



使用者指南

# AWS CloudFormation Guard



# AWS CloudFormation Guard: 使用者指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

# Table of Contents

什麼是 AWS CloudFormation Guard ? .....	1
您是第一次使用 Guard 嗎? .....	1
Guard 功能 .....	2
搭配 CloudFormation 勾點使用 Guard .....	2
存取 Guard .....	2
最佳實務 .....	2
設定 Guard .....	3
針對 Linux 和 macOS .....	3
從預先建置的發行二進位檔安裝 Guard .....	3
從 Cargo 安裝 Guard .....	4
從 Homebrew 安裝 Guard .....	5
適用於 Windows .....	5
先決條件 .....	5
從 Cargo 安裝 Guard .....	4
從 Chocolatey 安裝 Guard .....	6
做為 AWS Lambda 函數 .....	7
先決條件 .....	7
安裝 Rust 套件管理員 .....	7
將 Guard 安裝為 Lambda 函數 .....	8
建置和執行 .....	9
呼叫 Lambda 函數 .....	9
使用 Guard 規則的先決條件和概觀 .....	10
先決條件 .....	10
使用 Guard 規則的概觀 .....	10
撰寫 Guard 規則 .....	10
子句 .....	11
在子句中使用查詢 .....	13
在子句中使用運算子 .....	13
在子句中使用自訂訊息 .....	17
合併子句 .....	17
搭配 Guard 規則使用區塊 .....	18
使用內建函數 .....	21
定義查詢和篩選 .....	22
在 Guard 規則中指派和參考變數 .....	35

編寫具名規則區塊 .....	41
撰寫子句以執行內容感知評估 .....	47
測試 Guard 規則 .....	60
先決條件 .....	60
概觀 .....	60
逐步解說 .....	61
搭配 Guard 規則使用輸入參數 .....	70
如何使用 .....	71
範例使用方式 .....	71
多個輸入參數 .....	72
根據 Guard 規則驗證輸入資料 .....	72
先決條件 .....	72
使用 validate 命令 .....	72
針對多個資料檔案驗證多個規則 .....	74
Guard 故障診斷 .....	75
當沒有所選類型的資源時，子句失敗 .....	75
Guard 不會評估 CloudFormation 範本 .....	75
一般疑難排解主題 .....	75
Guard CLI 參考 .....	77
Guard CLI 全域參數 .....	77
parse-tree .....	77
語法 .....	77
Parameters .....	77
選項 .....	78
範例 .....	78
rulegen .....	78
語法 .....	78
Parameters .....	79
選項 .....	79
範例 .....	79
test .....	79
語法 .....	79
Parameters .....	79
選項 .....	80
範例 .....	80
Output .....	81

---

另請參閱 .....	81
validate .....	81
語法 .....	81
Parameters .....	81
選項 .....	82
範例 .....	84
Output .....	84
另請參閱 .....	84
安全 .....	85
文件歷史紀錄 .....	86
AWS 詞彙表 .....	88
.....	lxxxix

# 什麼是 AWS CloudFormation Guard ?

AWS CloudFormation Guard 是一種開放原始碼的通用型政策即程式碼評估工具。Guard 命令列界面 (CLI) 提供 simple-to-use 宣告式網域特定語言 (DSL)，可用來將政策表達為程式碼。此外，您可以使用 CLI 命令，根據這些規則驗證結構化階層式 JSON 或 YAML 資料。Guard 也提供內建的單元測試架構，以驗證您的規則是否如預期般運作。

Guard 不會驗證 CloudFormation 範本的有效語法或允許的屬性值。您可以使用 [cfn-lint](#) 工具對範本結構執行徹底檢查。

Guard 不提供伺服器端強制執行。您可以使用 CloudFormation Hooks 來執行伺服器端驗證和強制執行，您可以在其中封鎖或警告操作。

如需 AWS CloudFormation Guard 開發的詳細資訊，請參閱 [Guard GitHub 儲存庫](#)。

## 主題

- [您是第一次使用 Guard 嗎？](#)
- [Guard 功能](#)
- [搭配 CloudFormation 勾點使用 Guard](#)
- [存取 Guard](#)
- [最佳實務](#)

## 您是第一次使用 Guard 嗎？

如果您是第一次使用 Guard，建議您先閱讀以下章節：

- [設定 Guard](#) – 本節說明如何安裝 Guard。透過 Guard，您可以使用 Guard DSL 撰寫政策規則，並根據這些規則驗證 JSON 或 YAML 格式的結構化資料。
- [撰寫 Guard 規則](#) – 本節提供撰寫政策規則的詳細演練。
- [測試 Guard 規則](#) – 本節提供詳細的逐步解說，用於測試您的規則以驗證它們是否如預期般運作，並根據規則驗證 JSON 或 YAML 格式的結構化資料。
- [根據 Guard 規則驗證輸入資料](#) – 本節提供詳細的逐步解說，讓您根據規則驗證 JSON 或 YAML 格式的結構化資料。
- [Guard CLI 參考](#) – 本節說明 Guard CLI 中可用的命令。

## Guard 功能

使用 Guard，您可以撰寫政策規則來驗證任何 JSON 或 YAML 格式的結構化資料，包括但不限於 CloudFormation 範本。Guard 支援政策檢查的完整 end-to-end 評估範圍。規則在下列商業網域中很有用：

- 預防性控管和合規（左移測試） – 根據代表組織安全與合規最佳實務的政策規則，驗證基礎設施為程式碼 (IaC) 或基礎設施和服務組合。例如，您可以驗證 CloudFormation 範本、CloudFormation 變更集、JSON 型 Terraform 組態檔案或 Kubernetes 組態。
- Detective 控管和合規 – 驗證組態管理資料庫 (CMDB) 資源的一致性，例如 AWS Config 以組態項目 (CIs)。例如，開發人員可以使用針對 AWS Config CIs Guard 政策來持續監控已部署 AWS 和非 AWS 資源的狀態、偵測政策違規，以及開始修復。
- 部署安全 – 確保變更在部署之前是安全的。例如，根據政策規則驗證 CloudFormation 變更集，以防止導致資源取代的變更，例如重新命名 Amazon DynamoDB 資料表。

## 搭配 CloudFormation 勾點使用 Guard

您可以使用 CloudFormation Guard 在 CloudFormation Hooks 中撰寫勾點。CloudFormation Hooks 可讓您在 CloudFormation 建立、更新或刪除操作，以及 AWS 雲端控制 API 建立或更新操作之前，主動強制執行您的 Guard 規則。勾點可確保您的資源組態符合組織的安全性、營運和成本最佳化最佳實務。

如需如何使用 Guard 撰寫 CloudFormation Guard Hooks 的詳細資訊，請參閱 CloudFormation 《Hooks 使用者指南》中的 [Write Guard 規則來評估 Guard Hooks 的資源](#)。

## 存取 Guard

若要存取 Guard DSL 和命令，您必須安裝 Guard CLI。如需安裝 Guard CLI 的詳細資訊，請參閱 [設定 Guard](#)。

## 最佳實務

撰寫簡單的規則，並使用具名規則來參考其他規則中的規則。複雜的規則可能難以維護和測試。

# 設定 AWS CloudFormation Guard

AWS CloudFormation Guard 是開放原始碼命令列界面 (CLI)。它為您提供簡單、特定網域的語言，以撰寫政策規則，並根據這些規則驗證其結構化階層式 JSON 和 YAML 資料。這些規則可以代表與安全性、合規性等相關的公司政策指導方針。結構化階層式資料可以代表描述為程式碼的雲端基礎設施。例如，您可以建立規則，以確保一律在其 CloudFormation 範本中建立加密的 Amazon Simple Storage Service (Amazon S3) 儲存貯體模型。

下列主題提供如何使用您選擇的作業系統或 AWS Lambda 函數安裝 Guard 的相關資訊。

## 主題

- [安裝 Guard for Linux 和 macOS](#)
- [安裝 Guard for Windows](#)
- [安裝 Guard 做為 AWS Lambda 函數](#)

## 安裝 Guard for Linux 和 macOS

您可以使用預先建置的發行二進位檔、Cargo 或透過 Homebrew，AWS CloudFormation Guard 為 Linux 和 macOS 安裝。

### 從預先建置的發行二進位檔安裝 Guard

使用下列程序從預先建置的二進位檔安裝 Guard。

1. 開啟終端機，並執行下列命令。

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

2. 執行下列命令來設定您的PATH變數。

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

結果：您已成功安裝 Guard 並設定PATH變數。

- (選用) 若要確認 Guard 的安裝，請執行下列命令。

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

命令會傳回下列輸出：

```
cfn-guard 3.1.2
```

## 從 Cargo 安裝 Guard

Cargo 是 Rust 套件管理員。完成下列步驟以安裝 Rust，其中包含 Cargo。然後，從 Cargo 安裝 Guard。

1. 從終端機執行下列命令，並依照畫面上的指示安裝 Rust。

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

- (選用) 對於 Ubuntu 環境，請執行下列命令。

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

2. 設定您的PATH環境變數，並執行下列命令。

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

3. 安裝 Cargo 後，請執行下列命令來安裝 Guard。

```
cargo install cfn-guard
```

結果：您已成功安裝 Guard。

- (選用) 若要確認 Guard 的安裝，請執行下列命令。

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

命令會傳回下列輸出：

```
cfn-guard 3.1.2
```

## 從 Homebrew 安裝 Guard

Homebrew 是 macOS 和 Linux 的套件管理員。完成下列步驟以安裝 Homebrew。然後，從 Homebrew 安裝 Guard。

1. 從終端機執行下列命令，並依照畫面上的指示安裝 Homebrew。

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

2. 安裝 Homebrew 後，請執行下列命令來安裝 Guard。

```
brew install cloudformation-guard
```

結果：您已成功安裝 Guard。

- (選用) 若要確認 Guard 的安裝，請執行下列命令。

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

命令會傳回下列輸出：

```
cfn-guard 3.1.2
```

## 安裝 Guard for Windows

您可以透過 Cargo 或 Chocolatey AWS CloudFormation Guard 為 Windows 安裝。

### 先決條件

若要從命令列界面建置 Guard，您必須安裝 Build Tools for Visual Studio 2019。

1. 從[適用於 Visual Studio 的建置工具 2019 網站](#)下載 [Microsoft Visual C++ 建置工具](#)。
2. 執行安裝程式，然後選取預設值。

