Add a permission의

AWS CloudFormation Guard

AWS CloudFormation Guard: Add a permission의

Table of Contents

AWS CloudFormation Guard란 무엇인가요? 1
Guard를 처음 사용하시나요? 1
Guard 기능 2
CloudFormation 후크에서 Guard 사용 2
Guard 액세스 2
모범 사례 2
Guard 설정 3
Linux 및 macOS용 3
사전 구축된 릴리스 바이너리에서 Guard 설치 3
Guard from Cargo 설치 4
Homebrew에서 Guard 설치 5
Windows의 경우 5
사전 조건 5
Guard from Cargo 설치 4
Chocolatey에서 Guard 설치 6
AWS Lambda 함수로 7
사전 조건 7
Rust 패키지 관리자 설치 7
Guard를 Lambda 함수로 설치하려면 8
빌드 및 실행 9
Lambda 함수 호출 9
Guard 규칙 사용에 대한 사전 조건 및 개요 10
사전 조건 10
Guard 규칙 사용 개요 10
Guard 규칙 작성 10
절 11
절에서 쿼리 사용 13
절에서 연산자 사용 13
절에서 사용자 지정 메시지 사용 17
절 결합 17
Guard 규칙과 함께 블록 사용 18
내장 함수 사용 22
쿼리 정의 및 필터링 22
Guard 규칙에서 변수 할당 및 참조 36

명명된 규칙 블록 구성 42
컨텍스트 인식 평가를 수행하기 위한 절 작성 48
Guard 규칙 테스트 61
사전 조건 61
개요 62
안내 63
Guard 규칙과 함께 입력 파라미터 사용 72
사용 방법 73
사용 예시 73
다중 입력 파라미터 74
Guard 규칙에 대한 입력 데이터 검증 74
사전 조건 74
`validate` 명령 사용 75
여러 데이터 파일에 대해 여러 규칙 검증 76
Guard 문제 해결 77
선택한 유형의 리소스가 없는 경우 절이 실패합니다. 77
Guard는 CloudFormation 템플릿을 평가하지 않습니다. 77
일반 문제 해결 주제 77
Guard CLI 참조 79
Guard CLI 전역 파라미터 79
구문 분석 트리 79
구문 79
파라미터 79
옵션 80
예제 80
규칙 생성 80
구문 80
파라미터 81
옵션 81
예제 81
테스트 81
구문 81
파라미터 81
옵션 82
예제 83
출력 83

다음 사항도 참조하세요. 83
검증 83
구문 83
파라미터 83
옵션 85
예제 86
출력 86
다음 사항도 참조하세요. 87
보안 88
문서 이력 89
AWS 용어집 91
..... xcii

AWS CloudFormation Guard란 무엇인가요?

AWS CloudFormation Guard 는 오픈 소스, 범용, 코드형 정책 평가 도구입니다. Guard 명령줄 인터페이스(CLI)는 정책을 코드로 표현하는 데 사용할 수 있는 simple-to-use 선언적인 도메인별 언어(DSL)를 제공합니다. 또한 CLI 명령을 사용하여 이러한 규칙에 대해 구조화된 계층적 JSON 또는 YAML 데이터를 검증할 수 있습니다. 또한 Guard는 규칙이 의도한 대로 작동하는지 확인하는 기본 제공 단위 테스트 프레임워크를 제공합니다.

Guard는 CloudFormation 템플릿에서 유효한 구문 또는 허용된 속성 값을 확인하지 않습니다. cfn-lint 도구를 사용하여 템플릿 구조를 철저히 검사할 수 있습니다.

Guard는 서버 측 적용을 제공하지 않습니다. CloudFormation 후크를 사용하여 작업을 차단하거나 경고할 수 있는 서버 측 검증 및 적용을 수행할 수 있습니다.

AWS CloudFormation Guard 개발에 대한 자세한 내용은 Guard GitHub 리포지토리를 참조하세요.

주제

- Guard를 처음 사용하시나요?
- Guard 기능
- CloudFormation 후크에서 Guard 사용
- Guard 액세스
- 모범 사례

Guard를 처음 사용하시나요?

Guard를 처음 사용하는 경우 먼저 다음 섹션을 읽는 것이 좋습니다.

- Guard 설정 -이 섹션에서는 Guard를 설치하는 방법을 설명합니다. Guard를 사용하면 Guard DSL을 사용하여 정책 규칙을 작성하고 해당 규칙에 대해 JSON 또는 YAML 형식의 구조화된 데이터를 검증할 수 있습니다.
- Guard 규칙 작성 -이 섹션에서는 정책 규칙 작성에 대한 자세한 연습을 제공합니다.
- Guard 규칙 테스트 -이 섹션에서는 규칙이 의도한 대로 작동하는지 테스트하고 규칙에 대해 JSON 또는 YAML 형식의 구조화된 데이터를 검증하는 방법에 대한 자세한 설명을 제공합니다.
- Guard 규칙에 대한 입력 데이터 검증 -이 섹션에서는 규칙에 대해 JSON 또는 YAML 형식의 구조화된 데이터를 검증하기 위한 자세한 연습을 제공합니다.
- Guard CLI 참조 -이 섹션에서는 Guard CLI에서 사용할 수 있는 명령을 설명합니다.

Guard 기능

Guard를 사용하면 CloudFormation 템플릿을 포함하되 이에 국한되지 않는 JSON 또는 YAML 형식의 구조화된 데이터를 검증하는 정책 규칙을 작성할 수 있습니다. Guard는 정책 검사의 end-to-end 평가의 전체 스펙트럼을 지원합니다. 규칙은 다음 비즈니스 도메인에서 유용합니다.

- 예방적 거버넌스 및 규정 준수(전환 왼쪽 테스트) - 보안 및 규정 준수를 위한 조직의 모범 사례를 나타내는 정책 규칙에 따라 코드형 인프라(IaC) 또는 인프라 및 서비스 구성을 검증합니다. 예를 들어 CloudFormation 템플릿, CloudFormation 변경 세트, JSON 기반 Terraform 구성 파일 또는 Kubernetes 구성을 검증할 수 있습니다.
- 탐지 거버넌스 및 규정 준수 AWS Config- 기반 구성 항목(CIs. 예를 들어 개발자는 AWS Config CIs에 대한 가드 정책을 사용하여 배포된 리소스 AWS 및 비AWS 리소스의 상태를 지속적으로 모니터링하고, 정책 위반을 감지하고, 문제 해결을 시작할 수 있습니다.
- 배포 안전 - 배포 전에 변경 사항이 안전한지 확인합니다. 예를 들어 정책 규칙에 대해 CloudFormation 변경 세트를 검증하여 Amazon DynamoDB 테이블 이름 변경과 같이 리소스 교체로 이어지는 변경을 방지합니다.

CloudFormation 후크에서 Guard 사용

CloudFormation Guard를 사용하여 CloudFormation 후크에서 후크를 작성할 수 있습니다. CloudFormation 후크를 사용하면 CloudFormation에서 작업을 생성, 업데이트 또는 삭제하고 작업을 AWS Cloud Control API 생성 또는 업데이트하기 전에 Guard 규칙을 사전에 적용할 수 있습니다. 후크는 리소스 구성이 조직의 보안, 운영 및 비용 최적화 모범 사례를 준수하는지 확인합니다.

Guard를 사용하여 CloudFormation Guard 후크를 작성하는 방법에 대한 자세한 내용은 후크 사용 설명서의 Guard Hook에 대한 리소스를 평가하기 위한 Guard 규칙 쓰기를 참조하세요. CloudFormation

Guard 액세스

Guard DSL 및 명령에 액세스하려면 Guard CLI를 설치해야 합니다. Guard CLI 설치에 대한 자세한 내용은 섹션을 참조하세요Guard 설정.

모범 사례

간단한 규칙을 작성하고 명명된 규칙을 사용하여 다른 규칙에서 참조합니다. 복잡한 규칙은 유지 관리 및 테스트하기 어려울 수 있습니다.

설 AWS CloudFormation Guard정

AWS CloudFormation Guard 는 오픈 소스 명령줄 인터페이스(CLI)입니다. 정책 규칙을 작성하고 해당 규칙에 대해 구조화된 계층적 JSON 및 YAML 데이터를 검증할 수 있는 간단한 도메인별 언어를 제공합니다. 규칙은 보안, 규정 준수 등과 관련된 회사 정책 지침을 나타낼 수 있습니다. 구조화된 계층적 데이터는 코드로 설명된 클라우드 인프라를 나타낼 수 있습니다. 예를 들어 CloudFormation 템플릿에서 암호화된 Amazon Simple Storage Service(Amazon S3) 버킷을 항상 모델링하도록 규칙을 생성할 수 있습니다.

다음 주제에서는 선택한 운영 체제를 사용하거나 AWS Lambda 함수로 Guard를 설치하는 방법에 대한 정보를 제공합니다.

주제

- Linux 및 macOS용 Guard 설치
- Windows용 Guard 설치
- Guard를 AWS Lambda 함수로 설치

Linux 및 macOS용 Guard 설치

미리 빌드된 릴리스 바이너리인 Cargo를 사용하거나 Homebrew를 통해 Linux 및 macOS AWS CloudFormation Guard 용를 설치할 수 있습니다.

사전 구축된 릴리스 바이너리에서 Guard 설치

다음 절차에 따라 사전 빌드된 바이너리에서 Guard를 설치합니다.

1. 터미널을 열고 다음 명령을 실행합니다.

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

2. 다음 명령을 실행하여 PATH 변수를 설정합니다.

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

   결과: Guard를 성공적으로 설치하고 PATH 변수를 설정했습니다.

   - (선택 사항) Guard 설치를 확인하려면 다음 명령을 실행합니다.

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

명령은 다음 출력을 반환합니다.

```
cfn-guard 3.1.2
```

Guard from Cargo 설치

Cargo는 Rust 패키지 관리자입니다. 다음 단계를 완료하여 Cargo를 포함하는 Rust를 설치합니다. 그런 다음 Guard from Cargo를 설치합니다.

1. 터미널에서 다음 명령을 실행하고 화면의 지침에 따라 Rust를 설치합니다.

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

   - (선택 사항) Ubuntu 환경에서 다음 명령을 실행합니다.

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

2. PATH 환경 변수를 구성하고 다음 명령을 실행합니다.

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

3. Cargo가 설치된 상태에서 다음 명령을 실행하여 Guard를 설치합니다.

   ```
   cargo install cfn-guard
   ```

   결과: Guard를 성공적으로 설치했습니다.

