ユーザーガイド

# AWS CloudFormation Guard

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# AWS CloudFormation Guard: ユーザーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

# Table of Contents

とは AWS CloudFormation Guard ..... 1
初めての Guard ユーザーですか? ..... 1
ガード機能 ..... 2
CloudFormation フックでの Guard の使用 ..... 2
Guard へのアクセス ..... 3
ベストプラクティス ..... 3
Guard のセットアップ ..... 4
Linux および macOS の場合 ..... 4
ビルド済みリリースバイナリから Guard をインストールする ..... 4
Cargo から Guard をインストールする ..... 5
Homebrew から Guard をインストールする ..... 6
Windows の場合 ..... 6
前提条件 ..... 6
Cargo から Guard をインストールする ..... 5
Chocolatey から Guard をインストールする ..... 8
AWS Lambda 関数として ..... 8
前提条件 ..... 8
Rust パッケージマネージャーをインストールする ..... 9
Guard を Lambda 関数としてインストールするには ..... 9
を構築して実行するには ..... 10
Lambda 関数の呼び出し ..... 11
ガードルールを使用するための前提条件と概要 ..... 12
前提条件 ..... 12
ガードルールの使用の概要 ..... 12
ガードルールの記述 ..... 13
句 ..... 13
句でのクエリの使用 ..... 16
句での演算子の使用 ..... 16
句でのカスタムメッセージの使用 ..... 19
句の組み合わせ ..... 20
ガードルールでのブロックの使用 ..... 21
組み込み関数の使用 ..... 24
クエリとフィルタリングの定義 ..... 25
ガードルールでの変数の割り当てと参照 ..... 39

名前付きルールブロックの作成 ..... 46
コンテキスト対応評価を実行するための句の記述 ..... 52
ガードルールのテスト ..... 65
前提条件 ..... 65
概要: ..... 65
演習 ..... 67
Guard ルールでの入力パラメータの使用 ..... 76
の使用方法 ..... 76
使用例 ..... 76
複数の入力パラメータ ..... 78
Guard ルールに対する入力データの検証 ..... 78
前提条件 ..... 78
validate コマンドの使用 ..... 78
複数のデータファイルに対する複数のルールの検証 ..... 79
Guard のトラブルシューティング ..... 81
選択したタイプのリソースが存在しない場合、句は失敗します ..... 81
Guard が CloudFormation テンプレートを評価しない ..... 81
一般的なトラブルシューティングのトピック ..... 82
Guard CLI リファレンス ..... 83
Guard CLI グローバルパラメータ ..... 83
解析ツリー ..... 83
構文 ..... 83
パラメータ ..... 84
オプション ..... 84
例 ..... 84
rulegen ..... 84
構文 ..... 84
パラメータ ..... 85
オプション ..... 85
例 ..... 85
test ..... 85
構文 ..... 85
パラメータ ..... 86
オプション ..... 86
例 ..... 87
Output ..... 87

関連情報 ..... 87
validate ..... 87
構文 ..... 87
パラメータ ..... 87
オプション ..... 89
例 ..... 90
Output ..... 90
関連情報 ..... 91
セキュリティ ..... 92
ドキュメント履歴 ..... 93
AWS 用語集 ..... 96
..... xcvii

# とは AWS CloudFormation Guard

AWS CloudFormation Guard は、オープンソースの汎用ポリシーアズコード評価ツールです。Guard コマンドラインインターフェイス (CLI) は、ポリシーをコードとして表現するために使用できる simple-to-use宣言的なドメイン固有の言語 (DSL) を提供します。さらに、CLI コマンドを使用して、構造化された階層 JSON または YAML データをそれらのルールに対して検証できます。Guard には、ルールが意図したとおりに機能することを検証するためのユニットテストフレームワークも組み込まれています。

Guard は、有効な構文または許可されたプロパティ値について CloudFormation テンプレートを検証しません。cfn-lint ツールを使用して、テンプレート構造を徹底的に検査できます。

Guard はサーバー側の強制を提供しません。CloudFormation フックを使用して、 オペレーションをブロックまたは警告できるサーバー側の検証と適用を実行できます。

AWS CloudFormation Guard 開発の詳細については、Guard GitHub リポジトリを参照してください。

トピック


- 初めての Guard ユーザーですか?
- ガード機能
- CloudFormation フックでの Guard の使用
- Guard へのアクセス
- ベストプラクティス


## 初めての Guard ユーザーですか?

Guard を初めて使用する場合は、まず以下のセクションを読むことをお勧めします。

- Guard のセットアップ – このセクションでは、Guard をインストールする方法について説明します。Guard を使用すると、Guard DSL を使用してポリシールールを記述し、JSON 形式または YAML 形式の構造化データをそれらのルールに対して検証できます。
- ガードルールの記述 – このセクションでは、ポリシールールを記述するための詳細なチュートリアルを提供します。

- ガードルールのテスト – このセクションでは、ルールをテストして意図したとおりに動作することを検証し、JSON 形式または YAML 形式の構造化データをルールに照らして検証するための詳細なチュートリアルを提供します。
- Guard ルールに対する入力データの検証 – このセクションでは、JSON 形式または YAML 形式の構造化データをルールに照らして検証するための詳細なチュートリアルを提供します。
- Guard CLI リファレンス – このセクションでは、Guard CLI で使用できるコマンドについて説明します。

# ガード機能

Guard を使用すると、JSON 形式または YAML 形式の構造化データを検証するためのポリシールールを記述できます。 CloudFormation これにはテンプレートが含まれますが、これらに限定されません。Guard は、ポリシーチェックのend-to-end評価のスペクトル全体をサポートします。ルールは、以下のビジネスドメインで役立ちます。

- 予防的ガバナンスとコンプライアンス (シフト左テスト) — セキュリティとコンプライアンスに関する組織のベストプラクティスを表すポリシールールに照らして、Infrastructure as Code (IaC) またはインフラストラクチャとサービスの構成を検証します。例えば、CloudFormation テンプレート、CloudFormation 変更セット、JSON ベースの Terraform 設定ファイル、Kubernetes 設定を検証できます。
- 検出ガバナンスとコンプライアンス – AWS Configベースの設定項目 (CIs。たとえば、開発者は AWS Config CIs に対する Guard ポリシーを使用して、デプロイされたAWS リソース AWS と非リソースの状態を継続的にモニタリングし、ポリシーからの違反を検出して修復を開始できます。
- デプロイの安全性 – デプロイ前に変更が安全であることを確認します。例えば、CloudFormation の変更セットをポリシールールと照合して検証し、Amazon DynamoDB テーブルの名前変更など、リソースの置き換えにつながる変更を防止します。

# CloudFormation フックでの Guard の使用

CloudFormation Guard を使用して、CloudFormation フックでフックを作成できます。CloudFormation フックを使用すると、CloudFormation がオペレーションを作成、更新、または削除し、オペレーション AWS クラウドコントロール API を作成または更新する前に、Guard ルールを事前に適用できます。フックにより、リソース設定が組織のセキュリティ、運用、コスト最適化のベストプラクティスに準拠していることを確認できます。

Guard を使用して CloudFormation ガードフックを作成する方法の詳細については、「 フックユーザーガイド」の「ガードフックのリソースを評価するためのガードルールの記述」を参照してください。 CloudFormation

## Guard へのアクセス

Guard DSL および コマンドにアクセスするには、Guard CLI をインストールする必要があります。Guard CLI のインストールについては、「」を参照してくださいGuard のセットアップ。

## ベストプラクティス

シンプルなルールを作成し、名前付きルールを使用して他のルールで参照します。複雑なルールは、保守とテストが困難な場合があります。

# セットアップ AWS CloudFormation Guard

AWS CloudFormation Guard はオープンソースのコマンドラインインターフェイス (CLI) です。これは、ポリシールールを記述し、それらのルールに対して構造化された階層 JSON および YAML データを検証するためのシンプルなドメイン固有の言語を提供します。ルールは、セキュリティ、コンプライアンスなどに関連する会社のポリシーガイドラインを表すことができます。構造化階層データは、コードとして記述されたクラウドインフラストラクチャを表すことができます。たとえば、CloudFormation テンプレートで暗号化された Amazon Simple Storage Service (Amazon S3) バケットを常にモデル化するためのルールを作成できます。

以下のトピックでは、選択したオペレーティングシステムまたは AWS Lambda 関数を使用して Guard をインストールする方法について説明します。

トピック


- Linux および macOS 用の Guard のインストール
- Windows 用 Guard のインストール
- AWS Lambda 関数としての Guard のインストール


## Linux および macOS 用の Guard のインストール

Linux および macOS AWS CloudFormation Guard 用の をインストールするには、構築済みのリリースバイナリ、Cargo、または Homebrew を使用します。

### ビルド済みリリースバイナリから Guard をインストールする

ビルド済みのバイナリから Guard をインストールするには、次の手順に従います。

1. ターミナルを開き、次のコマンドを実行します。

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

2. 次のコマンドを実行して、PATH変数を設定します。

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

結果: Guard が正常にインストールされ、 PATH変数が設定されました。

- (オプション) Guard のインストールを確認するには、次のコマンドを実行します。

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

このコマンドは、以下の出力を返します。

```
cfn-guard 3.1.2
```

## Cargo から Guard をインストールする

貨物は Rust パッケージマネージャーです。Cargo を含む Rust をインストールするには、次の手順を実行します。次に、Cargo から Guard をインストールします。

1. ターミナルから次のコマンドを実行し、画面の指示に従って Rust をインストールします。

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

- (オプション) Ubuntu 環境の場合は、次のコマンドを実行します。

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

2. PATH 環境変数を設定し、次のコマンドを実行します。

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

3. Cargo がインストールされたら、次のコマンドを実行して Guard をインストールします。

```
cargo install cfn-guard
```

結果: Guard が正常にインストールされました。

- (オプション) Guard のインストールを確認するには、次のコマンドを実行します。

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

このコマンドは、以下の出力を返します。

```
cfn-guard 3.1.2
```

## Homebrew から Guard をインストールする

Homebrew は、macOS および Linux 用のパッケージマネージャーです。Homebrew をインストールするには、次の手順を実行します。次に、Homebrew から Guard をインストールします。