## 從 Cargo 安裝 Guard

Cargo 是 Rust 套件管理員。完成下列步驟以安裝 Rust，其中包含 Cargo。然後，從 Cargo 安裝 Guard。

1. [下載 Rust](#)，然後執行 `rustup-init.exe`。
2. 從命令提示字元中，選擇 1，這是預設選項。

命令會傳回下列輸出：

```
Rust is installed now. Great!
```

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

```
Press the Enter key to continue.
```

3. 若要完成安裝，請按 Enter 鍵。
4. 安裝 Cargo 後，請執行下列命令來安裝 Guard。

```
cargo install cfn-guard
```

結果：您已成功安裝 Guard。

- (選用) 若要確認 Guard 的安裝，請執行下列命令。

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

命令會傳回下列輸出：

```
cfn-guard 3.1.2
```

## 從 Chocolatey 安裝 Guard

Chocolatey 是 Windows 的套件管理員。完成下列步驟以安裝 Chocolatey。然後，從 Chocolatey 安裝 Guard。

1. 遵循本指南安裝 [Chocolatey](#)
2. 安裝 Chocolatey 後，請執行下列命令來安裝 Guard。

```
choco install cloudformation-guard
```

結果：您已成功安裝 Guard。

- (選用) 若要確認 Guard 的安裝，請執行下列命令。

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

命令會傳回下列輸出：

```
cfn-guard 3.1.2
```

## 安裝 Guard 做為 AWS Lambda 函數

您可以透過 Rust 套件管理員 AWS CloudFormation Guard Cargo 安裝。Guard 作為 AWS Lambda 函數 (cfn-guard-lambda) 是 Guard (cfn-guard) 周圍的輕量包裝函式，可用作 Lambda 函數。

### 先決條件

您必須先滿足下列先決條件，才能將 Guard 安裝為 Lambda 函數：

- AWS Command Line Interface (AWS CLI) 已設定可部署和叫用 Lambda 函數的許可。如需詳細資訊，請參閱[設定 AWS CLI](#)。
- AWS Identity and Access Management (IAM) 中的 AWS Lambda 執行角色。如需詳細資訊，請參閱[AWS Lambda 執行角色](#)。
- 在 CentOS/RHEL 環境中，將 musl-libc 套件儲存庫新增至您的 yum 組態。如需詳細資訊，請參閱 [ngompa/musl-libc](#)。

### 安裝 Rust 套件管理員

Cargo 是 Rust 套件管理員。完成下列步驟以安裝 Rust，其中包含 Cargo。

1. 從終端機執行下列命令，然後依照畫面上的指示安裝 Rust。

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

- (選用) 對於 Ubuntu 環境，請執行下列命令。

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

2. 設定您的PATH環境變數，並執行下列命令。

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

## 安裝 Guard 做為 Lambda 函數 (Linux、macOS 或 Unix)

若要將 Guard 安裝為 Lambda 函數，請完成下列步驟。

1. 從您的命令終端機執行下列命令。

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

- (選用) 若要確認將 Guard 安裝為 Lambda 函數，請執行下列命令。

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

命令會傳回下列輸出：

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

2. 若要安裝musl支援，請執行下列命令。

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

3. 使用 建置 musl，然後在終端機中執行下列命令。

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

對於[自訂執行期](#)，AWS Lambda 需要部署套件 bootstrap.zip 檔案中名稱為的可執行檔。將產生的可執行檔重新命名為 cfn-lambda，bootstrap然後將其新增至 .zip 封存檔。

- 對於 macOS 環境，請在 Rust 專案的根目錄或 中建立您的貨物組態檔案 `~/.cargo/config`。

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

- 變更為 `cfn-guard-lambda` 根目錄。

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

- 在終端機中執行下列命令。

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

- 執行下列命令，以 Lambda 函數 `cfn-guard` 的形式提交至您的帳戶。

```
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
```

## 建置和執行 Guard 做為 Lambda 函數

若要叫用以 Lambda 函數 `cfn-guard-lambda` 提交的，請執行下列命令。

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

## 呼叫 Lambda 函數請求結構

請求 `cfn-guard-lambda` 要求以下欄位：

- `data` – YAML 或 JSON 範本的字串版本
- `rules` – 規則集檔案的字串版本

# 使用 Guard 規則的先決條件和概觀

本節示範如何完成針對 JSON 或 YAML 格式資料撰寫、測試和驗證規則的核心 Guard 任務。此外，它還包含詳細的逐步解說，示範可回應特定使用案例的撰寫規則。

## 主題

- [先決條件](#)
- [使用 Guard 規則的概觀](#)
- [撰寫 AWS CloudFormation Guard 規則](#)
- [測試 AWS CloudFormation Guard 規則](#)
- [搭配 AWS CloudFormation Guard 規則使用輸入參數](#)
- [根據 AWS CloudFormation Guard 規則驗證輸入資料](#)

## 先決條件

您必須先安裝 Guard 命令列界面 (CLI)，才能使用 Guard 網域特定語言 (DSL) 撰寫政策規則。如需詳細資訊，請參閱[設定 Guard](#)。

## 使用 Guard 規則的概觀

使用 Guard 時，您通常會執行下列步驟：

1. 撰寫 JSON 或 YAML 格式的資料進行驗證。
2. 寫入 Guard 政策規則。如需詳細資訊，請參閱[撰寫 Guard 規則](#)。
3. 使用 Guard test 命令確認您的規則如預期般運作。如需單元測試的詳細資訊，請參閱[測試 Guard 規則](#)。
4. 使用 Guard validate 命令，根據規則驗證 JSON 或 YAML 格式的資料。如需詳細資訊，請參閱[根據 Guard 規則驗證輸入資料](#)。

## 撰寫 AWS CloudFormation Guard 規則

在中 AWS CloudFormation Guard，規則是 policy-as-code 規則。您可以使用 Guard 網域特定語言 (DSL) 撰寫規則，以驗證 JSON 或 YAML 格式的資料。規則由子句組成。

您可以將使用 Guard DSL 寫入的規則儲存到使用任何副檔名的純文字檔案中。

您可以建立多個規則檔案，並將其分類為規則集。規則集可讓您同時針對多個規則檔案驗證 JSON 或 YAML 格式的資料。

## 主題

- [子句](#)
- [在子句中使用查詢](#)
- [在子句中使用運算子](#)
- [在子句中使用自訂訊息](#)
- [合併子句](#)
- [搭配 Guard 規則使用區塊](#)
- [使用內建函數](#)
- [定義 Guard 查詢和篩選](#)
- [在 Guard 規則中指派和參考變數](#)
- [在中編寫具名規則區塊 AWS CloudFormation Guard](#)
- [撰寫子句以執行內容感知評估](#)

## 子句

子句是評估為 true (PASS) 或 false (FAIL) 的布林表達式。子句使用二進位運算子來比較在單一值上操作的兩個值或單一運算子。

### Unary 子句的範例

下列 unary 子句會評估集合是否為 `TcpBlockedPorts` 空。

```
InputParameters.TcpBlockedPorts not empty
```

下列 unary 子句會評估 `ExecutionRoleArn` 屬性是否為字串。

```
Properties.ExecutionRoleArn is_string
```

### 二進位子句的範例

下列二進位子句會評估 `BucketName` 屬性是否包含字串 `encrypted`，無論大小寫為何。

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

下列二進位子句會評估 ReadCapacityUnits 屬性是否小於或等於 5,000。

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

## 撰寫 Guard 規則子句的語法

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

## Guard 規則子句的屬性

### query

以點 (.) 分隔的表達式，寫入以周遊階層資料。查詢表達式可以包含篩選條件表達式，以值的子集為目標。您可以將查詢指派給變數，以便您可以寫入它們一次，並在規則集中的其他位置參考它們，這可讓您存取查詢結果。

如需撰寫查詢和篩選的詳細資訊，請參閱 [定義查詢和篩選](#)。

必要：是

### operator

單一或二進位運算子，可協助檢查查詢的狀態。二進位運算子的左側 (LHS) 必須是查詢，而右側 (RHS) 必須是查詢或值常值。

支援的二進位運算子：== (等於) | != (不等於) | > (大於) | >= (大於或等於) | < (小於) | <= (小於或等於) | IN (在形式 **【x、y、z】** 的清單中)

支援的 unary 運算子：exists | empty | is\_string | is\_list | is\_struct | not(!)