   - (선택 사항) Guard 설치를 확인하려면 다음 명령을 실행합니다.

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

     명령은 다음 출력을 반환합니다.

     ```
     cfn-guard 3.1.2
     ```

Homebrew에서 Guard 설치

Homebrew는 macOS 및 Linux용 패키지 관리자입니다. Homebrew를 설치하려면 다음 단계를 완료하세요. 그런 다음 Homebrew에서 Guard를 설치합니다.

1. 터미널에서 다음 명령을 실행하고 화면의 지침에 따라 Homebrew를 설치합니다.

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

2. Homebrew가 설치된 상태에서 다음 명령을 실행하여 Guard를 설치합니다.

   ```
   brew install cloudformation-guard
   ```

   결과: Guard를 성공적으로 설치했습니다.

   - (선택 사항) Guard 설치를 확인하려면 다음 명령을 실행합니다.

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

     명령은 다음 출력을 반환합니다.

     ```
     cfn-guard 3.1.2
     ```

Windows용 Guard 설치

Cargo 또는 Chocolatey를 통해 Windows AWS CloudFormation Guard 용를 설치할 수 있습니다.

사전 조건

명령줄 인터페이스에서 Guard를 빌드하려면 Visual Studio 2019용 빌드 도구를 설치해야 합니다.

1. Visual Studio 2019용 빌드 도구 웹 사이트에서 Microsoft Visual C++ 빌드 도구를 다운로드합니다.
2. 설치 프로그램을 실행하고 기본값을 선택합니다.

Guard from Cargo 설치

Cargo는 Rust 패키지 관리자입니다. 다음 단계를 완료하여 Cargo를 포함하는 Rust를 설치합니다. 그런 다음 Guard from Cargo를 설치합니다.

1. Rust를 다운로드한 다음 rustup-init.exe를 실행합니다.
2. 명령 프롬프트에서 기본 옵션인 1을 선택합니다.

   명령은 다음 출력을 반환합니다.

   ```
   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. 설치를 완료하려면 Enter 키를 누릅니다.
4. Cargo가 설치된 상태에서 다음 명령을 실행하여 Guard를 설치합니다.

   ```
   cargo install cfn-guard
   ```

   결과: Guard를 성공적으로 설치했습니다.

   - (선택 사항) Guard 설치를 확인하려면 다음 명령을 실행합니다.

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

     명령은 다음 출력을 반환합니다.

     ```
     cfn-guard 3.1.2
     ```

Chocolatey에서 Guard 설치

Chocolatey는 Windows용 패키지 관리자입니다. Chocolatey를 설치하려면 다음 단계를 완료하세요. 그런 다음 Chocolatey에서 Guard를 설치합니다.

1. 이 가이드에 따라 Chocolatey를 설치합니다.
2. Chocolatey가 설치된 상태에서 다음 명령을 실행하여 Guard를 설치합니다.

```
choco install cloudformation-guard
```

결과: Guard를 성공적으로 설치했습니다.

- (선택 사항) Guard 설치를 확인하려면 다음 명령을 실행합니다.

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

명령은 다음 출력을 반환합니다.

```
cfn-guard 3.1.2
```

Guard를 AWS Lambda 함수로 설치

Rust 패키지 관리자인 Cargo를 AWS CloudFormation Guard 통해를 설치할 수 있습니다. 함수(`cfn-guard-lambda`)로서의 가드 AWS Lambda는 Lambda 함수로 사용할 수 있는 Guard(`cfn-guard`) 주위의 경량 래퍼입니다.

사전 조건

Guard를 Lambda 함수로 설치하려면 먼저 다음 사전 조건을 충족해야 합니다.

- AWS Command Line Interface Lambda 함수를 배포하고 호출할 수 있는 권한으로 구성된 (AWS CLI)입니다. 자세한 내용은 AWS CLI구성 섹션을 참조하세요.
- AWS Identity and Access Management (IAM)의 AWS Lambda 실행 역할입니다. 자세한 내용은 AWS Lambda 실행 역할을 참조하세요.
- CentOS/RHEL 환경에서 `musl-libc` 패키지 리포지토리를 yum 구성에 추가합니다. 자세한 내용은 ngompa/musl-libc를 참조하세요.

Rust 패키지 관리자 설치

Cargo는 Rust 패키지 관리자입니다. 다음 단계를 완료하여 Cargo를 포함하는 Rust를 설치합니다.

1. 터미널에서 다음 명령을 실행한 다음 화면의 지침에 따라 Rust를 설치합니다.

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

- (선택 사항) Ubuntu 환경에서 다음 명령을 실행합니다.

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

2. PATH 환경 변수를 구성하고 다음 명령을 실행합니다.

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

Lambda 함수(Linux, macOS 또는 Unix)로 Guard 설치

Guard를 Lambda 함수로 설치하려면 다음 단계를 완료합니다.

1. 명령 터미널에서 다음 명령을 실행합니다.

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

   - (선택 사항) Guard를 Lambda 함수로 설치했는지 확인하려면 다음 명령을 실행합니다.

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

     명령은 다음 출력을 반환합니다.

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

2. musl 지원을 설치하려면 다음 명령을 실행합니다.

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

3. 를 사용하여 빌드musl한 다음 터미널에서 다음 명령을 실행합니다.

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

   사용자 지정 런타임의 경우 bootstrap 배포 패키지 .zip 파일에 이름이 인 실행 파일이 AWS Lambda 필요합니다. 생성된 cfn-lambda 실행 파일의 이름을 로 바꾼 bootstrap 다음 .zip 아카이브에 추가합니다.

- macOS 환경의 경우 Rust 프로젝트의 루트 또는에서 화물 구성 파일을 생성합니다~/.cargo/config.

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

4. cfn-guard-lambda 루트 디렉터리로 변경합니다.

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

5. 터미널에서 다음 명령을 실행합니다.

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

6. 다음 명령을 실행하여 계정에 Lambda 함수cfn-guard로를 제출합니다.

```
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 함수로 Guard를 빌드하고 실행하려면

Lambda 함수cfn-guard-lambda로 제출된를 호출하려면 다음 명령을 실행합니다.

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

Lambda 함수 요청 구조를 호출하려면

요청에는 다음 필드가 cfn-guard-lambda 필요합니다.

- data - YAML 또는 JSON 템플릿의 문자열 버전
- rules - 규칙 세트 파일의 문자열 버전

Guard 규칙 사용에 대한 사전 조건 및 개요

이 섹션에서는 JSON 또는 YAML 형식 데이터에 대한 규칙 작성, 테스트 및 검증이라는 핵심 Guard 작업을 완료하는 방법을 보여줍니다. 또한 특정 사용 사례에 대응하는 규칙 작성을 보여주는 자세한 연습이 포함되어 있습니다.

주제

- 사전 조건
- Guard 규칙 사용 개요
- AWS CloudFormation Guard 규칙 작성
- AWS CloudFormation Guard 규칙 테스트
- AWS CloudFormation Guard 규칙과 함께 입력 파라미터 사용
- AWS CloudFormation Guard 규칙에 대한 입력 데이터 검증

사전 조건

Guard 도메인별 언어(DSL)를 사용하여 정책 규칙을 작성하려면 먼저 Guard 명령줄 인터페이스(CLI)를 설치해야 합니다. 자세한 내용은 Guard 설정 단원을 참조하십시오.

Guard 규칙 사용 개요

Guard를 사용할 때는 일반적으로 다음 단계를 수행합니다.

1. JSON 또는 YAML 형식의 데이터를 작성하여 검증합니다.
2. Guard 정책 규칙을 작성합니다. 자세한 내용은 Guard 규칙 작성 단원을 참조하십시오.
3. Guard `test` 명령을 사용하여 규칙이 의도한 대로 작동하는지 확인합니다. 단위 테스트에 대한 자세한 내용은 섹션을 참조하세요Guard 규칙 테스트.
4. Guard `validate` 명령을 사용하여 규칙에 대해 JSON 또는 YAML 형식의 데이터를 검증합니다. 자세한 내용은 Guard 규칙에 대한 입력 데이터 검증 단원을 참조하십시오.

AWS CloudFormation Guard 규칙 작성

에서 AWS CloudFormation Guard규칙은 policy-as-code 규칙입니다. JSON 또는 YAML 형식의 데이터를 검증할 수 있는 규칙을 Guard 도메인별 언어(DSL)로 작성합니다. 규칙은 절로 구성됩니다.

Guard DSL을 사용하여 작성된 규칙을 파일 확장명을 사용하는 일반 텍스트 파일에 저장할 수 있습니다.

여러 규칙 파일을 생성하고 규칙 세트로 분류할 수 있습니다. 규칙 세트를 사용하면 여러 규칙 파일에 대해 JSON 또는 YAML 형식의 데이터를 동시에 검증할 수 있습니다.

주제

- 절
- 절에서 쿼리 사용
- 절에서 연산자 사용
- 절에서 사용자 지정 메시지 사용
- 절 결합
- Guard 규칙과 함께 블록 사용
- 내장 함수 사용
- Guard 쿼리 및 필터링 정의
- Guard 규칙에서 변수 할당 및 참조
- 에서 명명된 규칙 블록 구성 AWS CloudFormation Guard
- 컨텍스트 인식 평가를 수행하기 위한 절 작성

절

절은 true(PASS) 또는 false()로 평가되는 부울 표현식입니다FAIL. 절은 이진 연산자를 사용하여 단일 값에서 작동하는 두 값 또는 단항 연산자를 비교합니다.

단항 절의 예

다음 단항 절은 컬렉션TcpBlockedPorts이 비어 있는지 여부를 평가합니다.

```
InputParameters.TcpBlockedPorts not empty
```

다음 단항 절은 ExecutionRoleArn 속성이 문자열인지 여부를 평가합니다.

```
Properties.ExecutionRoleArn is_string
```

바이너리 절의 예

다음 바이너리 절은 대/소문자에 encrypted관계없이 BucketName 속성에 문자열이 포함되어 있는지 여부를 평가합니다.

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

다음 바이너리 절은 ReadCapacityUnits 속성이 5,000보다 작거나 같은지 여부를 평가합니다.