1. ターミナルから次のコマンドを実行し、画面の指示に従って Homebrew をインストールします。

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

2. Homebrew がインストールされたら、次のコマンドを実行して Guard をインストールします。

   ```
   brew install cloudformation-guard
   ```

   結果: Guard が正常にインストールされました。

   - (オプション) Guard のインストールを確認するには、次のコマンドを実行します。

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

     このコマンドは、以下の出力を返します。

     ```
     cfn-guard 3.1.2
     ```

# Windows 用 Guard のインストール

Windows AWS CloudFormation Guard 用 は、Cargo または Chocolatey を通じてインストールできます。

## 前提条件

コマンドラインインターフェイスから Guard をビルドするには、Build Tools for Visual Studio 2019 をインストールする必要があります。

1. Build Tools for Visual Studio 2019 ウェブサイトから Microsoft Visual C++ ビルドツールをダウンロードします。
2. インストーラを実行し、デフォルトを選択します。

## Cargo から Guard をインストールする

Cargo は Rust パッケージマネージャーです。Cargo を含む Rust をインストールするには、次の手順を実行します。次に、Cargo から Guard をインストールします。

1. Rust をダウンロードし、rustup-init.exe を実行します。
2. コマンドプロンプトから、デフォルトのオプションである 1 を選択します。

   このコマンドは、以下の出力を返します。

   ```
   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. インストールを完了するには、Enter キーを押します。
4. Cargo がインストールされたら、次のコマンドを実行して Guard をインストールします。

   ```
   cargo install cfn-guard
   ```

   結果: Guard が正常にインストールされました。

   - (オプション) Guard のインストールを確認するには、次のコマンドを実行します。

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

     このコマンドは、以下の出力を返します。

     ```
     cfn-guard 3.1.2
     ```

## Chocolatey から Guard をインストールする

Chocolatey は Windows のパッケージマネージャーです。Chocolatey をインストールするには、次の手順を実行します。次に、Chocolatey から Guard をインストールします。

1. このガイドに従って Chocolatey をインストールします。
2. Chocolatey がインストールされたら、次のコマンドを実行して Guard をインストールします。

```
choco install cloudformation-guard
```

結果: Guard が正常にインストールされました。

- (オプション) Guard のインストールを確認するには、次のコマンドを実行します。

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

このコマンドは、以下の出力を返します。

```
cfn-guard 3.1.2
```

# AWS Lambda 関数としての Guard のインストール

Rust パッケージマネージャーである Cargo AWS CloudFormation Guard を使用して をインストールできます。Guard as an AWS Lambda function (`cfn-guard-lambda`) は、Lambda 関数として使用できる Guard (`cfn-guard`) を囲む軽量ラッパーです。

## 前提条件

Guard を Lambda 関数としてインストールする前に、次の前提条件を満たす必要があります。

- AWS Command Line Interface (AWS CLI) Lambda 関数をデプロイして呼び出すためのアクセス許可で設定されています。詳細については、「Configuring the AWS CLI」を参照してください。
- AWS Identity and Access Management (IAM) AWS Lambda の実行ロール。詳細については、AWS Lambda 「実行ロール」を参照してください。
- CentOS/RHEL 環境では、`musl-libc`パッケージリポジトリを yum 設定に追加します。詳細については、「ngompa/musl-libc」を参照してください。

## Rust パッケージマネージャーをインストールする

Cargo は Rust パッケージマネージャーです。Cargo を含む Rust をインストールするには、次の手順を実行します。

1. ターミナルから次のコマンドを実行し、画面の指示に従って Rust をインストールします。

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

   - (オプション) Ubuntu 環境の場合は、次のコマンドを実行します。

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

2. PATH 環境変数を設定し、次のコマンドを実行します。

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

## Lambda 関数として Guard をインストールする (Linux、macOS、または Unix)

Guard を Lambda 関数としてインストールするには、次の手順を実行します。

1. コマンドターミナルから、次のコマンドを実行します。

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

   - (オプション) Lambda 関数としての Guard のインストールを確認するには、次のコマンドを実行します。

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

     このコマンドは、以下の出力を返します。

```
cfn-guard-lambda 3.1.2
```

2. musl サポートをインストールするには、次のコマンドを実行します。

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

3. でビルドしmusl、ターミナルで次のコマンドを実行します。

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

カスタムランタイムの場合、 にはデプロイパッケージ .zip ファイルbootstrapの名前の実行可能ファイル AWS Lambda が必要です。生成されたcfn-lambda実行可能ファイルの名前を に変更しbootstrap、.zip アーカイブに追加します。

- macOS 環境の場合は、Rust プロジェクトのルートまたは に貨物設定ファイルを作成します~/.cargo/config。

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

4. をcfn-guard-lambdaルートディレクトリに変更します。

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

5. ターミナルで次のコマンドを実行します。

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

6. 次のコマンドを実行して、Lambda 関数cfn-guardとして をアカウントに送信します。

```
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
```

## Lambda 関数として Guard を構築して実行するには

Lambda 関数cfn-guard-lambdaとして送信された を呼び出すには、次のコマンドを実行します。

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

## Lambda 関数リクエスト構造を呼び出すには

次のフィールドcfn-guard-lambdaを要求するリクエスト:

- data – YAML または JSON テンプレートの文字列バージョン
- rules – ルールセットファイルの文字列バージョン

# ガードルールを使用するための前提条件と概要

このセクションでは、JSON または YAML 形式のデータに対してルールを記述、テスト、検証するコア Guard タスクを完了する方法について説明します。さらに、特定のユースケースに対応するルールの記述を示す詳細なチュートリアルも含まれています。

トピック


- 前提条件
- ガードルールの使用の概要
- AWS CloudFormation Guard ルールの記述
- AWS CloudFormation Guard ルールのテスト
- AWS CloudFormation Guard ルールでの入力パラメータの使用
- AWS CloudFormation Guard ルールに対する入力データの検証


## 前提条件

Guard ドメイン固有の言語 (DSL) を使用してポリシールールを記述する前に、Guard コマンドラインインターフェイス (CLI) をインストールする必要があります。詳細については、「Guard のセットアップ」を参照してください。

## ガードルールの使用の概要

Guard を使用する場合は、通常、次の手順を実行します。

1. 検証する JSON 形式または YAML 形式のデータを書き込みます。
2. Guard ポリシールールを記述します。詳細については、「ガードルールの記述」を参照してください。
3. Guard `test` コマンドを使用して、ルールが意図したとおりに動作することを確認します。ユニットテストの詳細については、「」を参照してくださいガードルールのテスト。
4. Guard `validate` コマンドを使用して、ルールに対して JSON 形式または YAML 形式のデータを検証します。詳細については、「Guard ルールに対する入力データの検証」を参照してください。

# AWS CloudFormation Guard ルールの記述

では AWS CloudFormation Guard、ルールは policy-as-code ルールです。JSON 形式または YAML 形式のデータを検証できる Guard ドメイン固有の言語 (DSL) でルールを記述します。ルールは 句で構成されます。

Guard DSL を使用して記述されたルールは、任意のファイル拡張子を使用するプレーンテキストファイルに保存できます。

複数のルールファイルを作成し、ルールセットとして分類できます。ルールセットを使用すると、JSON 形式または YAML 形式のデータを複数のルールファイルに対して同時に検証できます。

トピック


- 句
- 句でのクエリの使用
- 句での演算子の使用
- 句でのカスタムメッセージの使用
- 句の組み合わせ
- ガードルールでのブロックの使用
- 組み込み関数の使用
- ガードクエリの定義とフィルタリング
- ガードルールでの変数の割り当てと参照
- で名前付きルールブロックを作成する AWS CloudFormation Guard
- コンテキスト対応評価を実行するための句の記述


## 句

句は、true (PASS) または false () のいずれかに評価されるブール式ですFAIL。句は、バイナリ演算子を使用して 2 つの値を比較するか、単一の値で動作する単一演算子を使用します。

単項句の例

次の unary 句は、コレクションTcpBlockedPortsが空かどうかを評価します。

```
InputParameters.TcpBlockedPorts not empty
```

次の unary 句は、 ExecutionRoleArnプロパティが文字列であるかどうかを評価します。

```
Properties.ExecutionRoleArn is_string
```

バイナリ句の例

次のバイナリ句は、大文字と小文字に関係なくencrypted、 BucketNameプロパティに文字列 が含まれているかどうかを評価します。

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

次のバイナリ句は、ReadCapacityUnitsプロパティが 5,000 以下であるかどうかを評価します。

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

## ガードルール句を記述するための構文

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

## ガードルール句のプロパティ

query

階層データをトラバースするために書き込まれたドット (.) 区切り式。クエリ式には、値のサブセットをターゲットとするフィルター式を含めることができます。クエリを変数に割り当てると、変数を 1 回書き込み、ルールセットの他の場所で参照できるため、クエリ結果にアクセスできます。

クエリの記述とフィルタリングの詳細については、「」を参照してくださいクエリとフィルタリングの定義。

必須: はい

operator

クエリの状態を確認するのに役立つ単項演算子またはバイナリ演算子。バイナリ演算子の左側 (LHS) はクエリで、右側 (RHS) はクエリまたは値リテラルである必要があります。

サポートされているバイナリ演算子: == (Equal) | != (Not equal) | > (Greater than) | >= (Greater than or equal to) | < (Less than) | <= (Less than than) | IN (In a list of form [x, y, z]

サポートされている単項演算子: exists | empty | is_string | is_list | is_struct | not(!)

必須: はい

query|value literal

クエリ、または string や などのサポートされている値リテラルinteger(64)。

サポートされている値リテラル:

- すべてのプリミティブタイプ: string、integer(64)、float(64)、bool、char、 regex
- integer(64)、、float(64)または 範囲を次のように表現するためのすべての特殊なchar範囲タイプ:
  - r[<lower_limit>, <upper_limit>]。これは、次の式kを満たす任意の値に変換されます。 lower_limit <= k <= upper_limit
  - r[<lower_limit>, <upper_limit>)。これは、次の式kを満たす任意の値に変換されます。 lower_limit <= k < upper_limit
  - r(<lower_limit>, <upper_limit>]。これは、次の式kを満たす任意の値に変換されます。 lower_limit < k <= upper_limit
  - r(<lower_limit>, <upper_limit>), これは、次の式kを満たす任意の値に変換されます。 lower_limit < k < upper_limit
- ネストされたキーと値の構造データの関連付け配列 (マップ ) 。例えば、次のようになります。