必要：是

### query|value literal

查詢或支援的值常值，例如 string 或 integer(64)。

支援的值常值：

- 所有基本類型：string、integer(64)、float(64)、bool、char、regex
- 表達 integer(64)、float(64) 或範圍的所有專用 char 範圍類型，表示為：

- `r[<lower_limit>, <upper_limit>]`，其會轉譯為滿足下列表達k式的任何值：  
`lower_limit <= k <= upper_limit`
- `r[<lower_limit>, <upper_limit>)k`，其會轉譯為滿足下列表達式的任何值：  
`lower_limit <= k < upper_limit`
- `r(<lower_limit>, <upper_limit>]`，其會轉譯為滿足下列表達k式的任何值：  
`lower_limit < k <= upper_limit`
- `r(<lower_limit>, <upper_limit>)`，可轉換為滿足下列表達k式的任何值：  
`lower_limit < k < upper_limit`
- 巢狀索引鍵/值結構資料的關聯陣列（對應）。例如：

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

- 基本類型或關聯陣列類型的陣列

必要：有條件；使用二進位運算子時為必要。

#### custom message

提供子句相關資訊的字串。訊息會顯示在 `validate` 和 `test` 命令的詳細輸出中，有助於了解或偵錯階層資料上的規則評估。

必要：否

## 在子句中使用查詢

如需撰寫查詢的資訊，請參閱 [定義查詢和篩選](#) 和 [在 Guard 規則中指派和參考變數](#)。

## 在子句中使用運算子

以下是 CloudFormation 範本 Template-1 和 的範例 Template-2。為了示範如何使用支援的運算子，本節中的範例查詢和子句會參考這些範例範本。

### 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

```

## 使用 unary 運算子的子句範例

- `empty` – 檢查集合是否為空。您也可以使用它來檢查查詢在階層資料中是否有值，因為查詢會導致集合。您無法使用它來檢查字串值查詢是否已定義空字串 ("")。如需詳細資訊，請參閱[定義查詢和篩選](#)。

下列子句會檢查範本是否已定義一或多個資源。它評估為 `PASS` 因為具有邏輯 ID 的資源 `S3Bucket` 是在 `Template-1` 中定義。

```
Resources !empty
```

下列子句會檢查是否為 `S3Bucket` 資源定義一或多個標籤。它會評估為 `PASS` 因為 `Template-1` 中 `S3Bucket` 有兩個為 `Tags` 屬性定義的標籤。

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

- `exists` – 檢查每個查詢的出現是否具有值，並可用於取代 `!= null`。

下列子句會檢查是否已為 定義 `BucketEncryption` 屬性 `S3Bucket`。它評估為 `PASS` 因為 `BucketEncryption` 是在 `S3Bucket` 中為 定義的 `Template-1`。

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

### Note

`empty` 和 `not exists` 檢查會在周遊輸入資料時評估 `true` 是否遺失屬性索引鍵。例如，如果未在 的範本中定義 `Properties` 區段 `S3Bucket`，則 子句 `Resources.S3Bucket.Properties.Tag empty` 評估為 `true`。`exists` 和 `empty` 檢查不會在錯誤訊息中顯示文件內的 JSON 指標路徑。這兩個子句通常都有無法維護此周遊資訊的擷取錯誤。

- `is_string` – 檢查查詢的每個出現是否為 `string` 類型。

下列子句會檢查是否為 `S3Bucket` 資源的 `BucketName` 屬性指定字串值。它會評估為 `PASS` 因為字串值 `"MyServiceS3Bucket"` 是在 `BucketName` 中為 指定的 `Template-1`。

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

- `is_list` – 檢查查詢的每個出現是否為 `list` 類型。

下列子句會檢查是否為 `S3Bucket` 資源的 `Tags` 屬性指定清單。它評估為 `PASS` 因為在 `Tags` 中為 指定了兩個鍵值對 `Template-1`。

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

- `is_struct` – 檢查查詢是否每次出現都是結構化資料。

下列子句會檢查是否為 `S3Bucket` 資源的 `BucketEncryption` 屬性指定結構化資料。它會評估為 `PASS` 因為 `BucketEncryption` 是使用 中的 `ServerSideEncryptionConfiguration` 屬性類型 `#####` 指定 `Template-1`。

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

### Note

若要檢查反轉狀態，您可以使用 ( not ! ) 運算子搭配 is\_string、is\_list 和 is\_struct 運算子。

## 使用二進位運算子的子句範例

下列子句會檢查 中 S3Bucket 資源 BucketName 屬性指定的值是否 Template-1 包含字串 encrypt，無論大小寫為何。這會評估為 FAIL，PASS 因為指定的儲存貯體名稱 "MyServiceS3Bucket" 不包含字串 encrypt。

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

下列子句會檢查 中 NewVolume 資源 Size 屬性指定的值是否 Template-2 在特定範圍內：50 <= Size <= 200。它會評估為 FAIL，PASS 因為 100 是針對所指定 Size。

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

下列子句會檢查 中 NewVolume 資源 VolumeType 屬性指定的值是否為 Template-2 io1、io2 或 gp3。它會評估為 FAIL，PASS 因為 io1 是針對所指定 NewVolume。

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

### Note

本節中的範例查詢示範使用具有邏輯 IDs S3Bucket 和 NewVolume 的資源來使用運算子。資源名稱通常是使用者定義的，並且可以在基礎設施中任意命名為程式碼 (IaC) 範本。若要撰寫一般規則並套用至範本中定義的所有 AWS::S3::Bucket 資源，最常使用的查詢形式是 Resources.\*[ Type == 'AWS::S3::Bucket' ]。如需詳細資訊，請參閱 [定義查詢和篩選](#) 以取得用量的詳細資訊，並探索 cloudformation-guard GitHub 儲存庫中的 [範例目錄](#)。

## 在子句中使用自訂訊息

在下列範例中，的子句Template-2包含自訂訊息。

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

## 合併子句

在 Guard 中，在新行上寫入的每個子句會使用結合（布林and邏輯）隱含地與下一個子句結合。請參閱以下範例。

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

您也可以透過or|OR在第一個子句結尾指定，使用解散來結合子句與下一個子句。

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

在 Guard 子句中，會先評估接合點，接著評估接合點。Guard 規則可以定義為將評估為 (PASS) 或 true() 的子句 false( or|ORand|AND的 ) 解譯的組合FAIL。這類似於 [Conjunctive 正常形式](#)。

下列範例示範子句的評估順序。

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

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

```
# (clause_L v clause_M v clause_N) ^ clause_0
clause_L OR
clause_M OR
clause_N
clause_0
```

所有以範例為基礎的子句Template-1都可以使用 `結合` 來合併。請參閱以下範例。

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

## 搭配 Guard 規則使用區塊

區塊是從一組相關子句、條件或規則中移除詳細程度和重複性的組合。區塊有三種類型：

- 查詢區塊
- when 區塊
- 具名規則區塊

### 查詢區塊

以下是以範例 為基礎的 子句Template-1。結合用於結合 子句。

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

每個子句中的查詢表達式部分都會重複。您可以使用查詢區塊，改善可編譯性，並從具有相同初始查詢路徑的一組相關子句中移除詳細程度和重複性。可以寫入同一組子句，如下列範例所示。

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

```
    BucketName != /(?!i)encrypt/  
    BucketEncryption exists  
    BucketEncryption is_struct  
    Tags is_list  
    Tags !empty  
}
```

在查詢區塊中，區塊前面的查詢會設定區塊內子句的內容。

如需使用區塊的詳細資訊，請參閱 [編寫具名規則區塊](#)。

## when 區塊

您可以使用採用下列格式的區塊來有條件地評估when區塊。

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

when 關鍵字會指定when區塊的開頭。condition 是 Guard 規則。只有在條件的評估結果為 true() 時，才會評估 區塊PASS。

以下是以 為基礎的範例when區塊Template-1。

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

只有在為 指定的值BucketName是字串時，才會評估when區塊中的 子句。如果在範本的 Parameters區段中參考為 BucketName 指定的值，如下列範例所示，則不會評估when區塊中的 子句。

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

```

    BucketName:
      Ref: S3BucketName
    ...

```

## 具名規則區塊

您可以將名稱指派給一組規則 (規則集)，然後在其他規則中參考這些模組化驗證區塊，稱為命名規則區塊。命名規則區塊採用下列形式。

```

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

```

`rule` 關鍵字會指定 `name-rule` 區塊的開頭。

`rule name` 是人類可讀取的字串，可唯一識別具名規則區塊。這是其封裝的 Guard 規則集的標籤。在此使用中，Guard 規則一詞包含子句、查詢區塊、`when` 區塊和命名規則區塊。規則名稱可用來參考其封裝的規則集評估結果，這使得命名規則區塊可重複使用。規則名稱也提供 `validate` 和 `test` 命令輸出中規則失敗的相關內容。規則名稱會與區塊的評估狀態 (PASS、FAIL 或 SKIP) 一起顯示在規則檔案的評估輸出中。請參閱以下範例。

```

# 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**

```

您也可以在規則名稱後面指定 `when` 關鍵字後面接著條件，以有條件的方式評估具名規則區塊。

以下是本主題先前討論的範例 `when` 區塊。

```

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

```

使用具名規則區塊，上述也可以編寫如下。

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

您可以使用其他 Guard 規則重複使用並分組具名規則區塊。以下是幾個範例。

```
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
    ...
}
```

## 使用內建函數

AWS CloudFormation Guard 提供內建函數，您可以在規則中用來執行字串操作、JSON 剖析和資料類型轉換等操作。函式僅透過指派至變數來支援。

## 關鍵函數

`json_parse(json_string)`

從範本剖析內嵌 JSON 字串。剖析之後，您可以評估所產生物件的屬性。

`count(collection)`

傳回查詢解析的項目數量。

`regex_replace(base_string, regex_to_extract, regex_replacement)`

使用規則表達式取代字串的一部分。

如需可用函數的完整清單，包括字串操作、集合操作和資料類型轉換函數，請參閱 Guard GitHub 儲存庫中的[函數文件](#)。

## 定義 Guard 查詢和篩選

本主題涵蓋寫入查詢，以及在寫入 Guard 規則子句時使用篩選。

### 先決條件

篩選是一種進階 AWS CloudFormation Guard 概念。我們建議您先檢閱下列基本主題，再了解如何篩選：

- [什麼是 AWS CloudFormation Guard ?](#)
- [撰寫規則、子句](#)

### 定義查詢

查詢表達式是寫入周遊階層資料的簡單點 (.) 分隔表達式。查詢表達式可以包含篩選條件表達式，以值的子集為目標。評估查詢時，會產生一系列的值，類似於從 SQL 查詢傳回的結果集。

下列範例查詢會在 CloudFormation 範本中搜尋 `AWS::IAM::Role` 資源。

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

查詢遵循下列基本原則：

- 使用明確索引鍵術語時，查詢的每個點 (.) 部分會沿著階層向下周遊，例如 `Resources` 或 `Properties.Encrypted`。如果查詢的任何部分不符合傳入的基準，Guard 會擲回擷取錯誤。

- 使用萬用字元的查詢的點 \* (.) 部分會周遊該層級結構的所有值。
- 使用陣列萬用字元的查詢的點 [\*] (.) 部分會周遊該陣列的所有索引。
- 您可以透過在方括號 內指定篩選條件來篩選所有集合 []。可以透過下列方式遇到集合：

- 基準中自然發生的陣列是集合。以下是 範例：

連接埠： [20, 21, 110, 190]

標籤： [{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]

- 周遊結構的所有值時，例如 Resources.\*
- 任何查詢結果本身就是可從中進一步篩選值的集合。請參閱以下範例。

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

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

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

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

以下是 CloudFormation 範本程式碼片段的範例。

