策略语句而未满足第二个意图。预计此测试将通过。

   以下是更新的单元测试文件。

```
---
- 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. test使用更新的单元测试文件运行。

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

第三个结果是FAIL，第四个结果是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. 在单元测试文件中注释掉测试 1—3。仅访问第四次测试的详细上下文。以下是更新的单元测试文件。

```
---
#- 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. 使用--verbose标志，在终端中运行test命令来检查评估结果。详细的上下文对于理解评估很有用。在这种情况下，它提供了有关为什么第四次测试成功并得出PASS结果的详细信息。

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

```
  --verbose
```

这是那次运行的输出。

```
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)
```

从输出中观察到的关键是线`Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`，它表示检查已通过。该示例还显示了本应为数组，但给出了单个值的情况`Types`。在这种情况下，Guard继续进行评估并提供了正确的结果。

9. 将像第四个测试用例这样的测试用例添加到具有指定`EndpointConfiguration`属性的`AWS::ApiGateway::RestApi`资源的单元测试文件中。测试用例将失败而不是通过。以下是更新的单元测试文件。

```
---
```

```
#- 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. 使用--verbose标志，使用更新的单元测试文件运行test命令。

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

结果符合预期FAIL，因为REGIONAL已指定，EndpointConfiguration但不是预料之中的。

```
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)
```

该test命令的详细输出遵循规则文件的结构。规则文件中的每个块都是详细输出中的一个块。最上面的方块是每条规则。如果存在违反该规则的when条件，则它们会出现在同级条件块中。在以下示例中，对条件%api_gws !empty进行了测试并通过了。

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

条件通过后，我们将测试规则子句。

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

%api_gws是与输出中的BlockClause级别相对应的分组规则（第 21 行）。规则子句是一组连词 (AND) 子句，其中每个连词子句都是一组分离词。OR连词只有一个子句，Properties.EndpointConfiguration.Types[*] == "PRIVATE"。因此，详细输出显示了一个子句。路径/Resources/apiGw/Properties/EndpointConfiguration/Types/1显示比较输入中的哪些值，在本例中为Types索引为 1 的元素。

在中根据防护规则验证输入数据，您可以使用本节中的示例使用validate命令根据规则评估输入数据。

将输入参数与 AWS CloudFormation Guard 规则配合使用

AWS CloudFormation Guard 允许您在验证期间使用输入参数进行动态数据查找。当您需要在规则中引用外部数据时，此功能特别有用。但是，在指定输入参数键时，Guard 要求路径没有冲突。

如何使用

1. 使用--input-parameters或-i标志指定包含输入参数的文件。可以指定多个输入参数文件，这些文件将组合起来形成一个共同的上下文。输入参数键不能有冲突的路径。
2. 使用--data或-d标志指定要验证的实际模板文件。

示例用法

1. 创建输入参数文件（例如，network.yaml）：

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

2. 在您的防护规则文件中引用以下参数（例如，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. 创建失败的数据模板（例如，security_groups_fail.yaml）：

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

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

4. 运行验证命令：

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

在此命令中：

- `-r`指定规则文件
- `-i`指定输入参数文件
- `-d`指定要验证的数据文件（模板）

多个输入参数

您可以指定多个输入参数文件：

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

用指定的所有文件都`-i`将合并为一个用于参数查找的单一上下文。

根据 AWS CloudFormation Guard 规则验证输入数据

您可以使用 AWS CloudFormation Guard `validate`命令根据防护规则验证数据。有关该`validate`命令的更多信息，包括其参数和选项，请参阅验证。

先决条件

- 编写 Guard 规则来验证您的输入数据。有关更多信息，请参阅 编写警卫规则。
- 测试您的规则，确保它们按预期运行。有关更多信息，请参阅 测试警卫规则。

使用 `validate` 命令

要根据 Guard 规则（例如 AWS CloudFormation 模板）验证您的输入数据，请运行 Guard `validate`命令。为`--rules`参数指定规则文件的名称。为`--data`参数指定输入数据文件的名称。

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

如果 Guard 成功验证了模板，则该validate命令将返回退出状态0（$?在 bash 中）。如果 Guard 发现了违反规则的情况，则该validate命令将返回失败规则的状态报告。使用摘要标志 (-s all) 查看详细的评估树，该树显示 Guard 是如何评估每条规则的。

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

针对多个数据文件验证多条规则

为了帮助维护规则，您可以将规则写入多个文件并根据需要组织规则。然后，您可以根据一个或多个数据文件验证多个规则文件。该validate命令可以获取--data和--rules选项的文件目录。例如，您可以运行以下命令，其中/path/to/dataDirectory包含一个或多个数据文件并/path/to/ruleDirectory包含一个或多个规则文件。

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

您可以编写规则来检查在多个 CloudFormation模板中定义的各种资源是否具有适当的属性分配，以保证静态加密。为了便于搜索和维护，您可以制定规则，用于检查每个资源的静态加密情况，这些文件名为s3_bucket_encryption.guardec2_volume_encryption.guard、，以及rds_dbinstance_encrytion.guard带有路径的目录~/GuardRules/encryption_at_rest。您需要验证的 CloudFormation模板位于路径为的目录中~/CloudFormation/templates。在这种情况下，按如下方式运行validate命令。

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

故障排除 AWS CloudFormation Guard

如果您在使用时遇到问题 AWS CloudFormation Guard，请查阅本节的主题。

主题

- 当选定类型的资源不存在时，子句失败
- Guard 不评估带有简短引用的 CloudFormation 模板 Fn::GetAtt
- 一般疑难解答主题

当选定类型的资源不存在时，子句失败

当查询使用过滤器时`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`，如果输入中没有`AWS::ApiGateway::RestApi`资源，则子句的计算结果为。`FAIL`

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

为避免出现这种结果，请为变量分配过滤器并使用`when`条件检查。

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

Guard 不评估带有简短引用的 CloudFormation 模板 Fn::GetAtt

Guard 不支持内部函数的简短形式。例如，不支持`!Sub`在YAML格式化的 AWS CloudFormation 模板中使用`!Join`。相反，请使用 CloudFormation 内部函数的扩展形式。例如`Fn::Join`，`Fn::Sub`在对照 Guard 规则评估 CloudFormation 模板时，在YAML格式化的模板中使用。

有关内部函数的更多信息，请参阅《用户指南》中的内部函数参考。AWS CloudFormation

一般疑难解答主题

- 确认`string`文字不包含嵌入的转义字符串。目前，Guard 不支持在`string`文字中嵌入转义字符串。
- 验证您的`!=`比较是否比较了兼容的数据类型。例如，a `string` 和 a `int` 是不兼容的数据类型，无法进行比较。执行`!=`比较时，如果值不兼容，则内部会发生错误。当前，错误被抑制并`false`转换为满足 Rus PartialEqt 中的特征。

AWS CloudFormation Guard CLI参数和命令参考

以下全局参数和命令可通过 AWS CloudFormation Guard 命令行界面 (CLI) 获得。

主题

- 保护CLI全局参数
- 解析树
- 规则
- 测试
- 验证

保护CLI全局参数

您可以在任何 AWS CloudFormation Guard CLI命令中使用以下参数。

`-h, --help`

打印帮助信息。

`-V, --version`

打印版本信息。

解析树

为规则文件中定义的 AWS CloudFormation Guard 规则生成解析树。

语法

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

参数

`-h, --help`

打印帮助信息。

`-j, --print-json`

以JSON格式打印输出。

`-y, --print-yaml`

以YAML格式打印输出。

`-V, --version`

打印版本信息。

选项

`-o, --output`

将生成的树写入输出文件。

`-r, --rules`

提供规则文件。

示例

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

规则

取一个 JSON-或 YAML-格式的 AWS CloudFormation 模板文件并自动生成一组与模板资源属性相匹配的 AWS CloudFormation Guard 规则。此命令是开始编写规则或根据已知良好的模板创建 ready-to-use规则的有用方法。

语法

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