  ```
  { "my-map": { "nested-maps": [ { "key": 10, "value": 20 } ] } }
  ```

- プリミティブ型または関連付け配列型の配列

必須: 条件付き。バイナリ演算子を使用する場合は必須です。

custom message

句に関する情報を提供する文字列。メッセージは validateおよび test コマンドの詳細出力に表示され、階層データのルール評価の理解やデバッグに役立ちます。

必須: いいえ

## 句でのクエリの使用

クエリの記述の詳細については、クエリとフィルタリングの定義「」および「」を参照してくださいガードルールでの変数の割り当てと参照。

## 句での演算子の使用

CloudFormation テンプレートの例を次に示しますTemplate-1Template-2。サポートされている演算子の使用を示すために、このセクションのクエリと句の例は、これらのサンプルテンプレートを参照しています。

Template-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
```

Template-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
```

## 単項演算子を使用する句の例

- empty – コレクションが空かどうかを確認します。これを使用して、クエリがコレクションになるため、クエリに階層データの値があるかどうかを確認することもできます。文字列値クエリに空の文字列 ("") が定義されているかどうかを確認するために使用することはできません。詳細については、「クエリとフィルタリングの定義」を参照してください。

  次の句は、テンプレートに 1 つ以上のリソースが定義されているかどうかを確認します。論理 ID を持つリソースS3Bucketが で定義されているPASSため、 に評価されますTemplate-1。

  ```
  Resources !empty
  ```

  次の句は、S3Bucketリソースに 1 つ以上のタグが定義されているかどうかを確認します。S3Bucket には の Tagsプロパティに 2 つのタグが定義されているPASSため、 に評価されますTemplate-1。

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

- exists – クエリの各出現に値があり、 の代わりに使用できるかどうかを確認します!= null。

  次の句は、 BucketEncryptionプロパティが に対して定義されているかどうかを確認しますS3Bucket。は S3Bucketで にBucketEncryption定義されているPASSため、 に評価されますTemplate-1。

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

> **Note**
>
> empty と not existsは、入力データをトラバースするときに、欠落しているプロパティキーtrueがないかを に評価します。たとえば、 のテンプレートで Propertiesセクションが定義されていない場合S3Bucket、 句は にResources.S3Bucket.Properties.Tag empty評価されますtrue。exists および emptyチェックでは、ドキュメント内の JSON

ポインタパスはエラーメッセージに表示されません。これらの句の両方に、このトラバーサル情報を保持しない取得エラーがあることがよくあります。

- is_string – クエリの各出現が stringタイプであるかどうかを確認します。

  次の句は、 S3Bucketリソースの BucketNameプロパティに文字列値が指定されているかどうかをチェックします。文字列値が BucketNameの に"MyServiceS3Bucket"指定されているPASSため、 に評価されますTemplate-1。

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

- is_list – クエリの各出現が listタイプであるかどうかを確認します。

  次の句は、S3Bucketリソースの Tagsプロパティにリストが指定されているかどうかを確認します。2 つのキーと値のペアが Tagsで に指定されているPASSため、 に評価されますTemplate-1。

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

- is_struct – クエリの各出現が構造化データであるかどうかをチェックします。

  次の句は、S3Bucketリソースの BucketEncryptionプロパティに構造化データが指定されているかどうかを確認します。BucketEncryption で ServerSideEncryptionConfigurationプロパティタイプ *(######)* を使用して が指定されPASSるため、 に評価されますTemplate-1。

  ```
  Resources.S3Bucket.Properties.BucketEncryption is_struct
  ```

> **Note**
>
> 逆状態を確認するには、( not !) 演算子を is_string、is_list、および is_struct演算子で使用できます。

## バイナリ演算子を使用する句の例

次の句は、 のS3Bucketリソースの BucketNameプロパティに指定された値に、大文字と小文字に関係なくencrypt文字列 Template-1が含まれているかどうかを確認します。指定された

バケット名に文字列 が含まれ"MyServiceS3Bucket"ていないPASSため、これは に評価されますencrypt。

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

次のTemplate-2句は、 のNewVolumeリソースの Sizeプロパティに指定された値が、50 <= Size <= 200 の範囲内にあるかどうかを確認します。が に100指定されているPASSため、 に評価されますSize。

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

次のTemplate-2句は、 のNewVolumeリソースの VolumeTypeプロパティに指定された値が io1、io2、または であるかどうかをチェックしますgp3。が にio1指定されているPASSため、 に評価されますNewVolume。

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

> **Note**
>
> このセクションのクエリ例は、論理 IDs S3Bucketと を持つリソースを使用した演算子の使用を示していますNewVolume。多くの場合、リソース名はユーザー定義であり、Infrastructure as Code (IaC) テンプレートで任意の名前を付けることができます。汎用的で、テンプレートで定義されているすべてのAWS::S3::Bucketリソースに適用されるルールを記述する場合、使用されるクエリの最も一般的な形式は ですResources.*[ Type == ‘AWS::S3::Bucket’ ]。詳細については、使用状況の詳細についてはクエリとフィルタリングの定義「」を参照し、cloudformation-guardGitHub リポジトリの examples ディレクトリを参照してください。

## 句でのカスタムメッセージの使用

次の例では、 の 句にカスタムメッセージTemplate-2が含まれています。

```
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>>
```

## 句の組み合わせ

Guard では、新しい行に書き込まれた各句は、結合 (ブールand口ジック) を使用して次の句と暗黙的に結合されます。次の例を参照してください。

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

差分を使用して、最初の句のor|OR最後に を指定することで、句を次の句と組み合わせることもできます。

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

Guard 句では、差分が最初に評価され、次に結合が評価されます。ガードルールは、 (or|OR) または true () のいずれかに評価される句 ( and|AND の PASS) の分散の組み合わせとして定義できますfalseFAIL。これは、補助的な通常の形式に似ています。

次の例は、 句の評価の順序を示しています。

```
# (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
```

この例に基づくすべての句は、結合を使用して結合Template-1できます。次の例を参照してください。

```
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
```

## ガードルールでのブロックの使用

ブロックは、関連する句、条件、またはルールのセットから冗長性と繰り返しを削除する構成です。ブロックには 3 つのタイプがあります。

- クエリブロック
- when ブロック
- 名前付きルールブロック

### クエリブロック

以下は、例 に基づく句ですTemplate-1。句を組み合わせるために結合が使用されました。

```
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
```

各句のクエリ式の一部が繰り返されます。クエリブロックを使用して、同じ初期クエリパスを持つ一連の関連句から、コンポジビリティを向上させ、冗長性と繰り返しを削除できます。次の例に示すように、同じ句のセットを記述できます。

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

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

クエリブロックでは、ブロックの前のクエリがブロック内の句のコンテキストを設定します。

ブロックの使用の詳細については、「」を参照してください名前付きルールブロックの作成。

### when ブロック

ブロックは、次の形式のwhenブロックを使用して条件付きで評価できます。

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

when キーワードはwhenブロックの開始を指定します。 conditionはガードルールです。ブロックは、条件の評価が true () になった場合にのみ評価されますPASS。

以下は、 に基づくwhenブロックの例ですTemplate-1。

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

when ブロック内の 句は、 に指定された値が文字列である場合にのみ評価BucketNameされます。次の例に示すように、 に指定された値がテンプレートの Parametersセクションで参照BucketNameされている場合、 whenブロック内の 句は評価されません。

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

## 名前付きルールブロック

ルールのセット (ルールセット) に名前を割り当て、他のルールで名前付きルールブロックと呼ばれるモジュラー検証ブロックを参照できます。名前付きルールブロックの形式は次のとおりです。

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

rule キーワードは、named-rule ブロックの開始を指定します。

rule name は、名前付きルールブロックを一意に識別する人間が読める文字列です。これは、カプセル化する Guard ルールセットのラベルです。この使用では、Guard ルールという用語に句、クエリブロック、whenブロック、名前付きルールブロックが含まれます。ルール名は、カプセル化するルールセットの評価結果を参照するために使用できます。これにより、名前付きルールブロックを再利用できます。ルール名は、 validateおよび test コマンド出力のルールの失敗に関するコンテキストも提供します。ルール名は、ルールファイルの評価出力にブロックの評価ステータス (PASS、FAIL、または SKIP) とともに表示されます。次の例を参照してください。

```
# Sample output of an evaluation where check1, check2, and check3 are rule names.
template.json Status = **FAIL**
**SKIP rules**
check1 **SKIP**
**PASS rules**
check2 **PASS**
**FAILED rules**
check3 **FAIL**
```

when キーワードを指定し、その後にルール名の後に条件を指定することで、名前付きルールブロックを条件付きで評価することもできます。

このトピックで前述したwhenブロックの例を次に示します。

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

名前付きルールブロックを使用すると、前述の内容を次のように記述することもできます。

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

名前付きルールブロックを再利用して、他の Guard ルールでグループ化できます。以下にいくつかの例を示します。