```
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
...

```

根據此範本，周遊的路徑為 `Resources.SampleRole`，`SampleRole` 選取的最終值為 `Type: AWS::IAM::Role`。

```

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

```

下列範例會顯示 `Resources.*[ Type == 'AWS::IAM::Role' ]` YAML 格式的查詢結果值。

```

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

```

您可以使用查詢的一些方式如下：

- 將查詢指派給變數，以便透過參考這些變數來存取查詢結果。
- 使用針對每個所選值進行測試的區塊來追蹤查詢。
- 直接比較查詢與基本子句。

## 將查詢指派給變數

Guard 支援指定範圍內的一次性變數指派。如需 Guard 規則中變數的詳細資訊，請參閱 [在 Guard 規則中指派和參考變數](#)。

您可以將查詢指派給變數，以便您寫入查詢一次，然後在 Guard 規則的其他位置參考它們。請參閱以下範例變數指派，示範本節稍後討論的查詢原則。

```

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

#
# A more complex query here (this will be explained below)
#
let iam_policies_allowing_log_creates = Resources.*[

```

```

Type in [/IAM::Policy/, /IAM::ManagedPolicy/]
some Properties.PolicyDocument.Statement[*] {
  some Action[*] == 'cloudwatch:CreateLogGroup'
  Effect == 'Allow'
}
]

```

## 直接從指派給查詢的變數逐一查看值

Guard 支援直接針對查詢的結果執行。在下列範例中，when 區塊會針對 CloudFormation 範本中找到的每個 AWS::EC2::Volume 資源，針對 VolumeType、Encrypted 和 AvailabilityZone 屬性進行測試。

```

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

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

```

## 直接子句層級比較

Guard 也支援查詢作為直接比較的一部分。請參閱下列範例。

```

let resources = Resources.*

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

```

在上述範例中，以所示形式表示的兩個子句（以 some 關鍵字開頭）會被視為獨立子句，並分別進行評估。

## 單一子句和區塊子句表單

總而言之，上一節中顯示的兩個範例子句不等同於下列區塊。

```

let resources = Resources.*

```

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

此區塊會查詢集合中每個Tag值，並將其屬性值與預期的屬性值進行比較。上一節中子句的組合形式會獨立評估兩個子句。請考慮下列輸入。

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

第一個形式的子句會評估為 PASS。驗證第一個形式的第一個子句時，跨 Resources、Tags、Properties 和 的下列路徑Key符合 值NotPRODEnd，且不符合預期的 值PROD。

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

第一個表單的第二個子句也是如此。Resources、Tags、Properties 和 之間的路徑Value符合值 AppStart。因此，第二個子句會獨立。

整體結果為 PASS。

不過，區塊形式會評估如下。對於每個Tags值，它會比較 Key 和 是否都Value相符；在以下範例中，NotAppStart 和 NotPRODEnd 值不相符。

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

由於評估會檢查 `Key == /PROD$/`、和 `Value == /^App/`，因此配對未完成。因此，結果為 FAIL。

### Note

使用集合時，建議您在想要比較集合中每個元素的多個值時，使用區塊子句表單。當集合是一組純量值，或您只打算比較單一屬性時，請使用單一子句表單。

## 查詢結果和相關聯的子句

所有查詢都會傳回值清單。周遊的任何部分，例如遺失的金鑰、存取所有索引時的陣列空值 (`Tags: []`)，或遇到空的映射時地圖的遺失值 (`Resources: {}`)，都可能導致擷取錯誤。

在針對此類查詢評估子句時，所有擷取錯誤都會被視為失敗。唯一的例外是在查詢中使用明確篩選條件時。使用篩選條件時，會略過相關聯的子句。

下列區塊故障與執行中的查詢相關聯。

- 如果範本不包含資源，則查詢會評估為 FAIL，而相關聯的區塊層級子句也會評估為 FAIL。
- 當範本包含像是 的空白資源區塊時 `{ "Resources": {} }`，查詢會評估為 FAIL，而相關聯的區塊層級子句也會評估為 FAIL。
- 如果範本包含資源，但沒有資源符合查詢，則查詢會傳回空白結果，並略過區塊層級子句。

## 在查詢中使用篩選條件

查詢中的篩選條件實際上是做為選取條件的 Guard 子句。以下是子句的結構。

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

當您使用篩選條件 [撰寫 AWS CloudFormation Guard 規則](#) 時，請記住下列重點：

- 使用 [並行法線格式 \(CNF\)](#) 結合子句。
- 在新行上指定每個結合 (and) 子句。
- 在兩個子句之間使用 `or` 關鍵字指定 (or)。

下列範例示範了連接子句和解除連接子句。

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

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

### 使用子句做為選取條件

您可以將篩選套用至任何集合。篩選可以直接套用到已經是等集合之輸入中的屬性 `securityGroups: [...]`。您也可以針對查詢套用篩選，查詢一律是值的集合。您可以使用子句的所有功能進行篩選，包括並行法線形式。

從 CloudFormation 範本依類型選取資源時，通常會使用下列常見查詢。

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

查詢會 `Resources.*` 傳回輸入 `Resources` 區段中存在的所有值。對於 `Resources` 中的範例範本輸入 [定義查詢](#)，查詢會傳回下列項目。

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

```
...
- Type: AWS::EC2::Subnet
...
```

現在，針對此集合套用篩選條件。要符合的條件是 `Type == AWS::IAM::Role`。以下是套用篩選條件後查詢的輸出。

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

接著，檢查 `AWS::IAM::Role` 資源的各種子句。

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

以下是篩選查詢的範例，可選取所有 `AWS::IAM::Policy` 和 `AWS::IAM::ManagedPolicy` 資源。

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

下列範例會檢查這些政策資源是否已 `PolicyDocument` 指定。

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

建置更複雜的篩選需求

請考慮下列輸入和輸出安全群組資訊的 `AWS Config` 組態項目範例。

```
---
resourceType: 'AWS::EC2::SecurityGroup'
configuration:
  ipPermissions:
    - fromPort: 172
      ipProtocol: tcp
```

```
    toPort: 172
    ipv4Ranges:
      - cidrIp: 10.0.0.0/24
      - cidrIp: 0.0.0.0/0
  - fromPort: 89
    ipProtocol: tcp
    ipv6Ranges:
      - cidrIpv6: ':::/0'
    toPort: 189
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 1.1.1.1/32
  - fromPort: 89
    ipProtocol: '-1'
    toPort: 189
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 1.1.1.1/32
ipPermissionsEgress:
  - ipProtocol: '-1'
    ipv6Ranges: []
    prefixListIds: []
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 0.0.0.0/0
    ipRanges:
      - 0.0.0.0/0
tags:
  - key: Name
    value: good-sg-delete-me
vpcId: vpc-0123abcd
InputParameter:
  TcpBlockedPorts:
    - 3389
    - 20
    - 21
    - 110
    - 143
```

注意下列事項：

- ipPermissions (輸入規則) 是組態區塊內的規則集合。
- 每個規則結構都包含屬性，例如 ipv4Ranges 和 ipv6Ranges，以指定 CIDR 區塊的集合。

讓我們編寫規則，選取允許來自任何 IP 地址連線的任何輸入規則，並驗證規則不允許公開 TCP 封鎖的連接埠。

從涵蓋 IPv4 的查詢部分開始，如下列範例所示。

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

`some` 關鍵字在此內容中很有用。所有查詢都會傳回符合查詢的值集合。根據預設，Guard 會評估因查詢而傳回的所有值是否與檢查相符。不過，這種行為不一定是您需要檢查的內容。請考慮組態項目的下列部分輸入。

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

有兩個值可供使用 `ipv4Ranges`。並非所有 `ipv4Ranges` 值都等於以表示的 IP 地址 `0.0.0.0/0`。您想要查看至少一個值是否符合 `0.0.0.0/0`。您告訴 Guard，不是從查詢傳回的所有結果都需要相符，但至少有一個結果必須相符。`some` 關鍵字會通知 Guard，以確保結果查詢中的一或多個值符合檢查。如果沒有相符的查詢結果值，Guard 會擲回錯誤。

接著，新增 IPv6，如下列範例所示。

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

最後，在下列範例中，驗證通訊協定不是 `udp`。

```
configuration.ipPermissions[
```

```

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

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

```

以下是完整的規則。

```

rule any_ip_ingress_checks
{

  let ports = InputParameter.TcpBlockedPorts[*]

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

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

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

```

```

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

```

### 根據集合的包含類型來分隔集合

使用基礎設施做為程式碼 (IaC) 組態範本時，您可能會遇到一個集合，其中包含組態範本中其他實體的參考。以下是 CloudFormation 範本範例，描述 Amazon Elastic Container Service (Amazon ECS) 任務，其中包含的本機參考 TaskArn、的 TaskRoleArn 參考，以及直接字串參考。