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

Guard 규칙 절 작성을 위한 구문

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

Guard 규칙 절의 속성

query

계층적 데이터를 통과하도록 작성된 점(.)으로 구분된 표현식입니다. 쿼리 표현식에는 값의 하위 집합을 대상으로 하는 필터 표현식이 포함될 수 있습니다. 쿼리를 변수에 할당하여 변수를 한 번 작성하고 규칙 세트의 다른 곳에서 참조할 수 있으므로 쿼리 결과에 액세스할 수 있습니다.

쿼리 작성 및 필터링에 대한 자세한 내용은 섹션을 참조하세요쿼리 정의 및 필터링.

필수 항목 여부: 예

operator

쿼리의 상태를 확인하는 데 도움이 되는 단항 또는 이진 연산자입니다. 이진 연산자의 왼쪽(LHS)은 쿼리여야 하며 오른쪽(RHS)은 쿼리 또는 값 리터럴이어야 합니다.

지원되는 이진 연산자: == (같음) | != (같지 않음) | > (크거나 같음) | >= (<작음) | <= (작거나 같음) | IN ([x, y, z] 형식의 목록에서

지원되는 단항 연산자: exists | empty | is_string | is_list | | is_struct | not(!)

필수 항목 여부: 예

query|value literal

쿼리 또는 string 또는와 같이 지원되는 값 리터럴입니다integer(64).

지원되는 값 리터럴:

- 모든 기본 유형: `string`, `integer(64)`, `float(64)`, `bool`, `char`, `regex`
- 다음과 같이 표현되는 `integer(64)`, `float(64)`또는 범위를 표현하기 위한 모든 특수 `char` 범위 유형:
  - `r[<lower_limit>, <upper_limit>]`- 다음 표현식을 `k` 충족하는 모든 값으로 변환됩니다. `lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`는 다음 표현식을 `k` 충족하는 모든 값으로 변환됩니다. `lower_limit <= k < upper_limit`
  - `r(<lower_limit>, <upper_limit>]`- 다음 표현식을 `k` 충족하는 모든 값으로 변환됩니다. `lower_limit < k <= upper_limit`
  - `r(<lower_limit>, <upper_limit>)`, 다음 표현식을 `k` 충족하는 모든 값으로 변환됩니다. `lower_limit < k < upper_limit`
- 중첩된 키-값 구조 데이터에 대한 연결 배열(맵)입니다. 예제:

  ```
  { "my-map": { "nested-maps": [ { "key": 10, "value": 20 } ] } }
  ```
- 프리미티브 유형 또는 연결 배열 유형의 배열

필수: 조건부, 이진 연산자를 사용할 때 필요합니다.

`custom message`

절에 대한 정보를 제공하는 문자열입니다. 메시지는 `validate` 및 `test` 명령의 상세 출력에 표시되며 계층적 데이터에 대한 규칙 평가를 이해하거나 디버깅하는 데 유용할 수 있습니다.

필수 항목 여부: 아니요

절에서 쿼리 사용

쿼리 작성에 대한 자세한 내용은 쿼리 정의 및 필터링 및 섹션을 참조하세요Guard 규칙에서 변수 할당 및 참조.

절에서 연산자 사용

다음은 CloudFormation 템플릿 `Template-1` 및의 예입니다`Template-2`. 지원되는 연산자의 사용을 보여주기 위해이 섹션의 예제 쿼리 및 절은 이러한 예제 템플릿을 참조합니다.

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
```

단항 연산자를 사용하는 절의 예

- `empty` - 컬렉션이 비어 있는지 확인합니다. 쿼리를 사용하면 컬렉션이 생성되므로 쿼리에 계층적 데이터의 값이 있는지 확인할 수도 있습니다. 이를 사용하여 문자열 값 쿼리에 빈 문자열("")이 정의되어 있는지 확인할 수 없습니다. 자세한 내용은 쿼리 정의 및 필터링 단원을 참조하십시오.

  다음 절은 템플릿에 정의된 리소스가 하나 이상 있는지 확인합니다. 논리적 ID가 인 리소스가에 정의되어 PASS 있으므로 로 평가`S3Bucket`됩니다`Template-1`.

```
Resources !empty
```

다음 절은 S3Bucket 리소스에 하나 이상의 태그가 정의되어 있는지 확인합니다. 에 Tags 속성에 대해 정의된 두 개의 태그S3Bucket가 PASS 있기 때문에 로 평가됩니다Template-1.

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

- exists - 쿼리의 각 발생에 값이 있고 대신 사용할 수 있는지 확인합니다!= null.

  다음 절은 BucketEncryption 속성이에 정의되어 있는지 확인합니다S3Bucket. 에서에 대해 BucketEncryption가 정의PASS되어 있기 때문에 S3Bucket로 평가됩니다Template-1.

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

> **Note**
>
> empty 및 not exists 검사는 입력 데이터를 통과할 때 누락된 속성 키가 true 있는지 로 평가합니다. 예를 들어 Properties 섹션이의 템플릿에 정의되지 않은 경우 절S3Bucket은 로 Resources.S3Bucket.Properties.Tag empty 평가됩니다true. exists 및 empty 검사는 오류 메시지의 문서 내에 JSON 포인터 경로를 표시하지 않습니다. 이러한 두 절 모두 이러한 순회 정보를 유지하지 않는 검색 오류가 있는 경우가 많습니다.

- is_string - 쿼리의 각 발생이 string 유형인지 확인합니다.

  다음 절은 S3Bucket 리소스의 BucketName 속성에 문자열 값이 지정되었는지 확인합니다. 문자열 값은에서에 대해 지정PASS되므로 BucketName로 평가"MyServiceS3Bucket"됩니다Template-1.

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

- is_list - 쿼리의 각 발생이 list 유형인지 확인합니다.

  다음 절은 S3Bucket 리소스의 Tags 속성에 목록이 지정되어 있는지 확인합니다. 에서에 대해 두 개의 키-값 페어가 지정PASS되므로 Tags로 평가됩니다Template-1.

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

- `is_struct` - 쿼리의 각 발생이 구조화된 데이터인지 확인합니다.

  다음 절은 S3Bucket 리소스의 BucketEncryption 속성에 구조화된 데이터가 지정되어 있는지 확인합니다. 에서 ServerSideEncryptionConfiguration 속성 유형*(##)*을 사용하여 BucketEncryption를 지정PASS하므로 로 평가됩니다Template-1.

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

> **Note**
>
> 역방향 상태를 확인하려면 (`not !`) 연산자를 , `is_string` `is_list`및 `is_struct` 연산자와 함께 사용할 수 있습니다.

이진 연산자를 사용하는 절의 예

다음 절은 대/소문자에 encrypt관계없이에서 S3Bucket 리소스의 BucketName 속성에 지정된 값에 문자열이 Template-1 포함되어 있는지 확인합니다. 지정된 버킷 이름에 문자열이 포함되어 "MyServiceS3Bucket" 있지 PASS 않기 때문에 로 평가됩니다encrypt.

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

다음 절은의 NewVolume 리소스 Size 속성에 지정된 값이 50 <= Size <= 200의 특정 범위 Template-2 내에 있는지 확인합니다. 에가 지정되어 있기 PASS 때문에 로 평가100됩니다Size.

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

다음 절은에서 NewVolume 리소스의 VolumeType 속성에 지정된 값이 Template-2 io1, io2또는 인지 확인합니다gp3. 에가 지정되어 있기 PASS 때문에 로 평가io1됩니다NewVolume.

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

> **Note**
>
> 이 섹션의 예제 쿼리는 논리적 IDs S3Bucket 및가 있는 리소스를 사용하는 연산자의 사용을 보여줍니다NewVolume. 리소스 이름은 종종 사용자 정의이며 코드형 인프라(IaC) 템플

릿에서 임의로 명명될 수 있습니다. 일반적이고 템플릿에 정의된 모든 AWS::S3::Bucket 리소스에 적용되는 규칙을 작성하려면 가장 일반적으로 사용되는 쿼리 형식은 입니다Resources.*[Type == 'AWS::S3::Bucket']. 자세한 내용은 쿼리 정의 및 필터링에서 사용에 대한 세부 정보를 확인하고 cloudformation-guard GitHub 리포지토리의 예제 디렉터리를 살펴보세요.

절에서 사용자 지정 메시지 사용

다음 예제에서의 절에는 사용자 지정 메시지가 Template-2 포함됩니다.

```
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>>
```

절 결합

Guard에서 새 줄에 작성된 각 절은 (부울 and 로직)을 사용하여 암시적으로 다음 절과 결합됩니다. 다음 예제를 참조하세요.

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

또한 분리를 사용하여 첫 번째 절의 or|OR 끝에를 지정하여 절을 다음 절과 결합할 수 있습니다.

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

Guard 절에서는 분리가 먼저 평가되고 그 다음에는 연결이 평가됩니다. 가드 규칙은 (or|OR) 또는 true ()로 평가되는 절(and|AND의 PASS)의 분리와 함께 정의할 수 있습니다falseFAIL. 이는 보조 정상 형식과 유사합니다.

다음 예제에서는 절의 평가 순서를 보여줍니다.

```
# (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
```

예제를 기반으로 하는 모든 절은 함께 사용하여 결합할 Template-1 수 있습니다. 다음 예제를 참조하세요.

```
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
```

Guard 규칙과 함께 블록 사용

블록은 일련의 관련 절, 조건 또는 규칙에서 세부 정보 및 반복을 제거하는 구성입니다. 블록에는 세 가지 유형이 있습니다.

- 쿼리 블록
- when 블록
- 명명된 규칙 블록

쿼리 블록

다음은 예제를 기반으로 하는 절입니다Template-1. 연결은 절을 결합하는 데 사용되었습니다.