```
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
    ...
}
```

## 組み込み関数の使用

AWS CloudFormation Guard には、文字列操作、JSON 解析、データ型変換などのオペレーションを実行するためにルールで使用できる組み込み関数が用意されています。関数は、変数への割り当てによってのみサポートされます。

### キー関数

`json_parse(json_string)`

テンプレートからインライン JSON 文字列を解析します。解析後、結果のオブジェクトのプロパティを評価できます。

`count(collection)`

クエリが解決される項目の数を返します。

`regex_replace(base_string, regex_to_extract, regex_replacement)`

正規表現を使用して文字列の一部を置き換えます。

文字列操作、コレクションオペレーション、データ型変換関数など、使用可能な関数の完全なリストについては、Guard GitHub リポジトリの Functions ドキュメントを参照してください。

## ガードクエリの定義とフィルタリング

このトピックでは、クエリの記述と、Guard ルール句の記述時のフィルタリングの使用について説明します。

### 前提条件

フィルタリングは高度な AWS CloudFormation Guard 概念です。フィルタリングについて学習する前に、以下の基本的なトピックを確認することをお勧めします。

- とは AWS CloudFormation Guard
- ルール、句の記述

### クエリの定義

クエリ式は、階層データをトラバースするために書き込まれる単純なドット (.) で区切られた式です。クエリ式には、値のサブセットをターゲットとするフィルター式を含めることができます。クエリが評価されると、SQL クエリから返された結果セットと同様に、値のコレクションが生成されます。

次のクエリ例では、 CloudFormation テンプレートで`AWS::IAM::Role`リソースを検索します。

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

クエリは、次の基本原則に従います。

- クエリの各ドット (.) 部分は、 Resourcesや Properties.Encrypted. などの明示的なキー用語が使用されると階層を通過します。クエリのいずれかの部分が受信データムと一致しない場合、Guard は取得エラーをスローします。
- ワイルドカードを使用するクエリのドット (.) 部分は、そのレベルで構造体のすべての値を*横断します。
- 配列ワイルドカードを使用するクエリのドット (.) 部分は、その配列のすべてのインデックス[*]を横断します。
- すべてのコレクションは、角括弧 内でフィルターを指定することでフィルタリングできます[]。コレクションは、次の方法で検出できます。
  - データムで自然に発生する配列はコレクションです。 の例を次に示します。

    ポート: [20, 21, 110, 190]

    タグ: [{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]
  - のような構造のすべての値をトラバースする場合 Resources.*
  - クエリ結果自体は、値をさらにフィルタリングできるコレクションです。次の例を参照してください。

```
# 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
}
```

CloudFormation テンプレートスニペットの例を次に示します。

```
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
    ...
```

このテンプレートに基づいて、トラバースされるパスは SampleRoleで、選択した最終値は です`Type: AWS::IAM::Role`。

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

YAML `Resources.*[ Type == 'AWS::IAM::Role' ]` 形式のクエリの結果の値を次の例に示します。

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

クエリを使用する方法には、次のようなものがあります。

- 変数にクエリを割り当てて、それらの変数を参照してクエリ結果にアクセスできるようにします。
- 選択した各値に対してテストするブロックを使用してクエリに従います。
- クエリを基本句と直接比較します。

## 変数へのクエリの割り当て

Guard は、特定のスコープ内でワンショット変数の割り当てをサポートします。ガードルールの変数の詳細については、「」を参照してくださいガードルールでの変数の割り当てと参照。

変数にクエリを割り当てると、クエリを 1 回書き込み、Guard ルールの他の場所で参照できます。このセクションで後述するクエリ原則を示す変数割り当ての例を以下に示します。

```
#
# 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'
    }
]
```

## クエリに割り当てられた変数の値を直接ループスルーする

Guard は、クエリの結果に対する直接実行をサポートしています。次の例では、 whenブロックは CloudFormation テンプレートにある各AWS::EC2::Volumeリソースの Encrypted、VolumeType、および AvailabilityZoneプロパティに対してテストします。

```
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']
        }
    }
}
```

## 句レベルの直接比較

Guard は、直接比較の一部としてクエリもサポートしています。例えば、次のようになります。

```
let resources = Resources.*
```

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

前の例では、示されている形式で表される 2 つの句 ( someキーワードで始まる) は独立した句と見なされ、個別に評価されます。

単一句とブロック句の形式

まとめると、前のセクションで示した 2 つのサンプル句は、次のブロックと同等ではありません。

```
let resources = Resources.*

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

このブロックは、コレクション内の各Tag値をクエリし、そのプロパティ値を予想されるプロパティ値と比較します。前のセクションの句の組み合わせ形式は、2 つの句を個別に評価します。次の入力を検討してください。

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

最初の形式の句は に評価されますPASS。最初の形式で最初の句を検証する場合、Resources、、Properties、および の次のパスNotPRODEndは 値Keyと一致しTags、想定値 と一致しませんPROD。

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

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

最初の形式の 2 番目の句でも同じことが起こります。Resources、Properties、、および のパスはTags、値 Valueと一致しますAppStart。その結果、2 番目の句は独立しています。

全体的な結果は ですPASS。

ただし、ブロックフォームは次のように評価されます。Tags 値ごとに、 Keyと の両方Valueが一致しているかどうかを比較します。次の例では、 NotAppStartと NotPRODEndの値は一致しません。

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

評価では Key == /PROD$/と の両方がチェックされるためValue == /^App/、一致は完了しません。したがって、結果は ですFAIL。

> **Note**
>
> コレクションを使用する場合は、コレクション内の各要素の複数の値を比較するときに、 ブロック句フォームを使用することをお勧めします。コレクションがスカラー値のセットである場合、または単一の属性のみを比較する場合は、単一句フォームを使用します。

## クエリ結果と関連句

すべてのクエリは値のリストを返します。キーの欠落、すべてのインデックスへのアクセス時の配列の空の値 (Tags: [])、空のマップ () に遭遇したときのマップの欠損値など、トラバーサルのどの部分でも、取得エラーが発生するResources: {}可能性があります。

このようなクエリに対して句を評価する場合、すべての取得エラーは失敗と見なされます。唯一の例外は、クエリで明示的なフィルターを使用する場合です。フィルターを使用すると、関連する句はスキップされます。

次のブロック障害は、実行中のクエリに関連付けられています。

- テンプレートにリソースが含まれていない場合、クエリは に評価されFAIL、関連するブロックレベルの句も に評価されますFAIL。
- テンプレートに のような空のリソースブロックが含まれている場合{ "Resources": {} }、クエリは に評価されFAIL、関連するブロックレベルの句も に評価されますFAIL。
- テンプレートにリソースが含まれているが、クエリに一致するものがない場合、クエリは空の結果を返し、ブロックレベルの句はスキップされます。

## クエリでのフィルターの使用

クエリのフィルターは、実質的に選択基準として使用されるガード句です。以下は、 句の構造です。

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

フィルターAWS CloudFormation Guard ルールの記述を使用する際の以下の重要なポイントに注意してください。

- Conjunctive Normal Form (CNF) を使用して句を結合します。
- 新しい行で各連結 (and) 句を指定します。
- 2 つの句間のorキーワードを使用して、差分 (or) を指定します。

次の例は、結合句と非結合句を示しています。

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

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

### 選択基準に句を使用する

任意のコレクションにフィルタリングを適用できます。フィルタリングは、 などのコレクションがすでに存在する入力内の属性に直接適用できますsecurityGroups: [....]。常に値のコレクションであるクエリにフィルタリングを適用することもできます。結合法線形式を含む句のすべての機能をフィルタリングに使用できます。

次の一般的なクエリは、CloudFormation テンプレートからタイプ別にリソースを選択するときによく使用されます。

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

クエリは、入力の Resourcesセクションに存在するすべての値Resources.*を返します。のテンプレート入力例の場合クエリの定義、クエリは以下を返します。

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

次に、このコレクションにフィルターを適用します。一致させる基準は ですType == AWS::IAM::Role。以下は、フィルターが適用された後のクエリの出力です。

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

次に、AWS::IAM::Roleリソースのさまざまな句を確認します。

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

以下は、すべての AWS::IAM::Policyおよび AWS::IAM::ManagedPolicyリソースを選択するフィルタリングクエリの例です。

```
Resources.*[
```

```
    Type in [ /IAM::Policy/,
              /IAM::ManagedPolicy/ ]
]
```

次の例では、これらのポリシーリソースに PolicyDocumentが指定されているかどうかを確認します。

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

### より複雑なフィルタリングニーズの構築

イングレスおよびエグレスセキュリティグループ情報 AWS Config の設定項目の次の例を考えてみましょう。

```
---
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
```

次の点に注意してください。

- ipPermissions (進入ルール) は、設定ブロック内のルールのコレクションです。
- 各ルール構造には、CIDR ブロックのコレクションを指定する ipv6Ranges ipv4Rangesや などの属性が含まれています。

任意の IP アドレスからの接続を許可する進入ルールを選択し、ルールで TCP ブロックされたポートを公開できないことを確認するルールを記述しましょう。

次の例に示すように、IPv4 を対象とするクエリ部分から始めます。

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

some キーワードはこのコンテキストで便利です。すべてのクエリは、クエリに一致する値のコレクションを返します。デフォルトでは、Guard は、クエリの結果として返されたすべての値がチェッ

クと照合されると評価します。ただし、この動作が常にチェックに必要な動作であるとは限りません。設定項目からの入力の次の部分を考慮してください。

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

には 2 つの値がありますipv4Ranges。すべてのipv4Ranges値が で表される IP アドレスと等しいわけではありません0.0.0.0/0。少なくとも 1 つの値が と一致するかどうかを確認します0.0.0.0/0。クエリから返されたすべての結果が一致する必要はありませんが、少なくとも 1 つの結果が一致する必要があることを Guard に伝えます。some キーワードは、結果のクエリの 1 つ以上の値がチェックと一致するように Guard に指示します。一致するクエリ結果値がない場合、Guard はエラーをスローします。

次に、次の例に示すように IPv6 を追加します。

```
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'
]
```

最後に、次の例では、プロトコルが ではないことを確認します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' ]
]
```

完全なルールは次のとおりです。

```
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
                    >>
                }
            }

        }
    }
}
```

### 含まれるタイプに基づいてコレクションを分離する

Infrastructure as Code (IaC) 設定テンプレートを使用する場合、設定テンプレート内の他のエンティティへの参照を含むコレクションが表示されることがあります。以下は、 へのローカル参照、 への参照`TaskRoleArn`、`TaskArn`および直接文字列参照を使用して Amazon Elastic Container Service (Amazon ECS) タスクを記述する CloudFormation テンプレートの例です。

```
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'
```

次のクエリについて考えます。

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

このクエリは、サンプルテンプレートに表示される 3 つのAWS::ECS::TaskDefinitionリソースすべてを含む値のコレクションを返します。次の例に示すように、TaskRoleArnローカル参照ecs_tasksを含む を他の から分離します。