```

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'

```

請看下列查詢。

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

此查詢會傳回值的集合，其中包含範例範本中顯示的所有三個AWS::ECS::TaskDefinition資源。分隔ecs\_tasks包含來自其他的TaskRoleArn本機參考，如下列範例所示。

```

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

let ecs_tasks_role_direct_strings = %ecs_tasks[
  Properties.TaskRoleArn is_string ]

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

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

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

```

```
>>  
}
```

## 在 Guard 規則中指派和參考變數

您可以在 AWS CloudFormation Guard 規則檔案中指派變數，以存放要在 Guard 規則中參考的資訊。Guard 支援一次性變數指派。變數會延遲評估，這表示 Guard 只會在規則執行時評估變數。

### 主題

- [指派變數](#)
- [參考變數](#)
- [變數範圍](#)
- [Guard 規則檔案中的變數範例](#)

### 指派變數

使用 `let` 關鍵字來初始化和指派變數。最佳實務是針對變數名稱使用蛇案例。變數可以存放查詢產生的靜態常值或動態屬性。在下列範例中，變數會 `ecs_task_definition_task_role_arn` 存放靜態字串值 `arn:aws:iam:123456789012:role/my-role-name`。

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

在下列範例中，變數會 `ecs_tasks` 儲存查詢的結果，以搜尋範本中的所有 `AWS::ECS::TaskDefinition` CloudFormation 資源。您可以在撰寫規則時參考 `ecs_tasks` 來存取這些資源的相關資訊。

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

### 參考變數

使用 `%` 字首來參考變數。

根據 [中的 `ecs\_task\_definition\_task\_role\_arn` 變數範例](#) [指派變數](#)，您可以在 Guard 規則子句的 `query|value literal` 區段 `ecs_task_definition_task_role_arn` 中參考。使用該參考可確保為 CloudFormation 範本中任何 `AWS::ECS::TaskDefinition` 資源的 `TaskDefinitionArn` 屬性指定的值是靜態字串值 `arn:aws:iam:123456789012:role/my-role-name`。

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

根據 `ecs_tasks` 變數範例 [指派變數](#)，您可以在查詢 `ecs_tasks` 中參考（例如 `%ecs_tasks.Properties`）。首先，Guard 會評估變數，`ecs_tasks` 然後使用傳回的值來周遊階層。如果變數 `ecs_tasks` 解析為非字串值，則 Guard 會擲回錯誤。

### Note

目前，Guard 不支援在自訂錯誤訊息內參考變數。

## 變數範圍

範圍是指規則檔案中定義的變數可見性。變數名稱只能在範圍內使用一次。有三個層級可以宣告變數，或三個可能的變數範圍：

- 檔案層級 – 通常在規則檔案頂端宣告，您可以在規則檔案內的所有規則中使用檔案層級變數。它們對整個檔案可見。

在下列範例規則檔案中，變數 `ecs_task_definition_task_role_arn` 和 `ecs_task_definition_execution_role_arn` 會在檔案層級初始化。

```
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
}
```

- 規則層級 – 在規則中宣告，規則層級變數僅對該特定規則可見。規則以外的任何參考都會導致錯誤。

在下列範例規則檔案中，變數 `ecs_task_definition_task_role_arn` 和 `ecs_task_definition_execution_role_arn` 會在規則層級初始化。`ecs_task_definition_task_role_arn` 只能在 `check_ecs_task_definition_task_role_arn` 具名規則中參考。您只能在 `check_ecs_task_definition_execution_role_arn` 具名規則中參考 `ecs_task_definition_execution_role_arn` 變數。

```
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
}
```

- 區塊層級 – 在區塊內宣告，例如子 `when` 句，區塊層級變數只能由該特定區塊顯示。區塊以外的任何參考都會導致錯誤。

在下列範例規則檔案中，變數 `ecs_task_definition_task_role_arn` 和 `ecs_task_definition_execution_role_arn` 會在 `AWS::ECS::TaskDefinition` 類型區塊內的區塊層級初始化。您只能參考 `AWS::ECS::TaskDefinition` 類型區塊中的 `ecs_task_definition_task_role_arn` 和 `ecs_task_definition_execution_role_arn` 變數，以取得其個別規則。

```
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
  }
}
```

## Guard 規則檔案中的變數範例

下列各節提供變數靜態和動態指派的範例。

### 靜態指派

以下是 CloudFormation 範本範例。

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

根據此範本，您可以撰寫稱為的規則 `check_ecs_task_definition_task_role_arn`，以確保所有 `AWS::ECS::TaskDefinition` 範本資源的 `TaskRoleArn` 屬性為 `arn:aws:iam::123456789012:role/my-role-name`。

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

在規則範圍內，您可以初始化名為的變數 `ecs_task_definition_task_role_arn`，並將靜態字串值指派給該變數 `'arn:aws:iam::123456789012:role/my-role-name'`。規則子句會參考 `query|value literal` 區段中的 `ecs_task_definition_task_role_arn` 變數，`arn:aws:iam::123456789012:role/my-role-name` 檢查為 `EcsTask` 資源 `TaskRoleArn` 屬性指定的值是否為。

## 動態指派

以下是 CloudFormation 範本範例。

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

根據此範本，您可以初始化檔案 `ecs_tasks` 範圍內名為 `ecs_tasks` 的變數，並將查詢指派給該變數 `Resources.*[ Type == 'AWS::ECS::TaskDefinition'`。Guard 會查詢輸入範本中的所有資源，並將這些資源的相關資訊儲存在 `ecs_tasks`。您也可以撰寫稱為 `check_ecs_task_definition_task_role_arn` 的規則，以確保所有 `AWS::ECS::TaskDefinition` 範本資源的 `TaskRoleArn` 屬性為 `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'
}
```

規則子句會參考 `query` 區段中的 `ecs_task_definition_task_role_arn` 變數，`arn:aws:iam::123456789012:role/my-role-name` 以檢查為 `EcsTask` 資源 `TaskRoleArn` 屬性指定的值是否為。

## 強制執行 CloudFormation 範本組態

讓我們演練更複雜的生產使用案例範例。在此範例中，我們撰寫 Guard 規則，以確保更嚴格控制 Amazon ECS 任務的定義方式。

以下是 CloudFormation 範本範例。

```
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'

```

根據此範本，我們會撰寫下列規則，以確保符合這些要求：

- 範本中的每個AWS::ECS::TaskDefinition資源都連接了任務角色和執行角色。
- 任務角色和執行角色是 AWS Identity and Access Management (IAM) 角色。
- 角色在範本中定義。
- 系統會為每個角色指定 PermissionsBoundary 屬性。

```

# 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::GetAtt-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::GetAtt-retrievable attribute
let execution_role_refs = some %ecs_tasks.Properties.ExecutionRoleArn.'Fn::GetAtt'[0]

# Verify requirement #1
rule all_ecs_tasks_must_have_task_end_execution_roles
  when %ecs_tasks !empty
  {
    %ecs_tasks.Properties {

```

```
    TaskRoleArn exists
    ExecutionRoleArn exists
  }
}

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

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

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

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

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

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

## 在中編寫具名規則區塊 AWS CloudFormation Guard

使用 撰寫具名規則區塊時 AWS CloudFormation Guard，您可以使用下列兩種合成樣式：

- 條件相依性
- 關聯性相依性

使用這些類型的相依性合成有助於提升可重複使用性，並減少具名規則區塊中的詳細程度和重複性。

## 主題

- [先決條件](#)
- [條件相依性合成](#)
- [關聯性相依性合成](#)

## 先決條件

了解[撰寫規則](#)中的具名規則區塊。

## 條件相依性合成

在此合成風格中，when區塊或具名規則區塊的評估對一或多個其他具名規則區塊或子句的評估結果具有條件相依性。下列範例 Guard 規則檔案包含示範條件相依性的具名規則區塊。

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

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

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

# Example-3, Named-rule block, rule_name_C, takes a conditional dependency on
rule_name_A ^ rule_name_B
rule rule_name_C when rule_name_A
```

```

        rule_name_B {
    Guard_rule_3
    Guard_rule_4
    ...
}

# Example-4, Named-rule block, rule_name_D, takes a conditional dependency on
# (rule_name_A v clause_A) ^ clause_B ^ rule_name_B
rule rule_name_D when rule_name_A OR
    clause_A
    clause_B
    rule_name_B {
    Guard_rule_3
    Guard_rule_4
    ...
}

```

在上述範例規則檔案中， Example-1有下列可能的結果：

- 如果 rule\_name\_A 評估為 PASS， rule\_name\_B 則會評估 封裝的 Guard 規則。
- 如果 rule\_name\_A 評估為 FAIL， rule\_name\_B 則不會評估 封裝的 Guard 規則。 rule\_name\_B 評估為 SKIP。
- 如果 rule\_name\_A 評估為 SKIP， rule\_name\_B 則不會評估 封裝的 Guard 規則。 rule\_name\_B 評估為 SKIP。

#### Note

如果 rule\_name\_A 條件式取決於評估為 FAIL 的規則， FAIL 並導致 rule\_name\_A 評估為 SKIP 的規則，就會發生這種情況 SKIP。

以下是來自傳入和傳出安全群組資訊項目的組態管理資料庫 (CMDDB) 組態 AWS Config 項目範例。此範例示範條件相依性合成。

```

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

rule check_parameter_validity when check_resource_type_and_parameter {
    InputParameters.TcpBlockedPorts[*] {

```

```

        this in r[0,65535]
    }
}

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

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

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