```
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
```

각 절의 쿼리 표현식 일부가 반복됩니다. 쿼리 블록을 사용하여 구성 가능성을 개선하고 초기 쿼리 경로가 동일한 관련 절 집합에서 세부 정보 및 반복을 제거할 수 있습니다. 다음 예제와 같이 동일한 절 집합을 작성할 수 있습니다.

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

쿼리 블록에서 블록 앞의 쿼리는 블록 내 절의 컨텍스트를 설정합니다.

블록 사용에 대한 자세한 내용은 섹션을 참조하세요명명된 규칙 블록 구성.

when 블록

다음 형식의 블록을 사용하여 조건부로 when 블록을 평가할 수 있습니다.

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

when 키워드는 when 블록의 시작을 지정합니다. condition는 가드 규칙입니다. 블록은 조건 평가 결과 true ()인 경우에만 평가됩니다PASS.

다음은를 기반으로 하는 when 블록의 예입니다Template-1.

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

```
}
```

when 블록 내의 절은에 지정된 값이 문자열인 경우에만 평가BucketName됩니다. 다음 예제와 같이 템플릿의 Parameters 섹션에서에 지정된 값이 참조BucketName되는 경우 when 블록 내의 절은 평가되지 않습니다.

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

명명된 규칙 블록

규칙 세트(규칙 세트)에 이름을 할당한 다음 다른 규칙에서 명명된 규칙 블록이라고 하는 이러한 모듈식 검증 블록을 참조할 수 있습니다. 명명된 규칙 블록의 형식은 다음과 같습니다.

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

rule 키워드는 명명된 규칙 블록의 시작을 지정합니다.

rule name는 명명된 규칙 블록을 고유하게 식별하는 사람이 읽을 수 있는 문자열입니다. 캡슐화하는 Guard 규칙 세트의 레이블입니다. 이 사용에서 Guard 규칙이라는 용어에는 절, 쿼리 블록, when 블록 및 명명된 규칙 블록이 포함됩니다. 규칙 이름은 캡슐화된 규칙 세트의 평가 결과를 참조하는 데 사용할 수 있으며, 이를 통해 명명된 규칙 블록을 재사용할 수 있습니다. 규칙 이름은 validate 및 test 명령 출력의 규칙 실패에 대한 컨텍스트도 제공합니다. 규칙 이름은 규칙 파일의 평가 출력에 블록의 평가 상태(PASS, FAIL또는 SKIP)와 함께 표시됩니다. 다음 예제를 참조하세요.

```
# 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 키워드 뒤에 규칙 이름 뒤에 조건을 지정하여 명명된 규칙 블록을 조건부로 평가할 수도 있습니다.

다음은이 주제에서 앞서 설명한 예제 when 블록입니다.

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

명명된 규칙 블록을 사용하여 앞의 내용을 다음과 같이 작성할 수도 있습니다.

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

다른 Guard 규칙과 함께 명명된 규칙 블록을 재사용하고 그룹화할 수 있습니다. 다음은 몇 가지 예입니다.

```
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
    ...
}
```

내장 함수 사용

AWS CloudFormation Guard 는 규칙에서 문자열 조작, JSON 구문 분석 및 데이터 유형 변환과 같은 작업을 수행하는 데 사용할 수 있는 내장 함수를 제공합니다. 함수는 변수에 대한 할당을 통해서만 지원됩니다.

주요 함수

`json_parse(json_string)`

템플릿의 인라인 JSON 문자열을 구문 분석합니다. 구문 분석 후 결과 객체의 속성을 평가할 수 있습니다.

`count(collection)`

쿼리가 해결하는 항목 수를 반환합니다.

`regex_replace(base_string, regex_to_extract, regex_replacement)`

정규식을 사용하여 문자열의 일부를 대체합니다.

문자열 조작, 수집 작업 및 데이터 유형 변환 함수를 포함하여 사용 가능한 함수의 전체 목록은 Guard GitHub 리포지토리의 함수 설명서를 참조하세요.

Guard 쿼리 및 필터링 정의

이 주제에서는 Guard 규칙 절을 작성할 때 쿼리 작성 및 필터링 사용에 대해 설명합니다.

사전 조건

필터링은 고급 AWS CloudFormation Guard 개념입니다. 필터링에 대해 알아보기 전에 다음 기본 주제를 검토하는 것이 좋습니다.

- AWS CloudFormation Guard란 무엇인가요?
- 규칙, 절 작성

쿼리 정의

쿼리 표현식은 계층적 데이터를 통과하기 위해 작성된 간단한 점(.)으로 구분된 표현식입니다. 쿼리 표현식에는 값의 하위 집합을 대상으로 하는 필터 표현식이 포함될 수 있습니다. 쿼리가 평가되면 SQL 쿼리에서 반환된 결과 집합과 유사한 값 모음이 생성됩니다.

다음 예제 쿼리는 CloudFormation 템플릿에서 `AWS::IAM::Role` 리소스를 검색합니다.

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

쿼리는 다음과 같은 기본 원칙을 따릅니다.

- 쿼리의 각 점(.) 부분은 `Resources` 또는 쿼리의 일부가 수신 기준과 일치하지 않는 `Properties.Encrypted`. 경우 Guard는 검색 오류를 발생시키는 등 명시적 키 용어를 사용할 때 계층 구조를 통과합니다.
- 와일드카드를 사용하는 쿼리의 점(.) 부분은 해당 수준에서 구조의 모든 값을 * 통과합니다.
- 배열 와일드카드를 사용하는 쿼리의 점(.) 부분은 해당 배열의 모든 인덱스를 `[*]` 통과합니다.
- 대괄호 안에 필터를 지정하여 모든 컬렉션을 필터링할 수 있습니다`[]`. 다음과 같은 방법으로 컬렉션을 찾을 수 있습니다.
  - 데이터에서 자연적으로 발생하는 배열은 컬렉션입니다. 다음은 예제입니다.

    포트: `[20, 21, 110, 190]`

    태그: `[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - 와 같은 구조의 모든 값을 통과하는 경우 `Resources.*`
  - 쿼리 결과 자체는 값을 추가로 필터링할 수 있는 모음입니다. 다음 예제를 참조하세요.

    ```
    # 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
}
```

다음은 CloudFormation 템플릿 코드 조각의 예입니다.

```
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
    ...
```

이 템플릿을 기반으로 통과한 경로는 SampleRole이고 선택한 최종 값은 입니다Type: AWS::IAM::Role.

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

Resources.*[Type == 'AWS::IAM::Role'] YAML 형식의 쿼리 결과 값은 다음 예제에 나와 있습니다.

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

쿼리를 사용할 수 있는 몇 가지 방법은 다음과 같습니다.

- 변수를 참조하여 쿼리 결과에 액세스할 수 있도록 변수에 쿼리를 할당합니다.
- 선택한 각 값에 대해 테스트하는 블록이 있는 쿼리를 따릅니다.
- 쿼리를 기본 절과 직접 비교합니다.

변수에 쿼리 할당

Guard는 지정된 범위 내에서 원샷 변수 할당을 지원합니다. Guard 규칙의 변수에 대한 자세한 내용은 섹션을 참조하세요Guard 규칙에서 변수 할당 및 참조.

쿼리를 변수에 할당하여 쿼리를 한 번 작성한 다음 Guard 규칙의 다른 곳에서 참조할 수 있습니다. 이 섹션의 뒷부분에서 설명하는 쿼리 원칙을 보여주는 다음 예제 변수 할당을 참조하세요.

```
#
# 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'
    }
]
```

쿼리에 할당된 변수의 값을 직접 반복

Guard는 쿼리의 결과에 대해 직접 실행을 지원합니다. 다음 예제에서 when 블록은 CloudFormation 템플릿에 있는 각 AWS::EC2::Volume 리소스의 EncryptedVolumeType, 및 AvailabilityZone 속성에 대해 테스트합니다.

```
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']
        }
    }
}
```

직접 절 수준 비교

또한 Guard는 직접 비교의 일부로 쿼리를 지원합니다. 예를 들어 다음을 참조하십시오.

```
let resources = Resources.*

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

앞의 예에서 표시된 형식으로 표현된 두 절(some키워드로 시작)은 독립 절로 간주되며 별도로 평가됩니다.

단일 절 및 블록 절 양식

위 섹션에 표시된 두 가지 예제 절은 모두 다음 블록과 동일하지 않습니다.

```
let resources = Resources.*

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

이 블록은 컬렉션의 각 Tag 값을 쿼리하고 속성 값을 예상 속성 값과 비교합니다. 이전 섹션의 결합된 절 형식은 두 절을 독립적으로 평가합니다. 다음 입력을 고려하세요.

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

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

첫 번째 형식의 절은 로 평가됩니다PASS. 첫 번째 절을 첫 번째 형식으로 검증할 때 , Tags, 및 Resources Properties의 다음 경로는 값과 Key 일치NotPRODEnd하고 예상 값과 일치하지 않습니다PROD.

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

첫 번째 양식의 두 번째 절에서도 마찬가지입니다. Resources, PropertiesTags, 및의 경로는 값과 Value 일치합니다AppStart. 따라서 두 번째 절은 독립적으로 적용됩니다.

전체 결과는 입니다PASS.

그러나 블록 형식은 다음과 같이 평가됩니다. 각 Tags 값에 대해 Key 및가 모두 일치하는지 비교Value합니다. 다음 예제에서는 NotAppStart 및 NotPRODEnd 값이 일치하지 않습니다.

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

평가는 Key == /PROD$/, 및 모두에 대해 확인되므로 Value == /^App/일치가 완료되지 않습니다. 따라서 결과는 입니다FAIL.

> **Note**
>
> 컬렉션 작업 시 컬렉션의 각 요소에 대해 여러 값을 비교하려는 경우 블록 절 양식을 사용하는 것이 좋습니다. 컬렉션이 스칼라 값 집합이거나 단일 속성만 비교하려는 경우 단일 절 양식을 사용합니다.

쿼리 결과 및 관련 절

모든 쿼리는 값 목록을 반환합니다. 누락된 키, 모든 인덱스에 액세스할 때 배열의 빈 값(`Tags: []`) 또는 빈 맵()이 발생할 때 맵의 누락된 값 등 순회 중 어떤 부분이든 검색 오류가 발생할 `Resources: {}`수 있습니다.

이러한 쿼리에 대해 절을 평가할 때 모든 검색 오류는 실패로 간주됩니다. 유일한 예외는 쿼리에 명시적 필터가 사용되는 경우입니다. 필터를 사용하면 연결된 절을 건너뜁니다.

다음 블록 실패는 실행 중인 쿼리와 연결됩니다.

- 템플릿에 리소스가 포함되어 있지 않은 경우 쿼리는 로 평가`FAIL`되고 연결된 블록 수준 절도 로 평가됩니다`FAIL`.
- 템플릿에와 같은 빈 리소스 블록이 포함된 경우 쿼리`{ "Resources": {} }`는 로 평가`FAIL`되고 연결된 블록 수준 절도 로 평가됩니다`FAIL`.
- 템플릿에 리소스가 포함되어 있지만 쿼리와 일치하는 리소스가 없는 경우 쿼리는 빈 결과를 반환하고 블록 수준 절은 건너뜁니다.

쿼리에서 필터 사용

쿼리의 필터는 선택 기준으로 사용되는 효과적인 Guard 절입니다. 절의 구조는 다음과 같습니다.

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

필터를 사용할 AWS CloudFormation Guard 규칙 작성 때의 다음 주요 사항에 유의하세요.

- CNF(Conjunctive Normal Form)를 사용하여 절을 결합합니다.
- 새 줄에 각 연결(and) 절을 지정합니다.
- 두 절 사이에 `or` 키워드를 사용하여 분리(or)를 지정합니다.

다음 예제에서는 결합 및 분리 절을 보여줍니다.

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

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

선택 기준에 절 사용

모든 컬렉션에 필터링을 적용할 수 있습니다. 필터링은 이미와 같은 컬렉션인 입력의 속성에 직접 적용할 수 있습니다`securityGroups: [....]`. 항상 값 모음인 쿼리에 필터링을 적용할 수도 있습니다. 결합 노멀 형식을 포함한 절의 모든 기능을 필터링에 사용할 수 있습니다.

CloudFormation 템플릿에서 유형별로 리소스를 선택할 때 다음과 같은 일반적인 쿼리가 자주 사용됩니다.

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

쿼리는 입력의 `Resources` 섹션에 있는 모든 값을 `Resources.*` 반환합니다. 의 예제 템플릿 입력의 경우 쿼리쿼리 정의는 다음을 반환합니다.

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

이제이 컬렉션에 필터를 적용합니다. 일치시킬 기준은 입니다`Type == AWS::IAM::Role`. 다음은 필터를 적용한 후 쿼리의 출력입니다.

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

그런 다음 `AWS::IAM::Role` 리소스에 대한 다양한 절을 확인합니다.

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

다음은 모든 AWS::IAM::Policy 및 AWS::IAM::ManagedPolicy 리소스를 선택하는 필터링 쿼리의 예입니다.

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

다음 예제에서는 이러한 정책 리소스에가 PolicyDocument 지정되어 있는지 확인합니다.

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

더 복잡한 필터링 요구 사항 구축

수신 및 송신 보안 그룹 정보에 대한 AWS Config 구성 항목의 다음 예를 고려합니다.