```
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
    >>
}
```

# ガードルールでの変数の割り当てと参照

AWS CloudFormation Guard ルールファイルに変数を割り当てて、Guard ルールで参照する情報を保存できます。Guard はワンショット変数割り当てをサポートしています。変数は遅延的に評価されます。つまり、Guard はルールの実行時にのみ変数を評価します。

トピック


- 変数の割り当て
- 変数の参照
- 変数のスコープ
- ガードルールファイルの変数の例


## 変数の割り当て

`let` キーワードを使用して変数を初期化して割り当てます。ベストプラクティスとして、変数名にはスネークケースを使用します。変数は、クエリの結果として生じる静的リテラルまたは動的プロパティを保存できます。次の例では、 変数は静的文字列値`ecs_task_definition_task_role_arn`を保存します`arn:aws:iam:123456789012:role/my-role-name`。

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

次の例では、 変数は、 テンプレート内のすべての`AWS::ECS::TaskDefinition` CloudFormation リソースを検索するクエリの結果`ecs_tasks`を保存します。ルールを記述するときに、`ecs_tasks`を参照してこれらのリソースに関する情報にアクセスできます。

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

## 変数の参照

`%` プレフィックスを使用して変数を参照します。

の`ecs_task_definition_task_role_arn`変数の例に基づいて変数の割り当て、Guard ルール句の `query|value literal`セクションで`ecs_task_definition_task_role_arn`

を参照できます。このリファレンスを使用すると、CloudFormation テンプレート内のAWS::ECS::TaskDefinitionリソースの TaskDefinitionArnプロパティに指定された値が静的文字列値 になりますarn:aws:iam:123456789012:role/my-role-name。

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

のecs_tasks変数の例に基づいて変数の割り当て、クエリecs_tasksで を参照できます (%ecs_tasks.Properties など )。まず、Guard は変数を評価しecs_tasks、返された値を使用して階層を横断します。変数が文字列以外の値にecs_tasks解決された場合、Guard はエラーをスローします。

> **Note**
>
> 現在、Guard はカスタムエラーメッセージ内の変数の参照をサポートしていません。

## 変数のスコープ

スコープとは、ルールファイルで定義された変数の可視性を指します。変数名は、スコープ内で 1 回のみ使用できます。変数を宣言できるレベルは 3 つ、または可能な変数スコープは 3 つあります。

- ファイルレベル – 通常、ルールファイルの上部で宣言され、ルールファイル内のすべてのルールでファイルレベルの変数を使用できます。ファイル全体に表示されます。

  次のルールファイルの例では、変数 ecs_task_definition_task_role_arnと ecs_task_definition_execution_role_arnはファイルレベルで初期化されます。

  ```
  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
}
```

- ルールレベル – ルール内で宣言されたルールレベルの変数は、その特定のルールにのみ表示されます。ルール外の参照はエラーになります。

  次のルールファイルの例では、変数 ecs_task_definition_task_role_arnと ecs_task_definition_execution_role_arnはルールレベルで初期化されます。は、check_ecs_task_definition_task_role_arn名前付きルール内でのみ参照ecs_task_definition_task_role_arnできます。ecs_task_definition_execution_role_arn 変数は、check_ecs_task_definition_execution_role_arn名前付きルール内でのみ参照できます。

```
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
}
```

- ブロックレベル – when句などのブロック内で宣言されたブロックレベルの変数は、その特定のブロックにのみ表示されます。ブロック外の参照はエラーになります。

  次のルールファイルの例では、変数 ecs_task_definition_task_role_arnと ecs_task_definition_execution_role_arnはAWS::ECS::TaskDefinitionタイプブロック内のブロックレベルで初期化されます。ecs_task_definition_task_role_arn および ecs_task_definition_execution_role_arn変数は、それぞれのルールのAWS::ECS::TaskDefinition型ブロック内でのみ参照できます。

```
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
    }
}
```

## ガードルールファイルの変数の例

以下のセクションでは、変数の静的割り当てと動的割り当ての両方の例を示します。

静的割り当て

CloudFormation テンプレートの例を次に示します。

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

このテンプレートに基づいて、すべてのテンプレートAWS::ECS::TaskDefinitionリソースの TaskRoleArnプロパティcheck_ecs_task_definition_task_role_arnが であることを保証する というルールを記述できます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
```

```
}
```

ルールの範囲内で、 という変数を初期化ecs_task_definition_task_role_arnし、静的文字列値 を割り当てることができます'arn:aws:iam::123456789012:role/my-role-name'。ルール句は、 query|value literalセクションの ecs_task_definition_task_role_arn変数を参照arn:aws:iam::123456789012:role/my-role-nameすることで、EcsTaskリソースのTaskRoleArnプロパティに指定された値が であるかどうかをチェックします。

動的割り当て

CloudFormation テンプレートの例を次に示します。

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

このテンプレートに基づいて、 ファイルecs_tasksの範囲内で という変数を初期化し、クエリ を割り当てることができますResources.*[ Type == 'AWS::ECS::TaskDefinition'。Guard は入力テンプレート内のすべてのリソースをクエリし、それらの情報を に保存しますecs_tasks。すべてのAWS::ECS::TaskDefinitionテンプレートリソースの TaskRoleArnプロパティcheck_ecs_task_definition_task_role_arnが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'
}
```

ルール句は、 queryセクションの ecs_task_definition_task_role_arn変数を参照arn:aws:iam::123456789012:role/my-role-nameすることで、EcsTaskリソースのTaskRoleArnプロパティに指定された値が であるかどうかをチェックします。

CloudFormation テンプレート設定の適用

本番稼働用ユースケースのより複雑な例を見てみましょう。この例では、Amazon ECS タスクの定義方法をより厳密に制御するための Guard ルールを記述します。

CloudFormation テンプレートの例を次に示します。

```
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'
```

このテンプレートに基づいて、これらの要件が満たされるように次のルールを記述します。

- テンプレート内の各AWS::ECS::TaskDefinitionリソースには、タスクロールと実行ロールの両方がアタッチされています。
- タスクロールと実行ロールは AWS Identity and Access Management (IAM) ロールです。
- ロールは テンプレートで定義されます。
- PermissionsBoundary プロパティはロールごとに指定されます。

```
# 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
    }
}
```

# で名前付きルールブロックを作成する AWS CloudFormation Guard

を使用して名前付きルールブロックを記述する場合 AWS CloudFormation Guard、次の 2 つの構成スタイルを使用できます。

- 条件依存関係
- 相関依存関係

これらの依存関係構成のいずれかのスタイルを使用すると、再利用性が向上し、名前付きルールブロックの冗長性と繰り返しが軽減されます。

トピック


- 前提条件
- 条件依存関係の構成
- 相関依存関係の構成


## 前提条件

ルールの記述で名前付きルールブロックについて説明します。

## 条件依存関係の構成

この構成スタイルでは、whenブロックまたは名前付きルールブロックの評価は、1 つ以上の他の名前付きルールブロックまたは句の評価結果に条件付き依存します。次の Guard ルールファイルの例には、条件依存関係を示す名前付きルールブロックが含まれています。

```
# 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
    ...
}
```

前述のルールファイルの例では、 Example-1には次のような結果があります。

- が にrule_name_A評価されるとPASS、 によってカプセル化された Guard ルールrule_name_Bが評価されます。
- が にrule_name_A評価される場合FAIL、 によってカプセル化された Guard ルールrule_name_Bは評価されません。 は にrule_name_B評価されますSKIP。
- が にrule_name_A評価される場合SKIP、 によってカプセル化された Guard ルールrule_name_Bは評価されません。 は にrule_name_B評価されますSKIP。

> **Note**
>
> このケースは、 が と評価FAILされ、 が とrule_name_A評価されるルール にrule_name_A条件付きで依存している場合に発生しますSKIP。

以下は、イングレスおよびエグレスセキュリティグループ情報の項目からの設定管理データベース (CMDB) 設定 AWS Config 項目の例です。この例では、条件依存関係の構成を示します。

```
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
```

```
                }
            }
        }
    }
}
```

前述の例では、 check_parameter_validityは条件付きで に依存しcheck_resource_type_and_parameter、check_ip_procotol_and_port_range_validityは条件付きで に依存しますcheck_parameter_validity。以下は、前述のルールに準拠した設定管理データベース (CMDB) 設定項目です。

```
---
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
```

## 相関依存関係の構成

この構成スタイルでは、whenブロックまたは名前付きルールブロックの評価は、1 つ以上の他の Guard ルールの評価結果に相関依存します。相関依存関係は次のように達成できます。

```
# 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
    ...
}
```

相関依存関係の構成を理解するには、次の Guard ルールファイルの例を確認してください。

```
#
# 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'
}
```

前述のルールファイルでは、 ensure_elbs_are_internal_and_secureには との相関依存関係がありますensure_all_elbs_are_secure。以下は、前述のルールに準拠する CloudFormation テンプレートの例です。

```
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...'
```

## コンテキスト対応評価を実行するための句の記述

AWS CloudFormation Guard 句は階層データに対して評価されます。ガード評価エンジンは、シンプルな点線表記を使用して、指定された階層データに従って受信データに対するクエリを解決します。多くの場合、データのマップまたはコレクションに対して評価するには、複数の句が必要です。Guard は、このような句を記述するのに便利な構文を提供します。エンジンはコンテキストを認識し、評価に関連する対応するデータを使用します。

以下は、コンテキスト対応評価を適用できるコンテナを使用した Kubernetes Pod 設定の例です。

```
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
```

Guard 句を作成して、このデータを評価できます。ルールファイルを評価する場合、コンテキストは入力ドキュメント全体です。以下は、Pod で指定されたコンテナの制限の適用を検証する句の例です。

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

## 評価contextにおける理解

ルールブロックレベルでは、受信コンテキストは完全なドキュメントです。when 条件の評価は、 属性apiVersionと kind 属性があるこの受信ルートコンテキストに対して行われます。前の例では、これらの条件は に評価されますtrue。

ここで、前の例spec.containers[*]に示す の階層をトラバースします。階層のトラバースごとに、コンテキスト値はそれに応じて変わります。spec ブロックのトラバーサルが完了すると、次の例に示すようにコンテキストが変更されます。