```

在上述範例中，`check_parameter_validity` 有條件地相依於 `check_resource_type_and_parameter`，而 `check_ip_procotol_and_port_range_validity` 有條件地相依於 `check_parameter_validity`。以下是符合上述規則的組態管理資料庫 (CMDB) 組態項目。

```

---
version: '1.3'
resourceType: 'AWS::EC2::SecurityGroup'
resourceId: sg-12345678abcdefghi
configuration:
    description: Delete-me-after-testing
    groupName: good-sg-test-delete-me

```

```
ipPermissions:
  - fromPort: 172
    ipProtocol: tcp
    ipv6Ranges: []
    prefixListIds: []
    toPort: 172
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 0.0.0.0/0
    ipRanges:
      - 0.0.0.0/0
  - fromPort: 89
    ipProtocol: tcp
    ipv6Ranges:
      - cidrIpv6: '::/0'
    prefixListIds: []
    toPort: 89
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 0.0.0.0/0
    ipRanges:
      - 0.0.0.0/0
ipPermissionsEgress:
  - ipProtocol: '-1'
    ipv6Ranges: []
    prefixListIds: []
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 0.0.0.0/0
    ipRanges:
      - 0.0.0.0/0
tags:
  - key: Name
    value: good-sg-delete-me
vpcId: vpc-0123abcd
InputParameters:
  TcpBlockedPorts:
    - 3389
    - 20
    - 110
    - 142
    - 1434
    - 5500
supplementaryConfiguration: {}
```

```
resourceTransitionStatus: None
```

## 關聯性相依性合成

在此合成風格中，when區塊或具名規則區塊的評估對一或多個其他 Guard 規則的評估結果具有相互依存性。關聯性相依性可如下所示。

```
# 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
  ...
}
```

為了協助您了解相互依存性合成，請檢閱 Guard 規則檔案的下列範例。

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

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

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

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

在上述規則檔案中，對 `ensure_elbs_are_internal_and_secure` 具有相互依存性 `ensure_all_elbs_are_secure`。以下是符合上述規則的範例 CloudFormation 範本。

```
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...'
```

## 撰寫子句以執行內容感知評估

AWS CloudFormation Guard 子句會根據階層資料進行評估。Guard 評估引擎會使用簡單的虛線表示法，依照指定的階層式資料來解決對傳入資料的查詢。通常需要多個子句，才能根據資料映射或集合進行評估。Guard 提供方便的語法來寫入這類子句。引擎具有情境感知，並使用與評估相關聯的對應資料。

以下是包含容器的 Kubernetes Pod 組態範例，您可以將內容感知評估套用到其中。

```
apiVersion: v1  
kind: Pod  
metadata:
```

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

您可以撰寫 Guard 子句來評估此資料。評估規則檔案時，內容是整個輸入文件。以下是驗證 Pod 中指定容器限制強制執行的範例子句。

```
#
# At this level, the root document is available for evaluation
#
#
# Our rule only evaluates for apiVersion == v1 and K8s kind is Pod
#
rule ensure_container_limits_are_enforced
  when apiVersion == 'v1'
    kind == 'Pod'
{
  spec.containers[*] {
    resources.limits {
      #
      # Ensure that cpu attribute is set
      #
      cpu exists
    }
  }
}
```

```

        Id: K8S_REC_18
        Description: CPU limit must be set for the container
    >>

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

```

## context 了解評估

在規則封鎖層級，傳入內容是完整的文件。when 條件的評估會針對 apiVersion 和 kind 屬性所在的傳入根內容進行。在先前的範例中，這些條件會評估為 true。

現在，在中周遊階層，spec.containers[\*] 如上述範例所示。對於階層的每個周遊，內容值會隨之變更。spec 區塊的周遊完成後，內容會變更，如下列範例所示。

```

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

```

周遊 `containers` 屬性之後，內容會顯示在下列範例中。

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

## 了解迴圈

您可以使用 表達式 `[*]` 為 `containers` 屬性的陣列中包含的所有值定義迴圈。系統會針對 內的每個元素評估 區塊 `containers`。在上述範例規則程式碼片段中，區塊中包含的子句會定義要根據容器定義驗證的檢查。內部包含的子句區塊會評估兩次，每個容器定義一次。

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

對於每次反覆運算，內容值是該對應索引的值。

### Note

支援的唯一索引存取格式為 `[<integer>]` 或 `[*]`。目前，Guard 不支援 等範圍 `[2..4]`。

## 陣列

通常在可接受陣列的地方，也會接受單一值。例如，如果只有一個容器，則可以捨棄陣列，並接受下列輸入。

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

如果屬性可以接受陣列，請確定您的規則使用陣列形式。在上述範例中，您使用 `containers[*]`，而不是 `containers`。Guard 只會在遇到單一值表單時，在資料周遊時正確評估。

### Note

屬性接受陣列時，在表達規則子句的存取權時，請務必使用陣列形式。即使使用單一值，Guard 也會正確評估。

## 使用表單 `spec.containers[*]` 而非 `spec.containers`

Guard 查詢會傳回已解析值的集合。當您使用表單時 `spec.containers`，查詢的解析值包含所參照的陣列 `containers`，而非其中的元素。當您使用表單時 `spec.containers[*]`，您可以參考每個包含的個別元素。當您想要評估陣列中包含的每個元素時，請記得使用 `[*]` 表單。

## 使用 `this` 參考目前的內容值

當您編寫 Guard 規則時，您可以使用參考內容值 `this`。通常，`this` 是隱含的，因為它繫結到內容的值。例如，`this.apiVersion`、`this.kind` 和 `this.spec` 繫結至根或文件。相反地，`this.resources` 繫結至的每個值 `containers`，例如 `/spec/containers/0/` 和 `/`

spec/containers/1。同樣地，`this.cpu` 和 `this.memory` 對應至限制，特別是 `/spec/containers/0/resources/limits` 和 `/spec/containers/1/resources/limits`。

在下一個範例中，Kubernetes Pod 組態的上述規則會重寫為 `this` 明確使用。

```
rule ensure_container_limits_are_enforced
  when this.apiVersion == 'v1'
    this.kind == 'Pod'
{
  this.spec.containers[*] {
    this.resources.limits {
      #
      # Ensure that cpu attribute is set
      #
      this.cpu exists
      <<
        Id: K8S_REC_18
        Description: CPU limit must be set for the container
      >>

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

您不需要 `this` 明確使用。不過，使用純量時 `this` 參考可能很有用，如下列範例所示。

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

在上一個範例中，`this` 用於參考每個連接埠號碼。

## 使用隱含的潛在錯誤 **this**

編寫規則和子句時，參考隱含this內容值的元素時有一些常見的錯誤。例如，請考慮要評估的下列輸入基準（這必須通過）。

```
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
```

根據上述範本進行測試時，下列規則會導致錯誤，因為它不正確地假設利用隱含 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
      >>
    }
  }
}

```

若要逐步解說此範例，請將上述規則檔案儲存為檔案名稱 `any_ip_ingress_check.guard`，並將資料儲存為檔案名稱 `ip_ingress.yaml`。然後，使用這些檔案執行下列 `validate` 命令。

```

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

```

在下列輸出中，引擎指出其嘗試擷取值 `InputParameters.TcpBlockedPorts[*]` 上的屬性/`configuration/ipPermissions/0/configuration/ipPermissions/1` 失敗。

```

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[*]

```

為了協助了解此結果，請使用 `this` 明確參考重新撰寫規則。

```

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` 參考變數 內包含的每個值 `any_ip_permissions`。指派給變數的查詢會選取相符 `configuration.ipPermissions` 的值。錯誤表示嘗試在此內容 `InputParameters` 中擷取，但 `InputParameters` 位於根內容中。

內部區塊也會參考超出範圍的變數，如下列範例所示。

```

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

```

```

    }
  }
}

```

`this` 是指中的每個連接埠值 [21, 22, 110]，但也是指 `fromPort` 和 `toPort`。它們都屬於外部區塊範圍。

### 解決隱含使用的錯誤 `this`

使用變數來明確指派和參考值。首先，`InputParameter.TcpBlockedPorts` 是輸入（根）內容的一部分。移 `InputParameter.TcpBlockedPorts` 出內部區塊並明確指派，如下列範例所示。

```

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

```

然後，明確參考此變數。

```

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
            >>
        }
    }
}
}

```

對 中的內部this參考執行相同的操作%ports。

不過，所有錯誤尚未修正，因為內部的迴圈ports仍有不正確的參考。下列範例顯示移除不正確的參考。

```

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

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

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

```

```

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

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

```

接著，再次執行 `validate` 命令。這次，它會通過。

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

以下是 `validate` 命令的輸出。

```

ip_ingress.yaml Status = PASS
PASS rules
check_ip_procotol_and_port_range_validity    PASS

```

若要測試此方法是否失敗，下列範例會使用承載變更。

```
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 的範圍介於 89–109 之間，且允許任何 IPv6 地址。以下是再次執行 `validate` 命令之後的輸出。

```

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

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

```

# 測試 AWS CloudFormation Guard 規則

您可以使用 AWS CloudFormation Guard 內建單元測試架構來驗證 Guard 規則是否如預期般運作。本節提供如何撰寫單元測試檔案，以及如何使用 `test` 命令來測試規則檔案的逐步解說。

您的單元測試檔案必須具有下列其中一個副檔名：`.json`、`.JSON`、`.json.yaml`、`.YAML`、或 `.yaml`。

## 主題

- [先決條件](#)
- [Guard 單位測試檔案概觀](#)
- [撰寫 Guard 規則單位測試檔案的演練](#)

## 先決條件

撰寫 Guard 規則來評估您的輸入資料。如需詳細資訊，請參閱[撰寫 Guard 規則](#)。

## Guard 單位測試檔案概觀

Guard 單位測試檔案是 JSON 或 YAML 格式的檔案，其中包含多個輸入，以及在 Guard 規則檔案中寫入規則的預期結果。可以有多個範例來評估不同的期望。我們建議您先測試空白輸入，然後逐步新增評估各種規則和子句的資訊。