```
---
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
```

다음 사항에 유의하세요.

- `ipPermissions` (수신 규칙)는 구성 블록 내의 규칙 모음입니다.
- 각 규칙 구조에는 `ipv4Ranges` 및와 같은 속성이 `ipv6Ranges` 포함되어 CIDR 블록 컬렉션을 지정합니다.

IP 주소로부터의 연결을 허용하는 수신 규칙을 선택하고 규칙이 TCP 차단 포트의 노출을 허용하지 않는지 확인하는 규칙을 작성해 보겠습니다.

다음 예제와 같이 IPv4를 포함하는 쿼리 부분으로 시작합니다.

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

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

some 키워드는이 컨텍스트에서 유용합니다. 모든 쿼리는 쿼리와 일치하는 값 모음을 반환합니다. 기본적으로 Guard는 쿼리의 결과로 반환된 모든 값이 검사와 일치하는지 평가합니다. 그러나이 동작이 항상 검사에 필요한 것은 아닐 수 있습니다. 구성 항목의 입력에서 다음 부분을 고려합니다.

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

에는 두 가지 값이 있습니다ipv4Ranges. 모든 ipv4Ranges 값이 로 표시된 IP 주소와 같은 것은 아닙니다0.0.0.0/0. 하나 이상의 값이와 일치하는지 확인하려고 합니다0.0.0.0/0. 쿼리에서 반환된 모든 결과가 일치해야 하는 것은 아니지만 하나 이상의 결과가 일치해야 한다고 Guard에 알립니다. some 키워드는 Guard에 결과 쿼리의 하나 이상의 값이 검사와 일치하는지 확인하도록 지시합니다. 쿼리 결과 값이 일치하지 않으면 Guard에서 오류가 발생합니다.

그런 다음 다음 예제와 같이 IPv6를 추가합니다.

```
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'
]
```

마지막으로 다음 예제에서는 프로토콜이가 아닌지 확인합니다udp.

```
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' ]
]
```

다음은 전체 규칙입니다.

```
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
                    >>
                }
            }

        }
    }
}
```

포함된 유형을 기반으로 컬렉션 분리

코드형 인프라(IaC) 구성 템플릿을 사용하는 경우 구성 템플릿 내의 다른 엔터티에 대한 참조가 포함된 컬렉션이 발생할 수 있습니다. 다음은에 대한 로컬 참조`TaskRoleArn`,에 대한 참조 `TaskArn`및 직접 문자열 참조를 사용하여 Amazon Elastic Container Service(Amazon ECS) 작업을 설명하는 CloudFormation 템플릿의 예입니다.

```
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'
```

다음과 같은 쿼리를 가정하겠습니다.

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

이 쿼리는 예제 템플릿에 표시된 세 AWS::ECS::TaskDefinition 리소스를 모두 포함하는 값 모음을 반환합니다. 다음 예제와 같이 TaskRoleArn 로컬 참조가 ecs_tasks 포함된를 다른와 구분합니다.

```
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
    >>
}
```

Guard 규칙에서 변수 할당 및 참조

AWS CloudFormation Guard 규칙 파일에 변수를 할당하여 Guard 규칙에서 참조하려는 정보를 저장할 수 있습니다. Guard는 원샷 변수 할당을 지원합니다. 변수는 느리게 평가됩니다. 즉, Guard는 규칙이 실행될 때만 변수를 평가합니다.

주제

- 변수 할당
- 변수 참조
- 변수 범위
- Guard 규칙 파일의 변수 예제

변수 할당

`let` 키워드를 사용하여 변수를 초기화하고 할당합니다. 변수 이름에는 스네이크 케이스를 사용하는 것이 가장 좋습니다. 변수는 쿼리로 인한 정적 리터럴 또는 동적 속성을 저장할 수 있습니다. 다음 예제에서 변수는 정적 문자열 값를 `ecs_task_definition_task_role_arn` 저장합니다`arn:aws:iam:123456789012:role/my-role-name`.

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

다음 예제에서 변수는 템플릿의 모든 `AWS::ECS::TaskDefinition` 리소스를 CloudFormation 검색하는 쿼리의 결과를 `ecs_tasks` 저장합니다. 규칙을 작성할 때 `ecs_tasks`를 참조하여 해당 리소스에 대한 정보에 액세스할 수 있습니다.

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

변수 참조

`%` 접두사를 사용하여 변수를 참조합니다.

의 `ecs_task_definition_task_role_arn` 변수 예제를 기반으로 변수 할당Guard 규칙 절의 `ecs_task_definition_task_role_arn` `query|value literal` 섹션에서를 참조할 수 있습니다. 이 참조를 사용하면 CloudFormation 템플릿에 있는 `AWS::ECS::TaskDefinition` 리소스의 `TaskDefinitionArn` 속성에 지정된 값이 정적 문자열 값가 됩니다`arn:aws:iam:123456789012:role/my-role-name`.

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

의 ecs_tasks 변수 예제를 기반으로 쿼리ecs_tasks에서를 참조변수 할당할 수 있습니다(예: %ecs_tasks.Properties). 먼저 Guard는 변수를 평가한 ecs_tasks 다음 반환된 값을 사용하여 계층 구조를 통과합니다. 변수가 문자열이 아닌 값으로 ecs_tasks 확인되면 Guard에서 오류가 발생합니다.

> **Note**
>
> 현재 Guard는 사용자 지정 오류 메시지 내에서 변수 참조를 지원하지 않습니다.

변수 범위

범위는 규칙 파일에 정의된 변수의 가시성을 나타냅니다. 변수 이름은 범위 내에서 한 번만 사용할 수 있습니다. 변수를 선언할 수 있는 세 가지 수준 또는 세 가지 가능한 변수 범위가 있습니다.

- 파일 수준 - 일반적으로 규칙 파일 상단에 선언되며 규칙 파일 내의 모든 규칙에서 파일 수준 변수를 사용할 수 있습니다. 전체 파일에 표시됩니다.

  다음 예제 규칙 파일에서 변수 ecs_task_definition_task_role_arn 및 ecs_task_definition_execution_role_arn는 파일 수준에서 초기화됩니다.

  ```
  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
  }
  ```

- 규칙 수준 - 규칙 내에서 선언된 규칙 수준 변수는 해당 특정 규칙에만 표시됩니다. 규칙 외부에서 참조하면 오류가 발생합니다.

다음 예제 규칙 파일에서 변수 ecs_task_definition_task_role_arn 및 ecs_task_definition_execution_role_arn는 규칙 수준에서 초기화됩니다. 는 check_ecs_task_definition_task_role_arn명명된 규칙 내에서만 참조할 ecs_task_definition_task_role_arn 수 있습니다. check_ecs_task_definition_execution_role_arn 명명된 규칙 내에서만 ecs_task_definition_execution_role_arn 변수를 참조할 수 있습니다.

```
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
}
```

- 블록 수준 - when 절과 같은 블록 내에서 선언된 블록 수준 변수는 해당 특정 블록에만 표시됩니다. 블록 외부의 모든 참조로 인해 오류가 발생합니다.

  다음 예제 규칙 파일에서 변수 ecs_task_definition_task_role_arn 및 ecs_task_definition_execution_role_arn는 AWS::ECS::TaskDefinition 유형 블록 내의 블록 수준에서 초기화됩니다. 각 규칙의 AWS::ECS::TaskDefinition 유형 블록 내에서만 ecs_task_definition_task_role_arn 및 ecs_task_definition_execution_role_arn 변수를 참조할 수 있습니다.

```
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 규칙 파일의 변수 예제

다음 섹션에서는 변수의 정적 할당과 동적 할당의 예를 제공합니다.

정적 할당

다음은 CloudFormation 템플릿의 예입니다.