```
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
```

containers 属性をトラバースした後、コンテキストを次の例に示します。

```
- 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
```

## ループについて

式を使用して[*]、 containers 属性の配列に含まれるすべての値のループを定義できます。ブロックは、 内の各要素について評価されますcontainers。前述のルールス二ペットの例では、 ブロックに含まれる句は、コンテナ定義に対して検証されるチェックを定義します。内部に含まれる句のブロックは、コンテナ定義ごとに 1 回、2 回評価されます。

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

反復ごとに、コンテキスト値は対応するインデックスの値です。

> **Note**
>
> サポートされているインデックスアクセス形式は、 [<integer>]または のみです[*]。現在、Guard は のような範囲をサポートしていません[2..4]。

## 配列

多くの場合、配列が受け入れられる場所では、単一の値も受け入れられます。たとえば、コンテナが 1 つしかない場合、配列を削除して次の入力を受け入れることができます。

```
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
```

属性が配列を受け入れることができる場合は、ルールが配列形式を使用していることを確認します。前の例では、 containers[*]ではなく を使用しますcontainers。Guard は、単一値フォームのみを検出したときにデータをトラバースするときに正しく評価します。

> **Note**
>
> 属性が配列を受け入れるときにルール句へのアクセスを表現するときは、常に配列形式を使用してください。単一の値を使用する場合でも、ガードは正しく評価されます。

## spec.containers[*] の代わりに フォームを使用する spec.containers

ガードクエリは解決された値のコレクションを返します。フォーム を使用する場合spec.containers、クエリの解決された値には、クエリ内の要素ではなくcontainers、 によって参照される配列が含まれます。フォームを使用する場合はspec.containers[*]、含まれる個々の要素を参照します。配列に含まれる各要素を評価する場合は、必ず [*]フォームを使用してください。

## this を使用して現在のコンテキスト値を参照する

ガードルールを作成するときは、 を使用してコンテキスト値を参照できますthis。多くの場合、 thisはコンテキストの値にバインドされているため、暗黙的です。たとえば、、this.apiVersionthis.kind、 this.specはルートまたはドキュメントにバインドされます。対照的に、 this.resourcesは /spec/containers/0/や などcontainers、の各値にバインドされます/spec/containers/1。同様に、 this.cpuと は制限、特に /spec/containers/0/resources/limitsと にthis.memoryマッピングされます/spec/containers/1/resources/limits。

次の例では、Kubernetes Pod 設定の前述のルールがthis明示的に を使用するように書き換えられています。

```
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
            >>
        }
    }
}
```

をthis明示的に使用する必要はありません。ただし、次の例に示すように、このthisリファレンスはスカラーを使用する場合に便利です。

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

前の例では、 this は各ポート番号を参照するために使用されます。

## 暗黙的な の使用による潜在的なエラー this

ルールと句を作成する場合、暗黙的なthisコンテキスト値から要素を参照するときによくある間違いがあります。たとえば、評価する次の入力データムを考えてみます (合格する必要があります ) 。

```
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
```

上記のテンプレートに対してテストすると、次のルールは暗黙的な を活用することを誤って想定するため、エラーが発生します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
                >>
            }
        }
    }
}
```

この例を実行するには、前述のルールファイルを という名前で保存any_ip_ingress_check.guardし、データを というファイル名で保存しますip_ingress.yaml。次に、これらのファイルを使用して次のvalidateコマンドを実行します。

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

次の出力では、エンジンは値 InputParameters.TcpBlockedPorts[*]のプロパティを取得しようとする試みが/configuration/ipPermissions/0/configuration/ipPermissions/1失敗したことを示します。

```
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[*]
```

この結果を理解するには、this明示的に参照された を使用してルールを書き換えます。

```
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 は、変数 に含まれる各値を参照しますany_ip_permissions。変数に割り当てられたクエリは、一致するconfiguration.ipPermissions値を選択します。エラーは、このコンテキストInputParamatersで を取得しようとInputParametersしたが、ルートコンテキストにあったことを示します。

内部ブロックは、次の例に示すように、範囲外の変数も参照します。

```
{
    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 は の各ポート値を参照しますが[21, 22, 110]、 fromPortおよび も参照しますtoPort。どちらも外部ブロックスコープに属します。

### の暗黙的な使用によるエラーの解決 **this**

変数を使用して、値を明示的に割り当て、参照します。まず、InputParameter.TcpBlockedPortsは入力 (ルート) コンテキストの一部です。次の例に示すように、内部ブロックInputParameter.TcpBlockedPortsの外に移動し、明示的に割り当てます。

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

次に、この変数を明示的に参照します。

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

内の内部this参照に対しても同じ操作を行います%ports。

ただし、内部のループの参照portsが正しくないため、すべてのエラーはまだ修正されていません。次の例は、誤った参照の削除を示しています。

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

次に、 validate コマンドを再度実行します。今回は合格です。

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

以下は、 validate コマンドの出力です。

```
ip_ingress.yaml Status = PASS
```

```
PASS rules
check_ip_procotol_and_port_range_validity    PASS
```

このアプローチに障害がないかテストするために、次の例ではペイロードの変更を使用します。

```
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 は、任意の IPv6 アドレスが許可されている 89～109 の範囲内です。コマンドを再度実行validateした後の出力を次に示します。

```
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 ルールのテスト

組み込みの AWS CloudFormation Guard ユニットテストフレームワークを使用して、Guard ルールが意図したとおりに動作することを確認できます。このセクションでは、ユニットテストファイルを作成する方法と、それを使用して `test` コマンドでルールファイルをテストする方法について説明します。

ユニットテストファイルには、、`.json`、`.JSON`、、`.jsn.yaml.YAML`、または のいずれかの拡張子が必要です`.yml`。

トピック

- 前提条件
- ガードユニットテストファイルの概要
- ガードルールユニットテストファイルの記述のチュートリアル

## 前提条件

入力データを評価するための Guard ルールを記述します。詳細については、「ガードルールの記述」を参照してください。

## ガードユニットテストファイルの概要

ガードユニットテストファイルは、複数の入力と、ガードルールファイル内に書き込まれたルールの期待される結果を含む JSON 形式または YAML 形式のファイルです。さまざまな期待を評価するためのサンプルが複数ある場合があります。まず空の入力をテストしてから、さまざまなルールと句を評価するための情報を段階的に追加することをお勧めします。

また、サフィックス`_test.json`または を使用してユニットテストファイルに名前を付けることをお勧めします`_tests.yaml`。たとえば、 という名前のルールファイルがある場合は`my_rules.guard`、ユニットテストファイル に名前を付けます`my_rules_tests.yaml`。

### 構文

以下は、YAML 形式のユニットテストファイルの構文を示しています。

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

## プロパティ

以下は、Guard テストファイルのプロパティです。

`input`

ルールをテストするデータ。次の例に示すように、最初のテストでは空の入力を使用することをお勧めします。

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

後続のテストでは、テストする入力データを追加します。

必須: はい

`expectations`

特定のルールが入力データに対して評価された場合に期待される結果。各ルールで期待される結果に加えて、テストする 1 つ以上のルールを指定します。期待される結果は、次のいずれかである必要があります。

- PASS – 入力データに対して実行すると、ルールは に評価されます`true`。
- FAIL – 入力データに対して実行すると、ルールは に評価されます`false`。
- SKIP – 入力データに対して実行すると、ルールはトリガーされません。

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

必須: はい

# ガードルールユニットテストファイルの記述のチュートリアル

以下は、 という名前のルールファイルですapi_gateway_private.guard。このルールの目的は、CloudFormation テンプレートで定義されたすべての Amazon API Gateway リソースタイプがプライベートアクセス専用にデプロイされているかどうかを確認することです。また、少なくとも 1 つのポリシーステートメントが Virtual Private Cloud (VPC) からのアクセスを許可するかどうかも確認します。

```
#
# 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
            }
        }
    }
}
```

このチュートリアルでは、最初のルールインテントをテストします。デプロイされたすべてのAWS::ApiGateway::RestApiリソースはプライベートである必要があります。

1. 次の初期テストapi_gateway_private_tests.yamlを含む というユニットテストファイルを作成します。最初のテストでは、空の入力を追加し、入力としてAWS::ApiGateway::RestApiリソースがないためルールがスキップcheck_rest_api_is_privateされることを期待します。

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

2. test コマンドを使用して、ターミナルで最初のテストを実行します。--rules-file パラメータには、ルールファイルを指定します。--test-data パラメータには、ユニットテストファイルを指定します。

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

   最初のテストの結果は ですPASS。

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

3. ユニットテストファイルに別のテストを追加します。次に、テストを拡張して空のリソースを含めます。更新されたapi_gateway_private_tests.yamlファイルは次のとおりです。

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

4. 更新されたユニットテストファイルtestを使用して を実行します。

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

2 番目のテストの結果は です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. ユニットテストファイルにさらに 2 つのテストを追加します。テストを拡張して、以下を含めます。

   - プロパティが指定されていないAWS::ApiGateway::RestApiリソース。

     > **Note**
     >
     > これは有効な CloudFormation テンプレートではありませんが、不正な形式の入力であってもルールが正しく機能するかどうかをテストすると便利です。

EndpointConfiguration プロパティが指定されていないため、 に設定されていないため、このテストは失敗することが予想されますPRIVATE。

- EndpointConfiguration プロパティを に設定して最初のインテントを満たすPRIVATEが、ポリシーステートメントが定義されていないため、2 番目のインテントを満たさない AWS::ApiGateway::RestApiリソース。このテストは合格すると予想されます。

更新されたユニットテストファイルを次に示します。

```
---
- 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. 更新されたユニットテストファイルtestを使用して を実行します。

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

3 番目の結果は でFAIL、4 番目の結果は です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. ユニットテストファイルでテスト 1~3 をコメントアウトします。4 番目のテストのみの詳細コンテキストにアクセスします。更新されたユニットテストファイルを次に示します。