```
--template <value>
```

参数

`-h, --help`

打印帮助信息。

`-V, --version`

打印版本信息。

选项

`-o, --output`

将生成的规则写入输出文件。鉴于可能会出现成百甚至数千条规则，我们建议使用此选项。

`-t, --template`

提供JSON或YAML格式的 CloudFormation 模板文件的路径。

示例

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

测试

根据JSON或YAML格式的 Guard 单元测试文件验证 AWS CloudFormation Guard 规则文件，以确定各个规则是否成功。

语法

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

参数

`-h, --help`

打印帮助信息。

`-m, --last-modified`

按目录内上次修改时间排序

`-V, --version`

打印版本信息。

`-v, --verbose`

增加输出详细程度。可以多次指定。

详细输出遵循 Guard 规则文件的结构。规则文件中的每个块都是详细输出中的一个块。最上面的方块是每条规则。如果存在违反该规则的when条件，则它们会显示为同级条件块。

选项

`-r, --rules-file`

提供规则文件的名称。

`-t, --test-data`

为或YAML格式的数据文件提供文件JSON或目录的名称。

args

<alphabetical>

在目录中按字母顺序排序。

示例

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

输出

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

另请参阅

测试警卫规则

验证

根据 AWS CloudFormation Guard 规则验证数据以确定成功或失败。

语法

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

参数

`-a, --alphabetical`

验证按字母顺序排列的目录中的文件。

`-h, --help`

打印帮助信息。

`-m, --last-modified`

验证按上次修改时间排序的目录中的文件。

`-P, --payload`

允许您通过以下方式提供以下JSON格式的规则和数据`stdin`：

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

例如：