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

이 템플릿을 기반으로 모든 `AWS::ECS::TaskDefinition` 템플릿 리소스의 `TaskRoleArn` 속성이 인지 `check_ecs_task_definition_task_role_arn` 확인하는 라는 규칙을 작성할 수 있습니다`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
}
```

규칙 범위 내에서 라는 변수를 초기화`ecs_task_definition_task_role_arn`하고 정적 문자열 값를 할당할 수 있습니다`'arn:aws:iam::123456789012:role/my-role-name'`. 규칙 절은 `query|value literal` 섹션의 `ecs_task_definition_task_role_arn` 변수를 `arn:aws:iam::123456789012:role/my-role-name` 참조하여 `EcsTask` 리소스의 `TaskRoleArn` 속성에 지정된 값이 인지 확인합니다.

동적 할당

다음은 CloudFormation 템플릿의 예입니다.

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

이 템플릿을 기반으로 파일 범위 ecs_tasks 내에서 라는 변수를 초기화하고 쿼리를 할 당할 수 있습니다Resources.*[Type == 'AWS::ECS::TaskDefinition'. Guard 는 입력 템플릿의 모든 리소스를 쿼리하고 이에 대한 정보를에 저장합니다ecs_tasks. 또한 모든 AWS::ECS::TaskDefinition 템플릿 리소스의 TaskRoleArn 속성check_ecs_task_definition_task_role_arn이 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'
}
```

규칙 절은 query 섹션의 ecs_task_definition_task_role_arn 변수를 arn:aws:iam::123456789012:role/my-role-name 참조하여 EcsTask 리소스의 TaskRoleArn 속성에 지정된 값이 인지 확인합니다.

CloudFormation 템플릿 구성 적용

프로덕션 사용 사례의 보다 복잡한 예를 살펴보겠습니다. 이 예제에서는 Amazon ECS 태스크 정의 방법에 대한 보다 엄격한 제어를 보장하기 위해 Guard 규칙을 작성합니다.

다음은 CloudFormation 템플릿의 예입니다.

```
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'
```

이 템플릿을 기반으로 다음 규칙을 작성하여 이러한 요구 사항이 충족되도록 합니다.

- 템플릿의 각 `AWS::ECS::TaskDefinition` 리소스에는 태스크 역할과 실행 역할이 모두 연결되어 있습니다.
- 작업 역할과 실행 역할은 AWS Identity and Access Management (IAM) 역할입니다.
- 역할은 템플릿에 정의되어 있습니다.
- `PermissionsBoundary` 속성은 각 역할에 대해 지정됩니다.

```
# 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
    }
}
```

에서 명명된 규칙 블록 구성 AWS CloudFormation Guard

를 사용하여 명명된 규칙 블록을 작성할 때 다음 두 가지 구성 스타일을 사용할 AWS CloudFormation Guard수 있습니다.

- 조건부 종속성
- 상관관계 종속성

이러한 종속성 구성 스타일 중 하나를 사용하면 재사용 가능성을 높이고 명명된 규칙 블록의 세부 정보 및 반복을 줄일 수 있습니다.

주제

- 사전 조건
- 조건부 종속성 구성
- 상관관계 종속성 구성

사전 조건

쓰기 규칙의 명명된 규칙 블록에 대해 알아봅니다.

조건부 종속성 구성

이러한 구성 스타일에서 when 블록 또는 명명된 규칙 블록의 평가는 하나 이상의 다른 명명된 규칙 블록 또는 절의 평가 결과에 조건부로 종속됩니다. 다음 예제 Guard 규칙 파일에는 조건부 종속성을 보여주는 명명된 규칙 블록이 포함되어 있습니다.

```
# 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
    ...
}
```

앞의 예제 규칙 파일에서 Example-1의 가능한 결과는 다음과 같습니다.

- 가 로 rule_name_A 평가되면 PASS에 의해 캡슐화된 Guard 규칙이 평가rule_name_B됩니다.
- 가 로 rule_name_A 평가되면 FAIL에 의해 캡슐화된 Guard 규칙rule_name_B은 평가되지 않습니다.는 로 rule_name_B 평가됩니다SKIP.
- 가 로 rule_name_A 평가되면 SKIP에 의해 캡슐화된 Guard 규칙rule_name_B은 평가되지 않습니다.는 로 rule_name_B 평가됩니다SKIP.

> **Note**
>
> 이 사례는가 로 평가FAIL되고가 로 rule_name_A 평가되는 규칙에 rule_name_A 조건부로 종속되는 경우에 발생합니다SKIP.

다음은 수신 및 송신 보안 그룹 정보를 위한 항목의 구성 관리 데이터베이스(CMDB) 구성 AWS Config 항목의 예입니다. 이 예제에서는 조건부 종속성 구성을 보여줍니다.

```
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
                }
            }
        }
    }
}
```

앞의 예에서 check_parameter_validity는에 조건부로 종속check_resource_type_and_parameter되고 check_ip_procotol_and_port_range_validity는에 조건부로 종속됩니다check_parameter_validity. 다음은 이전 규칙을 준수하는 구성 관리 데이터베이스(CMDB) 구성 항목입니다.

```
---
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
```

상관관계 종속성 구성

이러한 구성 스타일에서 when 블록 또는 명명된 규칙 블록의 평가는 하나 이상의 다른 Guard 규칙의 평가 결과에 상관 관계가 있습니다. 다음과 같이 상관관계 종속성을 달성할 수 있습니다.

```
# 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
    ...
}
```

상관관계 종속성 구성을 이해하는 데 도움이 되도록 Guard 규칙 파일의 다음 예제를 검토하세요.

```
#
# 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'
}
```

앞의 규칙 파일에서 ensure_elbs_are_internal_and_secure는에 대한 상관관계 종속성이 있습니다ensure_all_elbs_are_secure. 다음은 이전 규칙을 준수하는 CloudFormation 템플릿의 예입니다.

```
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...'
```

컨텍스트 인식 평가를 수행하기 위한 절 작성

AWS CloudFormation Guard 절은 계층적 데이터에 대해 평가됩니다. Guard 평가 엔진은 간단한 점 표기법을 사용하여 지정된 계층적 데이터를 따라 수신 데이터에 대한 쿼리를 해결합니다. 데이터 맵 또는 컬렉션과 비교하여 평가하려면 여러 절이 필요한 경우가 많습니다. Guard는 이러한 절을 작성할 수 있는 편리한 구문을 제공합니다. 엔진은 상황에 맞게 인식되며 평가에 연결된 해당 데이터를 사용합니다.

다음은 컨텍스트 인식 평가를 적용할 수 있는 컨테이너가 있는 Kubernetes 포드 구성의 예입니다.

```
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 절을 작성하여이 데이터를 평가할 수 있습니다. 규칙 파일을 평가할 때 컨텍스트는 전체 입력 문서입니다. 다음은 포드에 지정된 컨테이너에 대한 제한 적용을 검증하는 예제 절입니다.

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

평가**context**의 이해

규칙 블록 수준에서 수신 컨텍스트는 전체 문서입니다. when 조건에 대한 평가는 apiVersion 및 kind 속성이 있는이 수신 루트 컨텍스트에 대해 수행됩니다. 이전 예제에서 이러한 조건은 로 평가됩니다true.

이제 이전 예제에 spec.containers[*] 표시된의 계층 구조를 통과합니다. 계층 구조의 각 트래버스에 대해 컨텍스트 값이 그에 따라 변경됩니다. spec 블록의 순회가 완료되면 다음 예제와 같이 컨텍스트가 변경됩니다.

```
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 속성을 통과한 후 다음 예제에 컨텍스트가 표시됩니다.

```
- 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
```

루프 이해

표현[*]식을 사용하여 containers 속성의 배열에 포함된 모든 값에 대한 루프를 정의할 수 있습니다. 블록은 내부의 각 요소에 대해 평가됩니다containers. 앞의 예제 규칙 코드 조각에서 블록 내에 포함된 절은 컨테이너 정의에 대해 검증할 검사를 정의합니다. 내부에 포함된 절 블록은 각 컨테이너 정의에 대해 한 번씩 두 번 평가됩니다.

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

각 반복에 대해 컨텍스트 값은 해당 인덱스의 값입니다.

> **Note**
>
> 지원되는 유일한 인덱스 액세스 형식은 `[<integer>]` 또는 입니다`[*]`. 현재 Guard는와 같은 범위를 지원하지 않습니다`[2..4]`.

배열

배열이 허용되는 곳에서는 단일 값도 허용되는 경우가 많습니다. 예를 들어 컨테이너가 하나만 있는 경우 배열을 삭제할 수 있으며 다음 입력이 허용됩니다.

```
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
```

속성이 배열을 수락할 수 있는 경우 규칙이 배열 양식을 사용하는지 확인합니다. 이전 예제에서는 `containers[*]`가 아닌를 사용합니다`containers`. Guard는 단일 값 형식만 발견하면 데이터를 통과할 때 올바르게 평가됩니다.

> **Note**
>
> 속성이 배열을 수락할 때 규칙 절에 대한 액세스를 표현할 때는 항상 배열 양식을 사용합니다. Guard는 단일 값이 사용되는 경우에도 올바르게 평가됩니다.

spec.containers[*] 대신 양식 사용 spec.containers

가드 쿼리는 해결된 값 모음을 반환합니다. 양식을 사용할 때 쿼리spec.containers의 확인된 값에는 안에 있는 요소가 containers아니라에서 참조하는 배열이 포함됩니다. 양식을 사용할 때 포함된 각 개별 요소를 spec.containers[*]참조합니다. 배열에 포함된 각 요소를 평가하려는 경우 항상 [*] 양식을 사용해야 합니다.

this를 사용하여 현재 컨텍스트 값 참조

Guard 규칙을 작성할 때를 사용하여 컨텍스트 값을 참조할 수 있습니다this. this는 컨텍스트의 값에 바인딩되기 때문에 암시적인 경우가 많습니다. 예를 들어 , this.kind및 this.apiVersionthis.spec는 루트 또는 문서에 바인딩됩니다. 반대로 this.resources는 / spec/containers/0/ 및 containers와 같은의 각 값에 바인딩됩니다/spec/containers/1. 마찬가지로 this.cpu 및는 제한, 특히 /spec/containers/0/resources/limits 및에 this.memory 매핑됩니다/spec/containers/1/resources/limits.

다음 예제에서는 Kubernetes 포드 구성에 대한 이전 규칙을 this 명시적으로 사용하도록 다시 작성합니다.

```
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
```

```
            >>
        }
    }
}
```

를 this 명시적으로 사용할 필요는 없습니다. 그러나 다음 예제와 같이 스칼라로 작업할 때 this 참조가 유용할 수 있습니다.

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

이전 예제에서 this는 각 포트 번호를 참조하는 데 사용됩니다.