```
---
#- 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. 評価結果を確認するには、 --verboseフラグを使用してターミナルで test コマンドを実行します。詳細なコンテキストは、評価を理解するのに役立ちます。この場合、4 番目のテストがPASS結果で成功した理由に関する詳細情報を提供します。

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

以下は、その実行からの出力です。

```
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)
```

出力からのキー観測値はClause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)、チェックに合格したことを示す行 です。この例では、 Typesが配列であると予想されるが、単一の値が指定されたケースも示しました。この場合、Guard は引き続き評価を行い、正しい結果を提供しました。

9. 4 番目のテストケースのようなテストケースを、 EndpointConfigurationプロパティが指定された AWS::ApiGateway::RestApiリソースのユニットテストファイルに追加します。テストケースは合格ではなく失敗します。更新されたユニットテストファイルを次に示します。

```
---
#- 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. --verbose フラグを使用して、更新されたユニットテストファイルで test コマンドを実行します。

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

は に指定されているEndpointConfigurationが、 は想定されていないためREGIONAL、結果は想定FAILどおりです。

```
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)
```

test コマンドの詳細出力は、ルールファイルの構造に従います。ルールファイル内のすべてのブロックは、詳細出力のブロックです。一番上のブロックは各ルールです。ルールにwhen条件がある場合は、兄弟条件ブロックに表示されます。次の例では、 条件%api_gws !emptyがテストされ、合格します。

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

条件が成功したら、ルール句をテストします。

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

`%api_gws` は、出力`BlockClause`のレベル (line:21) に対応するブロックルールです。ルール句は、連結 (AND) 句のセットであり、各連結句は一連の差分 (ORs) です。連結には 1 つの句 がありますProperties.EndpointConfiguration.Types[*] == "PRIVATE"。したがって、詳細出力には 1 つの句が表示されます。パスは、入力内のどの値が比較されるか/Resources/apiGw/Properties/EndpointConfiguration/Types/1を示します。この場合、 は 1 でTypesインデックス付けされた の 要素です。

ではGuard ルールに対する入力データの検証、このセクションの例を使用して、 `validate` コマンドを使用して入力データをルールに対して評価できます。

# AWS CloudFormation Guard ルールでの入力パラメータの使用

AWS CloudFormation Guard では、検証中に動的データルックアップの入力パラメータを使用できます。この機能は、ルールで外部データを参照する必要がある場合に特に便利です。ただし、入力パラメータキーを指定する場合、Guard は競合するパスがないことを要求します。

## の使用方法

1. `--input-parameters` または `-i`フラグを使用して、入力パラメータを含むファイルを指定します。複数の入力パラメータファイルを指定でき、組み合わせて共通のコンテキストを形成します。入力パラメータキーに競合するパスを含めることはできません。
2. `--data` または `-d`フラグを使用して、検証する実際のテンプレートファイルを指定します。

## 使用例

1. 入力パラメータファイルを作成します (例: `network.yaml`)。

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

2. ガードルールファイルでこれらのパラメータを参照します (例: `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. 失敗したデータテンプレートを作成します (例: `security_groups_fail.yaml`)。

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

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

4. validate コマンドを実行します。

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

このコマンドで:

- `-r` はルールファイルを指定します。
- `-i` は入力パラメータファイルを指定します。
- `-d` は、検証するデータファイル (テンプレート) を指定します。

### 複数の入力パラメータ

複数の入力パラメータファイルを指定できます。

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

で指定されたすべてのファイルは結合され、パラメータルックアップの単一のコンテキストを形成し-iます。

## AWS CloudFormation Guard ルールに対する入力データの検証

コマンドを使用して AWS CloudFormation Guard validate、Guard ルールに対してデータを検証できます。コマンドのパラメータやオプションなどの詳細についてはvalidate、「検証」を参照してください。

### 前提条件

- 入力データを検証する Guard ルールを記述します。詳細については、「ガードルールの記述」を参照してください。
- ルールをテストして、意図したとおりに動作することを確認します。詳細については、「ガードルールのテスト」を参照してください。

### validate コマンドの使用

AWS CloudFormation テンプレートなどの Guard ルールに対して入力データを検証するには、Guard validate コマンドを実行します。--rules パラメータには、ルールファイルの名前を指定します。--data パラメータには、入力データファイルの名前を指定します。

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

Guard がテンプレートを正常に検証した場合、validateコマンドは終了ステータス 0 ($?bash 単位) を返します。Guard がルール違反を特定すると、validateコマンドは失敗したルールのステータスレポートを返します。概要フラグ (-s all) を使用して、Guard が各ルールをどのように評価したかを示す詳細な評価ツリーを表示します。

```
template.json Status = FAIL
SKIP rules
```

```
rules.guard/aws_apigateway_deployment_checks    SKIP
rules.guard/aws_apigateway_stage_checks         SKIP
rules.guard/aws_dynamodb_table_checks           SKIP
PASS rules
rules.guard/aws_events_rule_checks              PASS
rules.guard/aws_iam_role_checks                 PASS
FAILED rules
rules.guard/aws_ec2_volume_checks               FAIL
rules.guard/mixed_types_checks                  FAIL
---
Evaluation of rules rules.guard against data template.json
--
Property [/Resources/vol2/Properties/Encrypted] in data [template.json] is not
 compliant with [rules.guard/aws_ec2_volume_checks] because provided value [false] did
 not match expected value [true]. Error Message []
Property traversed until [/Resources/vol2/Properties] in data [template.json] is not
 compliant with [rules.guard/aws_ec2_volume_checks] due to retrieval error. Error
 Message [Attempting to retrieve array index or key from map at path = /Resources/vol2/
Properties , Type was not an array/object map, Remaining Query = Size]
Property [/Resources/vol2/Properties/Encrypted] in data [template.json] is not
 compliant with [rules.guard/mixed_types_checks] because provided value [false] did not
 match expected value [true]. Error Message []
--
Rule [rules.guard/aws_iam_role_checks] is compliant for data [template.json]
Rule [rules.guard/aws_events_rule_checks] is compliant for data [template.json]
--
Rule [rules.guard/aws_apigateway_deployment_checks] is not applicable for data
 [template.json]
Rule [rules.guard/aws_apigateway_stage_checks] is not applicable for data
 [template.json]
Rule [rules.guard/aws_dynamodb_table_checks] is not applicable for data [template.json]
```

## 複数のデータファイルに対する複数のルールの検証

ルールを維持するために、ルールを複数のファイルに書き込み、必要に応じてルールを整理できます。次に、データファイルまたは複数のデータファイルに対して複数のルールファイルを検証できます。validate コマンドは、 --dataおよび --rulesオプションのファイルのディレクトリを取得できます。たとえば、 に 1 つ以上のデータファイル/path/to/dataDirectoryが含まれ、 に 1 つ以上のルールファイル/path/to/ruleDirectoryが含まれている場合、次のコマンドを実行できます。

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

複数の CloudFormation テンプレートで定義されたさまざまなリソースに、保管時の暗号化を保証する適切なプロパティ割り当てがあるかどうかをチェックするルールを作成できます。検索とメンテナンスを容易にするために、、、および という個別のファイル内の各リソースの保管時の暗号化をs3_bucket_encryption.guardec2_volume_encryption.guard、パス を持つディレクトリrds_dbinstance_encrytion.guardでチェックするルールを設定できます~/GuardRules/encryption_at_rest。検証する必要がある CloudFormation テンプレートは、パス を持つディレクトリにあります~/CloudFormation/templates。この場合、次のように validate コマンドを実行します。

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

# トラブルシューティング AWS CloudFormation Guard

の使用中に問題が発生した場合は AWS CloudFormation Guard、このセクションのトピックを参照してください。

トピック


- 選択したタイプのリソースが存在しない場合、句は失敗します
- Guard は、短い形式の Fn::GetAtt リファレンスを使用して CloudFormation テンプレートを評価しません
- 一般的なトラブルシューティングのトピック


## 選択したタイプのリソースが存在しない場合、句は失敗します

クエリが のようなフィルターを使用する場合`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`、入力に`AWS::ApiGateway::RestApi`リソースがない場合、 句は に評価されます`FAIL`。

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

この結果を回避するには、変数にフィルタを割り当て、`when`条件チェックを使用します。

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

## Guard は、短い形式の Fn::GetAtt リファレンスを使用して CloudFormation テンプレートを評価しません

Guard は、短い形式の組み込み関数をサポートしていません。たとえば、YAML 形式の CloudFormation テンプレート`!Join``!Sub`での の使用はサポートされていません。代わりに、CloudFormation 組み込み関数の拡張形式を使用します。例えば、Guard ルールに照らして評価する場合は`Fn::Join`、YAML 形式の CloudFormation テンプレート`Fn::Sub`で を使用します。

組み込み関数の詳細については、 AWS CloudFormation ユーザーガイドの組み込み関数リファレンスを参照してください。

# 一般的なトラブルシューティングのトピック

- `string` リテラルにエスケープされた文字列が埋め込まれていないことを確認します。Guard はリ`string`テラルの埋め込みエスケープ文字列をサポートしていません。インライン JSON 文字列を解析する場合は、Guard 3.0.0 以降で利用可能な `json_parse()`関数を使用します。詳細については、「組み込み関数の使用」を参照してください。
- != 比較で互換性のあるデータ型が比較されていることを確認します。たとえば、 `string`と`int`は、比較用の互換性のないデータ型です。!= 比較を実行するときに、値が互換性がない場合、内部でエラーが発生します。現在、エラーは抑制され、Rust の PartialEq 特性を満たす`false`ために に変換されます。

# AWS CloudFormation Guard CLI パラメータとコマンドリファレンス

コマンドラインインターフェイス (CLI) では、次のグローバルパラメータと AWS CloudFormation Guard コマンドを使用できます。

トピック


- Guard CLI グローバルパラメータ
- 解析ツリー
- rulegen
- test
- validate


## Guard CLI グローバルパラメータ

次のパラメータは、任意の AWS CloudFormation Guard CLI コマンドで使用できます。

`-h, --help`

ヘルプ情報を表示します。

`-V, --version`

バージョン情報を表示します。

## 解析ツリー

AWS CloudFormation Guard ルールファイルで定義されたルールの解析ツリーを生成します。

### 構文

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

## パラメータ

`-h, --help`

ヘルプ情報を表示します。

`-p, --print-json`

出力を JSON 形式で出力します。

`-y, --print-yaml`

出力を YAML 形式で出力します。

`-V, --version`

バージョン情報を表示します。

## オプション

`-o, --output`

生成されたツリーを出力ファイルに書き込みます。

`-r, --rules`

ルールファイルを提供します。

## 例

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

# rulegen

JSON 形式または YAML 形式の AWS CloudFormation テンプレートファイルを取得し、テンプレートリソースのプロパティに一致する一連の AWS CloudFormation Guard ルールを自動生成します。このコマンドは、ルールの記述を開始したり、既知の優れたテンプレートからready-to-useルールを作成したりするのに役立ちます。

## 構文