```
{"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\"" ]}
```

对于 “规则”，指定规则文件的字符串列表。在 “数据” 中，指定数据文件的字符串列表。

如果您指定标--payload志，请不要指定--rules或--data选项。

-p, --print-json

以JSON格式打印输出。

-s, --show-clause-failures

显示条款失败，包括摘要。

-V, --version

打印版本信息。

-v, --verbose

增加输出详细程度。可以多次指定。

选项

-d，--data（字符串）

为或YAML格式的数据文件提供文件JSON或目录的名称。如果您提供目录，Guard 会针对该目录中的所有数据文件评估指定的规则。该目录必须仅包含数据文件；不能同时包含数据文件和规则文件。

如果您指定了标--payload志，请不要指定该--data选项。

-o，--output-format（字符串）

写入输出文件。

默认值：single-line-summary

允许的值：json | yaml | single-line-summary

-r，--rules（字符串）

提供规则文件或规则文件目录的名称。如果您提供目录，Guard 会根据指定的数据评估该目录中的所有规则。该目录必须仅包含规则文件；不能同时包含数据和规则文件。

如果您指定了标--payload志，请不要指定该--rules选项。

--show-summary（字符串）

指定 Guard 规则评估摘要的详细程度。如果您指定all，Guard 将显示完整摘要。如果您指定pass,fail，Guard 将仅显示通过或失败的规则的摘要信息。如果您指定none，Guard 将不显示摘要信息。默认情况下，指定了 all。

允许的值：all | pass,fail | none

-t，--type（字符串）

提供输入数据的格式。当您指定输入数据类型时，Guard 会在输出中显示 CloudFormation 模板资源的逻辑名称。默认情况下，Guard 会显示属性路径和值，例如Property [/Resources/vol2/Properties/Encrypted。

允许的值：CFNTemplate

示例

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

输出

如果 Guard 成功验证了模板，则该`validate`命令将返回退出状态`0`（`$?`在 bash 中）。如果 Guard 发现了违反规则的情况，则该`validate`命令将返回失败规则的状态报告。使用 verbose 标志 (`-v`) 查看详细的评估树，该树显示 Guard 是如何评估每条规则的。

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

另请参阅

根据防护规则验证输入数据

安全性 AWS CloudFormation Guard

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将此描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于 Guard 的合规计划，请参阅按合规计划划分的范围内的AWSAWS 服务按合规计划。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

以下文档可帮助您了解在将 Guard 作为 AWS Lambda 函数安装 Guard 时如何应用责任共担模型 (cfn-guard-lambda)：

- 《AWS Command Line Interface 用户指南》中的@@ 安全性
- 《AWS Lambda 开发人员指南》中的@@ 安全性
- 《AWS Identity and Access Management 用户指南》中的@@ 安全性

AWS CloudFormation Guard 文档历史记录

下表描述了的文档版本 AWS CloudFormation Guard。

- 上次文档更新日期：2023 年 6 月 30 日
- 最新版本：3.0.0

变更	说明	日期
3.0.0 版本已发布	版本 3.0.0 引入了以下改进： • 更新了 Guard 3.0.0 版本的介绍和安装主题。 • 添加了 Homebrew 和 Chocolatey 的安装说明。 • 更新了与迁移 Guard 规则相关的信息，以反映 Guard 版本 3.0.0 中的更改。 • 添加了指向 AWS CloudFormation Guard GitHub 存储库的醒目链接。	2023 年 6 月 30 日
版本 2.1.3 已发布	版本 2.1.3 引入了以下改进： 已添加有关 Guard 2.1.3 增强功能的信息。对 Guard 2.0 的引用已更新为 Guard 2.1.3。	2023 年 6 月 9 日
2.0.4 版本已发布	版本 2.0.4 引入了以下改进： 该`--payload` 标志已添加到`validate`命令中。 有关更多信息，请参阅 Guard CLI 参考中的验证。	2021 年 10 月 19 日
版本 2.0.3 已发布	版本 2.0.3 引入了以下改进：	2021 年 7 月 27 日

	• 您可以在单元测试文件中为每个测试提供测试名称。有关更多信息，请参阅 测试警卫规则。
	• validate命令中添加了以下选项：
	  • --output-format
	  • --show-summary
	  • --type
	有关更多信息，请参阅 Guard CLI 参考中的验证。	

初始版本	《 AWS CloudFormation Guard 用户指南》的初始版本。	2021 年 7 月 15 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。