此外，我們建議您使用尾碼 `_test.json` 或命名單元測試檔案 `_tests.yaml`。例如，如果您有一個名為 `my_rules.guard` 的規則檔案，請命名您的單元測試檔案 `my_rules_tests.yaml`。

## 語法

以下顯示 YAML 格式的單位測試檔案語法。

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

## Properties

以下是 Guard 測試檔案的屬性。

### input

要測試規則的資料。我們建議您的第一個測試使用空白輸入，如下列範例所示。

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

對於後續測試，新增要測試的輸入資料。

必要：是

### expectations

根據您的輸入資料評估特定規則時的預期結果。除了每個規則的預期結果之外，指定您要測試的一或多個規則。預期結果必須是下列其中一項：

- PASS – 當對您的輸入資料執行時，規則會評估為 true。
- FAIL – 當對您的輸入資料執行時，規則會評估為 false。
- SKIP – 對輸入資料執行時，不會觸發規則。

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

必要：是

## 撰寫 Guard 規則單位測試檔案的演練

以下是名為 `api_gateway_private.guard` 的規則檔案。此規則的目的是檢查 CloudFormation 範本中定義的所有 Amazon API Gateway 資源類型是否僅部署為私有存取。它也會檢查至少一個政策陳述式是否允許從虛擬私有雲端 (VPC) 存取。

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

此演練會測試第一個規則意圖：所有部署AWS::ApiGateway::RestApi的資源都必須是私有的。

1. 建立名為的單元測試檔案`api_gateway_private_tests.yaml`，其中包含下列初始測試。在初始測試中，新增空白輸入，並預期規則`check_rest_api_is_private`會略過，因為沒有`AWS::ApiGateway::RestApi`資源做為輸入。

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

2. 使用 `test` 命令在終端機中執行第一個測試。針對 `--rules-file` 參數，指定您的規則檔案。針對 `--test-data` 參數，指定您的單位測試檔案。

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

第一個測試的結果是 PASS。

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

3. 將另一個測試新增至您的單元測試檔案。現在，擴展測試以包含空白資源。以下是更新的 `api_gateway_private_tests.yaml` 檔案。

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

4. `test` 使用更新的單位測試檔案執行。

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

第二個測試的結果是 PASS。

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

5. 將另外兩個測試新增至您的單元測試檔案。擴展測試以包含下列項目：

- 未指定屬性 `AWS::ApiGateway::RestApi` 的資源。

#### Note

這不是有效的 CloudFormation 範本，但即使輸入格式不正確，測試規則是否正常運作也很有用。

預期此測試會失敗，因為未指定 `EndpointConfiguration` 屬性，因此未設定為 `PRIVATE`。

- 一種 `AWS::ApiGateway::RestApi` 資源，其滿足 `EndpointConfiguration` 屬性設定為的第一個意圖，`PRIVATE` 但無法滿足第二個意圖，因為它未定義政策陳述式。預期此測試將會通過。

以下是更新的單元測試檔案。

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

```

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

```

## 6. test 使用更新的單位測試檔案執行。

```

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

```

第三個結果是 FAIL，第四個結果是 PASS。

```

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

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

Test Case #3
Name: "MyTest3"

```

## PASS Rules:

```
check_rest_api_is_private: Expected = FAIL, Evaluated = FAIL
```

## Test Case #4

```
Name: "MyTest4"
```

## PASS Rules:

```
check_rest_api_is_private: Expected = PASS, Evaluated = PASS
```

7. 在您的單元測試檔案中註解測試 1–3。僅存取第四個測試的詳細內容。以下是更新的單位測試檔案。

```
---
#- 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
```



9. 將如第四個測試案例的測試案例新增至具有指定 `EndpointConfiguration` 屬性之 `AWS::ApiGateway::RestApi` 資源的單元測試檔案。測試案例將會失敗，而不是通過。以下是更新的單位測試檔案。

```
---
#- 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]
```



```

        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 命令的詳細輸出遵循規則檔案的結構。規則檔案中的每個區塊都是詳細輸出中的區塊。最上方的區塊是每個規則。如果有針對規則when的條件，這些條件會出現在同盟條件區塊中。在下列範例中，%api\_gws !empty測試條件並通過。

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

一旦條件通過，我們會測試規則子句。

```

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

```

%api\_gws 是區塊規則，對應至輸出中的BlockClause層級（行：21）。規則子句是一組結合 (AND) 子句，其中每個結合子句都是一組接合 (OR)。結合具有單一子句 Properties.EndpointConfiguration.Types[\*] == "PRIVATE"。因此，詳細輸出會顯示單一子句。路徑/Resources/apiGw/Properties/EndpointConfiguration/Types/1會顯示要比較輸入中的哪些值，在此案例中是索引Types為 1 的元素。

在中[根據 Guard 規則驗證輸入資料](#)，您可以使用本節中的範例，使用 validate 命令根據規則評估輸入資料。

## 搭配 AWS CloudFormation Guard 規則使用輸入參數

AWS CloudFormation Guard 可讓您在驗證期間使用輸入參數進行動態資料查詢。當您需要參考規則中的外部資料時，此功能特別有用。不過，指定輸入參數索引鍵時，Guard 要求沒有衝突的路徑。

## 如何使用

1. 使用 `--input-parameters` 或 `-i` 旗標來指定包含輸入參數的檔案。您可以指定多個輸入參數檔案，並將合併以形成共同內容。輸入參數索引鍵不能有衝突的路徑。
2. 使用 `--data` 或 `-d` 旗標指定要驗證的實際範本檔案。

## 範例使用方式

1. 建立輸入參數檔案 ( 例如 `network.yaml` ) :

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

2. 在您的防護規則檔案中參考這些參數 ( 例如 `security_groups.guard` ) :

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

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

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

3. 建立失敗的資料範本 ( 例如 `security_groups_fail.yaml` ) :

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

Resources:
  mySecurityGroup:
```

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

#### 4. 執行驗證命令：

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

在此命令中：

- `-r` 會指定規則檔案。
- `-i` 指定輸入參數檔案。
- `-d` 指定要驗證的資料檔案（範本）。

## 多個輸入參數

您可以指定多個輸入參數檔案：

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

使用 指定的所有檔案 `-i` 都會合併，以形成參數查詢的單一內容。

## 根據 AWS CloudFormation Guard 規則驗證輸入資料

您可以使用 AWS CloudFormation Guard `validate` 命令來驗證 Guard 規則的資料。如需 `validate` 命令的詳細資訊，包括其參數和選項，請參閱[驗證](#)。

### 先決條件

- 寫入 Guard 規則以驗證您的輸入資料。如需詳細資訊，請參閱[撰寫 Guard 規則](#)。
- 測試您的規則，以確保它們如預期般運作。如需詳細資訊，請參閱[測試 Guard 規則](#)。

## 使用 `validate` 命令

若要根據 Guard 規則驗證輸入資料，例如 AWS CloudFormation 範本，請執行 Guard `validate` 命令。針對 `--rules` 參數，指定規則檔案的名稱。針對 `--data` 參數，指定輸入資料檔案的名稱。

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

如果 Guard 成功驗證範本，`validate`命令會傳回結束狀態 `0` ( `$?`  中的)。如果 Guard 識別規則違規，`validate`命令會傳回失敗規則的狀態報告。使用摘要旗標 (`-s all`) 來查看詳細評估樹狀結構，其中顯示 Guard 如何評估每個規則。

```
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]
```

## 針對多個資料檔案驗證多個規則

為了協助維護規則，您可以將規則寫入多個檔案，並根據需要組織規則。然後，您可以針對資料檔案或多個資料檔案驗證多個規則檔案。validate 命令可以取得 --data 和 --rules 選項的檔案目錄。例如，您可以執行下列命令，其中 /path/to/dataDirectory 包含一或多個資料檔案，以及 /path/to/ruleDirectory 包含一或多個規則檔案。

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

您可以撰寫規則來檢查多個 CloudFormation 範本中定義的各種資源是否具有適當的屬性指派，以確保靜態加密。為了簡化搜尋和維護，您可以在具有路徑的目錄中，使用稱為 s3\_bucket\_encryption.guard、ec2\_volume\_encryption.guard 和的個別檔案中，在每個資源 rds\_dbinstance\_encryption.guard 中檢查靜態加密的規則 ~/GuardRules/encryption\_at\_rest。您需要驗證的 CloudFormation 範本位於路徑為的目錄中 ~/CloudFormation/templates。在此情況下，請執行 validate 命令，如下所示。

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

# 故障診斷 AWS CloudFormation Guard

如果您在使用時遇到問題 AWS CloudFormation Guard，請參閱本節中的主題。

## 主題

- [當沒有所選類型的資源時，子句失敗](#)
- [Guard 不會評估具有短格式 Fn::GetAtt 參考的 CloudFormation 範本](#)
- [一般疑難排解主題](#)

## 當沒有所選類型的資源時，子句失敗

當查詢使用類似的篩選條件時 `Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`，如果輸入中沒有 `AWS::ApiGateway::RestApi` 資源，子句會評估為 FAIL。

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

若要避免此結果，請將篩選條件指派給變數，並使用 `when` 條件檢查。

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

## Guard 不會評估具有短格式 Fn::GetAtt 參考的 CloudFormation 範本

Guard 不支援短形式的內部函數。例如，不支援在 YAML 格式 CloudFormation 範本 `!Join!Sub` 中使用。反之，請使用擴充形式的 CloudFormation 內部函數。例如，在 YAML 格式的 CloudFormation 範本 `Fn::Sub` 中使用 `Fn::Join`，根據 Guard 規則進行評估。