```
cfn-guard rulegen
```

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

## パラメータ

`-h, --help`

ヘルプ情報を表示します。

`-V, --version`

バージョン情報を表示します。

## オプション

`-o, --output`

生成されたルールを出力ファイルに書き込みます。数百または数千のルールが発生する可能性があるため、このオプションを使用することをお勧めします。

`-t, --template`

CloudFormation テンプレートファイルへのパスを JSON または YAML 形式で提供します。

## 例

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

# test

AWS CloudFormation Guard ルールファイルを JSON または YAML 形式の Guard ユニットテストファイルと照合して検証し、個々のルールの成功を判断します。

## 構文

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

## パラメータ

`-a, --alphabetical`

ディレクトリ内でアルファベット順にソートします。

`-h, --help`

ヘルプ情報を表示します。

`-m, --last-modified`

ディレクトリ内の最終変更時刻でソートします。

`-V, --version`

バージョン情報を表示します。

`-v, --verbose`

出力の詳細度を高めます。複数回指定できます。

詳細な出力は、Guard ルールファイルの構造に従います。ルールファイル内のすべてのブロックは、詳細出力のブロックです。一番上のブロックは各ルールです。ルールに対してwhen条件がある場合、それらは兄弟条件ブロックとして表示されます。

## オプション

`-d, --dir`

ルールのルートディレクトリを指定します。

`-o, --output-format`

出力を表示する形式を指定します。

デフォルト: `single-line-summary`

使用できる値: `json` | `yaml` | `single-line-summary` | `junit`

`-r, --rules-file`

ルールファイルの名前を指定します。

`-t, --test-data`

JSON 形式または YAML 形式のデータファイルの名前またはディレクトリを指定します。

## 例

```
cfn-guard test --rules-file rules.guard --test-data example.json
```

## Output

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

## 関連情報

ガードルールのテスト

# validate

AWS CloudFormation Guard ルールに照らしてデータを検証し、成功または失敗を判断します。

## 構文

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

## パラメータ

`-a, --alphabetical`

アルファベット順に並べられたディレクトリ内のファイルを検証します。

`-h, --help`

ヘルプ情報を表示します。

-m, --last-modified

最終変更時刻順に並べられたディレクトリ内のファイルを検証します。

-P, --payload

経由で次の JSON 形式でルールとデータを指定しますstdin。

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

例えば、次のようになります。

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

「ルール」には、ルールファイルの文字列バージョンのリストを指定します。「data」には、データファイルの文字列バージョンのリストを指定します。

--payload が指定され--rulesていて、指定--dataできない場合。

-p, --print-json

出力を JSON 形式で出力します。

-s, --show-clause-failures

概要を含む句の失敗を表示します。

-V, --version

バージョン情報を表示します。

-v, --verbose

出力の詳細度を高めます。複数回指定できます。

`-z, --structured`

構造化された有効な JSON/YAML のリストを出力します。この引数は、verbose、print-json、show-summary: all/fail/pass/skip、output-format: single-line-summary の引数と競合します。

## オプション

`-d、 --data` (文字列)

JSON または YAML のデータファイルまたはデータファイルのディレクトリを提供します。このオプションを繰り返し使用して、複数の値を渡すことができます。

例: `--data template1.yaml --data ./data-dir1 --data template2.yaml`

`data-dir1` 上記のようなディレクトリ引数の場合、スキャンは拡張子
が .yaml、.yml、.json、.jsn、.template のファイルでのみサポートされます。

`--payload` フラグを指定する場合は、 `--data`オプションを指定しないでください。

`-i、 --input-parameters` (文字列)

JSON または YAML のパラメータファイルのパラメータファイルまたはディレクトリを提供し、結合コンテキストとして使用するデータファイルとともに使用する追加のパラメータを指定します。入力として渡されたすべてのパラメータファイルがマージされ、この結合コンテキストは再び の引数として渡された各ファイルにマージされますdata。このため、すべてのファイルには、重複することなく、相互に排他的なプロパティが含まれていることが期待されます。このオプションを繰り返し使用して、複数の値を渡すことができます。

ディレクトリ引数の場合、スキャンは拡張子が .yaml、.yml、.json、.jsn、.template のファイルでのみサポートされます。

`-o、 --output-format` (文字列)

出力の形式を指定します。

デフォルト: `single-line-summary`

許可できる値: `json` | `yaml` | `single-line-summary` | `junit` | `sarif`

`-r、 --rules` (文字列)

ルールファイルまたはルールファイルのディレクトリを提供します。このオプションを繰り返し使用して、複数の値を渡すことができます。

例: `--rules rule1.guard --rules ./rules-dir1 --rules rule2.guard`

`rules-dir1` 上記のようなディレクトリ引数の場合、スキャンは拡張子が .guard、.ruleset のファイルでのみサポートされます。

`--payload` フラグを指定する場合は、 `--rules`オプションを指定しないでください。

`--show-summary` (文字列)

概要テーブルを表示する必要があるかどうかを制御します。 `--show-summary fail` (デフォルト) または `--show-summary pass,fail` (合格/不合格となったルールのみを表示) または `--show-summary none` (オフにするには) または `--show-summary all` (合格、不合格、スキップしたすべてのルールを表示するために ) 。

デフォルト: `fail`

許可できる値: none | `all` | `pass` | `fail` | `skip`

`-t`、 `--type` (文字列)

入力データの形式を提供します。入力データ型を指定すると、Guard は CloudFormation テンプレートリソースの論理名を出力に表示します。デフォルトでは、Guard は などのプロパティパスと値を表示しますProperty [/Resources/vol2/Properties/Encrypted。

許可される値: `CFNTemplate`

## 例

```
cfn-guard validate --data example.json --rules rules.guard
```

## Output

Guard がテンプレートを正常に検証した場合、`validate`コマンドは終了ステータス `0` ($?bash 単位) を返します。Guard がルール違反を特定すると、`validate`コマンドは失敗したルールのステータスレポートを返します。

```
example.json Status = FAIL
FAILED rules
rules.guard/policy_effect_is_deny    FAIL
---
Evaluation of rules rules.guard against data example.json
```

```
--
Property [/path/to/Effect] in data [example.json] is not compliant with
 [policy_effect_is_deny] because provided value ["Allow"] did not match expected value
 ["Deny"]. Error Message [ Policy statement "Effect" must be "Deny".]
```

## 関連情報

- Guard ルールに対する入力データの検証
- Guard ルールでの入力パラメータの使用

# のセキュリティ AWS CloudFormation Guard

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS とお客様の間の責任共有です。責任共有モデルではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。Guard に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

以下のドキュメントは、Guard を AWS Lambda 関数 (cfn-guard-lambda) としてインストールするときに責任共有モデルを適用する方法を理解するのに役立ちます。cfn-guard-lambda

- AWS Command Line Interface ユーザーガイドのセキュリティ
- AWS Lambda デベロッパーガイドのセキュリティ
- AWS Identity and Access Management ユーザーガイドのセキュリティ

# AWS CloudFormation Guard ドキュメント履歴

次の表に、 のドキュメントリリースを示します AWS CloudFormation Guard。

- ドキュメントの最終更新日: 2025 年 7 月 30 日
- 最新バージョン: 3.1.2

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| ドキュメントの更新 | 現在の実装に合わせて Guard CLI コマンドリファレンスドキュメントを更新しました。バージョンリファレンスが Guard 3.1.2 に更新されました。 | 2025 年 7 月 30 日 |
| バージョン 3.0.0 リリース | バージョン 3.0.0 では、次の改善が導入されています。<br>• Guard 3.0.0 のリリース用に概要とインストールのトピックが更新されました。<br>• Homebrew および のインストール手順を追加しました Chocolatey。<br>• Guard バージョン 3.0.0 の変更を反映するように更新された Guard ルールの移行に関する情報。<br>• AWS CloudFormation Guard GitHub リポジトリへの目立つリンクを追加しました。 | 2023 年 6 月 30 日 |
| バージョン 2.1.3 リリース | バージョン 2.1.3 では、次の改善が導入されています。 | 2023 年 6 月 9 日 |

| | | |
|---|---|---|
| | Guard 2.1.3 の機能強化に関する情報が追加されました。Guard 2.0 のリファレンスが Guard 2.1.3 に更新されました。 | |
| バージョン 2.0.4 リリース | バージョン 2.0.4 では、次の改善が導入されています。<br><br>`--payload` フラグが `validate` コマンドに追加されました。<br><br>詳細については、「Guard CLI リファレンス」の「検証」を参照してください。 | 2021 年 10 月 19 日 |
| バージョン 2.0.3 リリース | バージョン 2.0.3 では、次の改善が導入されています。<br><br>• ユニットテストファイルでは、各テストのテスト名を指定できます。詳細については、「ガードルールのテスト」を参照してください。<br>• `validate` コマンドに次のオプションが追加されました。<br>  • `--output-format`<br>  • `--show-summary`<br>  • `--type`<br><br>詳細については、「Guard CLI リファレンス」の「検証」を参照してください。 | 2021 年 7 月 27 日 |

| | | |
|---|---|---|
| 初回リリース | AWS CloudFormation Guard ユーザーガイドの初回リリース。 | 2021 年 7 月 15 日 |

# AWS 用語集

最新の AWS 用語については、 AWS の用語集 リファレンスのAWS 用語集を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。