암시적 사용에 따른 잠재적 오류 **this**

규칙 및 절을 작성할 때 암시적 this 컨텍스트 값의 요소를 참조할 때 몇 가지 일반적인 실수가 있습니다. 예를 들어 평가할 다음 입력 데이텀을 고려합니다(반드시 통과해야 함).

```
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
```

이전 템플릿에 대해 테스트할 때 다음 규칙은 암시적를 활용한다는 잘못된 가정을 하기 때문에 오류가 발생합니다this.

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

이 예제를 살펴보려면 앞의 규칙 파일을 이름으로 저장any_ip_ingress_check.guard하고 데이터를 파일 이름으로 저장합니다ip_ingress.yaml. 그런 다음 이러한 파일을 사용하여 다음 validate 명령을 실행합니다.

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

다음 출력에서 엔진은 값에 InputParameters.TcpBlockedPorts[*] 대한 속성 검색 시도가 /configuration/ipPermissions/0 /configuration/ipPermissions/1 실패했음을 나타냅니다.

```
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[*]
```

이 결과를 이해하는 데 도움이 되도록 this 명시적으로 참조된를 사용하여 규칙을 다시 작성합니다.

```
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.InputParameters는 변수에 포함된 각 값을 참조합니다any_ip_permissions. 변수에 할당된 쿼리는 일치하는 configuration.ipPermissions 값을 선택합니다. 오류는이 컨텍스트InputParamaters에서를 검색하려는 시도를 나타내지만 루트 컨텍스트에 InputParameters 있었습니다.

내부 블록은 다음 예제와 같이 범위를 벗어난 변수도 참조합니다.

```
{
    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
        >>
    }
}
```

this는의 각 포트 값을 참조하지만[21, 22, 110], fromPort 및 도 참조합니다toPort. 둘 다 외부 블록 범위에 속합니다.

의 암시적 사용에 따른 오류 해결 **this**

변수를 사용하여 값을 명시적으로 할당하고 참조합니다. 먼저 InputParameter.TcpBlockedPorts는 입력(루트) 컨텍스트의 일부입니다. 다음 예제와 같이 내부 블록InputParameter.TcpBlockedPorts에서 벗어나 명시적으로 할당합니다.

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

그런 다음이 변수를 명시적으로 참조하세요.

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

내에서 내부 this 참조에 대해 동일한 작업을 수행합니다%ports.

하지만 내부의 루프에 ports 여전히 잘못된 참조가 있기 때문에 모든 오류가 아직 수정되지는 않습니다. 다음 예제에서는 잘못된 참조의 제거를 보여줍니다.

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

그런 다음 validate 명령을 다시 실행합니다. 이번에는 통과합니다.

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

다음은 validate 명령의 출력입니다.

```
ip_ingress.yaml Status = PASS
PASS rules
check_ip_procotol_and_port_range_validity    PASS
```

이 접근 방식에 장애가 있는지 테스트하기 위해 다음 예제에서는 페이로드 변경을 사용합니다.

```
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
```

90은 IPv6 주소가 허용되는 89~109 범위 내에 있습니다. 다음은 `validate` 명령을 다시 실행한 후의 출력입니다.

```
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 규칙 테스트

기본 제공 단위 테스트 프레임워크를 AWS CloudFormation Guard 사용하여 Guard 규칙이 의도한 대로 작동하는지 확인할 수 있습니다. 이 섹션에서는 단위 테스트 파일을 작성하는 방법과 `test` 명령을 사용하여 규칙 파일을 테스트하는 데 사용하는 방법을 안내합니다.

단위 테스트 파일에는 `.json`, , , `.JSON`, 또는 확장명 중 하나가 있어야 합니다`.jsn.yaml.YAML.yml`.

주제

- 사전 조건
- Guard 유닛 테스트 파일 개요
- Guard 규칙 단위 테스트 파일 작성 연습

사전 조건

입력 데이터를 평가할 Guard 규칙을 작성합니다. 자세한 내용은 Guard 규칙 작성 단원을 참조하십시오.

Guard 유닛 테스트 파일 개요

가드 단위 테스트 파일은 여러 입력과 Guard 규칙 파일 내에 작성된 규칙에 대한 예상 결과를 포함하는 JSON 또는 YAML 형식의 파일입니다. 다양한 기대치를 평가하기 위한 여러 샘플이 있을 수 있습니다. 먼저 빈 입력을 테스트한 다음 다양한 규칙 및 절을 평가하기 위한 정보를 점진적으로 추가하는 것이 좋습니다.

또한 접미사 `_test.json` 또는를 사용하여 단위 테스트 파일의 이름을 지정하는 것이 좋습니다`_tests.yaml`. 예를 들어 라는 규칙 파일이 있는 경우 단위 테스트 파일 `my_rules.guard`이름을로 지정합니다`my_rules_tests.yaml`.

구문

다음은 단위 테스트 파일의 구문을 YAML 형식으로 보여줍니다.

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

속성

다음은 Guard 테스트 파일의 속성입니다.

`input`

규칙을 테스트할 데이터입니다. 다음 예제와 같이 첫 번째 테스트에서 빈 입력을 사용하는 것이 좋습니다.

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

후속 테스트의 경우 테스트할 입력 데이터를 추가합니다.

필수 항목 여부: 예

`expectations`

특정 규칙이 입력 데이터에 대해 평가될 때 예상되는 결과입니다. 각 규칙에 대한 예상 결과 외에도 테스트하려는 하나 이상의 규칙을 지정합니다. 예상 결과는 다음 중 하나여야 합니다.

- `PASS` - 입력 데이터에 대해 실행하면 규칙이 로 평가됩니다`true`.
- `FAIL` - 입력 데이터에 대해 실행하면 규칙이 로 평가됩니다`false`.
- `SKIP` - 입력 데이터에 대해 실행하면 규칙이 트리거되지 않습니다.

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

필수 항목 여부: 예

Guard 규칙 단위 테스트 파일 작성 연습

다음은 라는 규칙 파일입니다`api_gateway_private.guard`. 이 규칙의 목적은 CloudFormation 템플릿에 정의된 모든 Amazon API Gateway 리소스 유형이 프라이빗 액세스용으로만 배포되었는지 확인하는 것입니다. 또한 하나 이상의 정책 문이 Virtual Private Cloud(VPC)에서 액세스를 허용하는지 여부를 확인합니다.

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

이 연습에서는 첫 번째 규칙 의도를 테스트합니다. 배포된 모든 AWS::ApiGateway::RestApi 리소스는 프라이빗이어야 합니다.

1. 다음과 같은 초기 테스트api_gateway_private_tests.yaml가 포함된 라는 단위 테스트 파일을 생성합니다. 초기 테스트에서는 빈 입력을 추가하고 입력으로 AWS::ApiGateway::RestApi 리소스가 없기 때문에 규칙이 건너뛸 것으로 예상check_rest_api_is_private합니다.

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

```
      check_rest_api_is_private: SKIP
```

2. test 명령을 사용하여 터미널에서 첫 번째 테스트를 실행합니다. --rules-file 파라미터에 규칙 파일을 지정합니다. --test-data 파라미터에 단위 테스트 파일을 지정합니다.

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

첫 번째 테스트의 결과는 입니다PASS.

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

3. 단위 테스트 파일에 다른 테스트를 추가합니다. 이제 빈 리소스를 포함하도록 테스트를 확장합니다. 다음은 업데이트된 api_gateway_private_tests.yaml 파일입니다.

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

4. 업데이트된 단위 테스트 파일로 test를 실행합니다.

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

두 번째 테스트의 결과는 입니다PASS.

```
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. 단위 테스트 파일에 두 개의 테스트를 추가합니다. 다음을 포함하도록 테스트를 확장합니다.

   - 속성이 지정되지 않은 AWS::ApiGateway::RestApi 리소스입니다.

     > **Note**
     >
     > 이는 유효한 CloudFormation 템플릿은 아니지만 잘못된 형식의 입력에 대해서도 규칙이 올바르게 작동하는지 테스트하는 것이 유용합니다.

     EndpointConfiguration 속성이 지정되지 않아 로 설정되지 않았으므로이 테스트가 실패할 것으로 예상됩니다PRIVATE.

   - EndpointConfiguration 속성이 로 설정된 첫 번째 의도를 충족하지PRIVATE만 정책 문이 정의되지 않았기 때문에 두 번째 의도를 충족하지 않는 AWS::ApiGateway::RestApi 리소스입니다. 이 테스트가 통과할 것으로 예상합니다.

   다음은 업데이트된 단위 테스트 파일입니다.

```
---
- 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. 업데이트된 단위 테스트 파일로 test를 실행합니다.

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

세 번째 결과는 이고 FAIL네 번째 결과는 입니다PASS.

```
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. 단위 테스트 파일에 테스트 1~3을 주석 처리합니다. 네 번째 테스트에 대해서만 상세 컨텍스트에 액세스합니다. 다음은 업데이트된 단위 테스트 파일입니다.

```
---
#- 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. `--verbose` 플래그를 사용하여 터미널에서 `test` 명령을 실행하여 평가 결과를 검사합니다. 상세 컨텍스트는 평가를 이해하는 데 유용합니다. 이 경우 네 번째 테스트가 PASS 결과로 성공한 이유에 대한 자세한 정보를 제공합니다.

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

다음은 해당 실행의 출력입니다.

```
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)
```

출력의 주요 관측값은 검사 통과를 `Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`나타내는 줄입니다. 이 예제에서는가 배열일 것으로 Types 예상되지만 단일 값이 제공된 사례도 보여주었습니다. 이 경우 Guard는 계속 평가하여 올바른 결과를 제공했습니다.

9. `EndpointConfiguration` 속성이 지정된 `AWS::ApiGateway::RestApi` 리소스의 단위 테스트 파일에 네 번째 테스트 사례와 같은 테스트 사례를 추가합니다. 테스트 케이스는 통과 대신 실패합니다. 다음은 업데이트된 단위 테스트 파일입니다.

```
---
#- 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 플래그를 사용하여 업데이트된 단위 테스트 파일로 test 명령을 실행합니다.

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

는에 대해 지정EndpointConfiguration되지만 REGIONAL는 예상되지 않기 때문에 결과는 예상FAIL대로입니다.

```
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 명령의 상세 출력은 규칙 파일의 구조를 따릅니다. 규칙 파일의 모든 블록은 상세 출력의 블록입니다. 최상위 블록은 각 규칙입니다. 규칙에 대한 when 조건이 있는 경우 형제 조건 블록에 표시됩니다. 다음 예제에서는 조건을 %api_gws !empty 테스트하고 통과합니다.

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

조건이 통과되면 규칙 절을 테스트합니다.