如需內部函數的詳細資訊，請參閱 AWS CloudFormation 《使用者指南》中的 [內部函數參考](#)。

## 一般疑難排解主題

- 確認 `string` 常值不包含內嵌的逸出字串。Guard 不支援 `string` 常值的內嵌逸出字串。如果您的意圖是剖析內嵌 JSON 字串，請使用 Guard 3.0.0 和更新版本中提供的 `json_parse()` 函數。如需詳細資訊，請參閱 [使用內建函數](#)。

- 確認您的 `!=` 比較比較相容的資料類型。例如，`string` 和 `int` 是不相容的比較資料類型。執行 `!=` 比較時，如果值不相容，則會在內部發生錯誤。目前，錯誤會被隱藏並轉換為 `false` 以滿足 Rust 中的 [PartialEq](#) 特性。

# AWS CloudFormation Guard CLI 參數和命令參考

下列全域參數和命令可透過 AWS CloudFormation Guard 命令列界面 (CLI) 取得。

## 主題

- [Guard CLI 全域參數](#)
- [parse-tree](#)
- [rulegen](#)
- [test](#)
- [validate](#)

## Guard CLI 全域參數

您可以搭配任何 CLI AWS CloudFormation Guard 命令使用下列參數。

`-h, --help`

列印說明資訊。

`-V, --version`

列印版本資訊。

## parse-tree

為 AWS CloudFormation Guard 規則檔案中定義的規則產生剖析樹狀結構。

## 語法

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

## Parameters

`-h, --help`

列印說明資訊。

`-p, --print-json`

以 JSON 格式列印輸出。

`-y, --print-yaml`

以 YAML 格式列印輸出。

`-V, --version`

列印版本資訊。

## 選項

`-o, --output`

將產生的樹寫入輸出檔案。

`-r, --rules`

提供規則檔案。

## 範例

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

## rulegen

採用 JSON 或 YAML 格式的 AWS CloudFormation 範本檔案，並自動產生一組符合範本資源屬性的 AWS CloudFormation Guard 規則。此命令是開始使用規則撰寫或從已知良好的範本建立 ready-to-use 規則的有用方法。

## 語法

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

## Parameters

`-h, --help`

列印說明資訊。

`-V, --version`

列印版本資訊。

## 選項

`-o, --output`

將產生的規則寫入輸出檔案。由於可能存在數百甚至數千個規則，我們建議使用此選項。

`-t, --template`

以 JSON 或 YAML 格式提供 CloudFormation 範本檔案的路徑。

## 範例

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

## test

根據 JSON 或 YAML 格式的 Guard 單位測試檔案驗證 AWS CloudFormation Guard 規則檔案，以判斷個別規則是否成功。

## 語法

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

## Parameters

`-a, --alphabetical`

在目錄中依字母順序排序。

`-h, --help`

列印說明資訊。

`-m, --last-modified`

依目錄中上次修改的時間排序

`-V, --version`

列印版本資訊。

`-v, --verbose`

增加輸出詳細程度。可以多次指定。

詳細輸出遵循 Guard 規則檔案的結構。規則檔案中的每個區塊都是詳細輸出中的區塊。最上方的區塊是每個規則。如果有針對規則when的條件，它們會顯示為同盟條件區塊。

## 選項

`-d, --dir`

提供規則的根目錄。

`-o, --output-format`

指定輸出應顯示的格式。

預設：`single-line-summary`

允許的值：`json | yaml | single-line-summary | junit`

`-r, --rules-file`

提供規則檔案的名稱。

`-t, --test-data`

提供 JSON 或 YAML 格式之資料檔案的檔案名稱或目錄。

## 範例

```
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
```

## 另請參閱

[測試 Guard 規則](#)

## validate

根據 AWS CloudFormation Guard 規則驗證資料，以判斷成功或失敗。

## 語法

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

## Parameters

-a, --alphabetical

依字母順序驗證目錄中的檔案。

-h, --help

列印說明資訊。

-m, --last-modified

驗證目錄中依上次修改時間排序的檔案。

-P, --payload

透過 以下列 JSON 格式提供規則和資料stdin :

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

例如：

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

對於「規則」，指定規則檔案的字串版本清單。針對「資料」，指定資料檔案的字串版本清單。

--payload 指定 --rules 且 --data 無法指定時。

-p, --print-json

以 JSON 格式列印輸出。

-s, --show-clause-failures

顯示子句失敗，包括摘要。

-V, --version

列印版本資訊。

-v, --verbose

增加輸出詳細程度。可以多次指定。

-z, --structured

列印結構化和有效的 JSON/YAML 清單。此引數與下列引數衝突：verbose，print-json，show-summary：all/fail/pass/skip，output-format：single-line-summary

## 選項

-d, --data (字串)

提供 JSON 或 YAML 中的資料檔案或資料檔案目錄。重複使用此選項，支援傳遞多個值。

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

對於data-dir1上述的目錄引數，僅支援掃描具有下列副檔名的檔案：

案：.yaml、.yml、.json、.jsn、.template

如果您指定 --payload旗標，請勿指定 --data選項。

**-i、--input-parameters ( 字串 )**

提供 JSON 或 YAML 中參數檔案的參數檔案或目錄，指定要與要用作合併內容的資料檔案搭配使用的任何其他參數。做為輸入傳遞的所有參數檔案都會合併，而此合併內容會再次與做為引數傳遞的每個檔案合併data。因此，每個檔案都應該包含互斥屬性，而不會有任何重疊。重複使用此選項，支援傳遞多個值。

對於目錄引數，僅支援掃描具有下列副檔名的檔案：.yaml、.yml、.json、.jsn、.template

**-o、--output-format ( 字串 )**

指定輸出的格式。

預設：single-line-summary

允許的值：json | yaml | single-line-summary | junit | sarif

**-r、--rules ( 字串 )**

提供規則檔案或規則檔案目錄。重複使用此選項，支援傳遞多個值。

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

對於上述等目錄引數rules-dir1，僅支援掃描具有下列副檔名的檔案：.guard、.ruleset

如果您指定 --payload旗標，請勿指定 --rules選項。

**--show-summary (string)**

控制是否需要顯示摘要資料表。--show-summary fail ( 預設 ) 或 --show-summary pass, fail ( 僅顯示已通過/失敗的規則 ) 或 --show-summary none ( 關閉 ) 或 --show-summary all ( 顯示通過、失敗或略過的所有規則 )。

預設：fail

允許的值：none | all | pass | fail | skip

-t、--type ( 字串 )

提供輸入資料的格式。當您指定輸入資料類型時，Guard 會在輸出中顯示 CloudFormation 範本資源的邏輯名稱。根據預設，Guard 會顯示屬性路徑和值，例如 Property [/Resources/vol2/Properties/Encrypted]。

允許值：CFNTemplate

## 範例

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

## Output

如果 Guard 成功驗證範本，validate 命令會傳回結束狀態為 0 (\$? 在 bash 中)。如果 Guard 識別規則違規，validate 命令會傳回失敗規則的狀態報告。

```
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".]
```

## 另請參閱

- [根據 Guard 規則驗證輸入資料](#)
- [搭配 Guard 規則使用輸入參數](#)

## 中的安全性 AWS CloudFormation Guard

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間共同責任。[共同責任模型](#)將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。AWS 也為您提供可安全使用的服務。在[AWS 合規計畫](#)中，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用於 Guard 的合規計畫，請參閱[AWS 合規計畫的服務範圍](#)。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您的公司的要求和適用法律和法規

下列文件可協助您了解如何在將 [Guard 安裝為 AWS Lambda 函數](#) (cfn-guard-lambda) 時套用共同責任模型：

- AWS Command Line Interface 《使用者指南》中的[安全性](#)
- 《AWS Lambda 開發人員指南》中的[安全性](#)
- AWS Identity and Access Management 《使用者指南》中的[安全性](#)

# AWS CloudFormation Guard 文件歷史紀錄

下表說明 文件的發行版本 AWS CloudFormation Guard。

- 文件最近更新時間：2025 年 7 月 30 日
- 最新版本：3.1.2

變更	描述	日期
<a href="#">文件更新</a>	更新 Guard CLI 命令參考文件，以符合目前的實作。版本參考已更新至 Guard 3.1.2。	2025 年 7 月 30 日
<a href="#">3.0.0 版</a>	3.0.0 版推出下列改進功能： <ul style="list-style-type: none"><li>• 已更新 Guard 3.0.0 版本的簡介和安裝主題。</li><li>• 新增 Homebrew 和 的安裝說明Chocolatey。</li><li>• 更新遷移 Guard 規則的相關資訊，以反映 Guard 3.0.0 版中的變更。</li><li>• 已將醒目連結新增至 AWS CloudFormation Guard GitHub 儲存庫。</li></ul>	2023 年 6 月 30 日
<a href="#">2.1.3 版</a>	2.1.3 版推出下列改進功能：  已新增 Guard 2.1.3 增強功能的相關資訊。Guard 2.0 的參考已更新為 Guard 2.1.3。	2023 年 6 月 9 日
<a href="#">2.0.4 版</a>	2.0.4 版推出下列改進功能：  <code>--payload</code> 旗標已新增至 <code>validate</code> 命令。	2021 年 10 月 19 日

如需詳細資訊，請參閱 Guard CLI 參考中的[驗證](#)。

### [2.0.3 版](#)

2.0.3 版推出下列改進功能： 2021 年 7 月 27 日

- 您可以在單元測試檔案中提供每個測試的測試名稱。如需詳細資訊，請參閱[測試 Guard 規則](#)。
- 下列選項已新增至 `validate` 命令：
  - `--output-format`
  - `--show-summary`
  - `--type`

如需詳細資訊，請參閱 Guard CLI 參考中的[驗證](#)。

### [初始版本](#)

AWS CloudFormation Guard 2021 年 7 月 15 日  
使用者指南的初始版本。

# AWS 詞彙表

如需最新的 AWS 術語，請參閱 AWS 詞彙表 參考中的[AWS 詞彙表](#)。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。