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

%api_gws는 출력의 BlockClause 레벨(line:21)에 해당하는 블록 규칙입니다. 규칙 절은 연결(AND) 절 집합이며, 여기서 각 연결 절은 분리(ORs) 집합입니다. 이 연결에는 단일 절인가 있습니다Properties.EndpointConfiguration.Types[*] == "PRIVATE". 따라서 상세 출력에는 단일 절이 표시됩니다. 경로는 입력의 어떤 값이 비교되는지 /Resources/apiGw/Properties/EndpointConfiguration/Types/1 보여줍니다.이 경우는 1로 인Types덱싱된의 요소입니다.

에서는이 섹션의 예제를 사용하여 validate 명령을 사용하여 규칙에 대해 입력 데이터를 평가할 Guard 규칙에 대한 입력 데이터 검증수 있습니다.

AWS CloudFormation Guard 규칙과 함께 입력 파라미터 사용

AWS CloudFormation Guard 를 사용하면 검증 중에 동적 데이터 조회에 입력 파라미터를 사용할 수 있습니다. 이 기능은 규칙에서 외부 데이터를 참조해야 할 때 특히 유용합니다. 그러나 입력 파라미터 키를 지정할 때 Guard에서는 충돌하는 경로가 없어야 합니다.

사용 방법

1. `--input-parameters` 또는 `-i` 플래그를 사용하여 입력 파라미터가 포함된 파일을 지정합니다. 여러 입력 파라미터 파일을 지정할 수 있으며 결합되어 공통 컨텍스트를 형성합니다. 입력 파라미터 키에는 충돌하는 경로가 있을 수 없습니다.
2. `--data` 또는 `-d` 플래그를 사용하여 검증할 실제 템플릿 파일을 지정합니다.

사용 예시

1. 입력 파라미터 파일(예: `network.yaml`)을 생성합니다.

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

2. 가드 규칙 파일에서 다음 파라미터를 참조합니다(예: `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. 실패한 데이터 템플릿(예: `security_groups_fail.yaml`)을 생성합니다.

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

Resources:
```

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

4. validate 명령을 실행합니다.

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

이 명령에서:

- `-r`는 규칙 파일을 지정합니다.
- `-i`는 입력 파라미터 파일을 지정합니다.
- `-d`는 검증할 데이터 파일(템플릿)을 지정합니다.

다중 입력 파라미터

여러 입력 파라미터 파일을 지정할 수 있습니다.

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

로 지정된 모든 파일이 결합되어 파라미터 조회를 위한 단일 컨텍스트를 형성`-i`합니다.

AWS CloudFormation Guard 규칙에 대한 입력 데이터 검증

명령을 AWS CloudFormation Guard `validate` 사용하여 Guard 규칙에 대한 데이터를 검증할 수 있습니다. 파라미터 및 옵션을 포함하여 `validate` 명령에 대한 자세한 내용은 검증을 참조하세요.

사전 조건

- 입력 데이터를 검증할 Guard 규칙을 작성합니다. 자세한 내용은 Guard 규칙 작성 단원을 참조하십시오.
- 규칙을 테스트하여 의도한 대로 작동하는지 확인합니다. 자세한 내용은 Guard 규칙 테스트 단원을 참조하십시오.

validate 명령 사용

AWS CloudFormation 템플릿과 같은 Guard 규칙에 대해 입력 데이터를 검증하려면 Guard validate 명령을 실행합니다. --rules 파라미터에 규칙 파일의 이름을 지정합니다. --data 파라미터에 입력 데이터 파일의 이름을 지정합니다.

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

Guard가 템플릿을 성공적으로 검증하면 validate 명령은 종료 상태 0 ($? bash)를 반환합니다. Guard가 규칙 위반을 식별하면 validate 명령은 실패한 규칙의 상태 보고서를 반환합니다. 요약 플래그(-s all)를 사용하여 Guard가 각 규칙을 평가한 방법을 보여주는 세부 평가 트리를 확인합니다.

```
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]
```

여러 데이터 파일에 대해 여러 규칙 검증

규칙을 유지 관리하는 데 도움이 되도록 규칙을 여러 파일에 작성하고 원하는 대로 규칙을 구성할 수 있습니다. 그런 다음 데이터 파일 또는 여러 데이터 파일에 대해 여러 규칙 파일을 검증할 수 있습니다. `validate` 명령은 `--data` 및 `--rules` 옵션에 대한 파일 디렉터리를 가져올 수 있습니다. 예를 들어 가 하나 이상의 데이터 파일을 `/path/to/dataDirectory` 포함하고가 하나 이상의 규칙 파일을 `/path/to/ruleDirectory` 포함하는 다음 명령을 실행할 수 있습니다.

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

여러 CloudFormation 템플릿에 정의된 다양한 리소스에 저장 시 암호화를 보장하는 적절한 속성 할당이 있는지 확인하는 규칙을 작성할 수 있습니다. 검색 및 유지 관리를 쉽게 하기 위해 경로가 인 디렉터리에서 , `s3_bucket_encryption.guard` `ec2_volume_encryption.guard`및 라는 별도의 파일에 있는 각 리소스`rds_dbinstance_encrytion.guard`의 저장 시 암호화를 확인하는 규칙을 가질 수 있습니다`~/GuardRules/encryption_at_rest`. 검증해야 하는 CloudFormation 템플릿은 경로가 인 디렉터리에 있습니다`~/CloudFormation/templates`. 이 경우 다음과 같이 `validate` 명령을 실행합니다.

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

문제 해결 AWS CloudFormation Guard

작업 중에 문제가 발생하면이 섹션의 주제를 AWS CloudFormation Guard참조하세요.

주제

- 선택한 유형의 리소스가 없는 경우 절이 실패합니다.
- Guard는 짧은 형식 Fn::GetAtt 참조로 CloudFormation 템플릿을 평가하지 않습니다.
- 일반 문제 해결 주제

선택한 유형의 리소스가 없는 경우 절이 실패합니다.

쿼리가와 같은 필터를 사용하는 경우 입력에 `AWS::ApiGateway::RestApi` 리소스가 없는 `Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`경우 절은 로 평가됩니다`FAIL`.

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

이러한 결과를 방지하려면 변수에 필터를 할당하고 `when` 조건 확인을 사용합니다.

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

Guard는 짧은 형식 Fn::GetAtt 참조로 CloudFormation 템플릿을 평가하지 않습니다.

Guard는 짧은 형태의 내장 함수를 지원하지 않습니다. 예를 들어 YAML 형식 CloudFormation 템플릿`!Join!Sub`에서를 사용하는 것은 지원되지 않습니다. 대신 확장된 형태의 CloudFormation 내장 함수를 사용합니다. 예를 들어 `Fn::Join`Guard 규칙에 대해 평가할 때 YAML 형식의 CloudFormation 템플릿`Fn::Sub`에서를 사용합니다.

내장 함수에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 내장 함수 참조를 참조하세요.

일반 문제 해결 주제

- `string` 리터럴에 이스케이프 처리된 임베디드 문자열이 포함되어 있지 않은지 확인합니다. Guard는 `string` 리터럴의 임베디드 이스케이프 문자열을 지원하지 않습니다. 인라인 JSON 문자열을 구

문 분석하려는 경우 Guard 3.0.0 이상에서 사용할 수 있는 `json_parse()` 함수를 사용합니다. 자세한 내용은 내장 함수 사용 단원을 참조하십시오.

- `!=` 비교가 호환되는 데이터 유형을 비교하는지 확인합니다. 예를 들어 `string` 및 `int`는 비교를 위해 호환되지 않는 데이터 형식입니다. `!=` 비교를 수행할 때 값이 호환되지 않으면 내부적으로 오류가 발생합니다. 현재 오류는 Rust의 PartialEq 특성을 충족하기 `false` 위해 억제되고 로 변환됩니다.

AWS CloudFormation Guard CLI 파라미터 및 명령 참조

다음 전역 파라미터 및 명령은 AWS CloudFormation Guard 명령줄 인터페이스(CLI)를 통해 사용할 수 있습니다.

주제

- Guard CLI 전역 파라미터
- 구문 분석 트리
- 규칙 생성
- 테스트
- 검증

Guard CLI 전역 파라미터

AWS CloudFormation Guard CLI 명령과 함께 다음 파라미터를 사용할 수 있습니다.

`-h, --help`

도움말 정보를 인쇄합니다.

`-V, --version`

버전 정보를 인쇄합니다.

구문 분석 트리

AWS CloudFormation Guard 규칙 파일에 정의된 규칙에 대한 구문 분석 트리를 생성합니다.

구문

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

파라미터

`-h, --help`

도움말 정보를 인쇄합니다.

`-p, --print-json`

출력을 JSON 형식으로 인쇄합니다.

`-y, --print-yaml`

출력을 YAML 형식으로 인쇄합니다.

`-V, --version`

버전 정보를 인쇄합니다.

옵션

`-o, --output`

생성된 트리를 출력 파일에 씁니다.

`-r, --rules`

규칙 파일을 제공합니다.

예제

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

규칙 생성

JSON 또는 YAML 형식의 AWS CloudFormation 템플릿 파일을 가져와 템플릿 리소스의 속성과 일치하는 AWS CloudFormation Guard 규칙 세트를 자동으로 생성합니다. 이 명령은 규칙 작성을 시작하거나 알려진 정상 템플릿에서 ready-to-use 수 있는 규칙을 생성하는 유용한 방법입니다.

구문

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

파라미터

`-h, --help`

도움말 정보를 인쇄합니다.

`-V, --version`

버전 정보를 인쇄합니다.

옵션

`-o, --output`

생성된 규칙을 출력 파일에 씁니다. 수백 또는 수천 개의 규칙이 나타날 수 있으므로이 옵션을 사용하는 것이 좋습니다.

`-t, --template`

CloudFormation 템플릿 파일의 경로를 JSON 또는 YAML 형식으로 제공합니다.

예제

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

테스트

개별 AWS CloudFormation Guard 규칙의 성공을 확인하기 위해 JSON 또는 YAML 형식의 Guard 유닛 테스트 파일과 비교하여 규칙 파일을 검증합니다.

구문

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

파라미터

`-a, --alphabetical`

디렉터리 내에서 알파벳순으로 정렬합니다.

`-h, --help`

도움말 정보를 인쇄합니다.

`-m, --last-modified`

디렉터리 내에서 마지막으로 수정된 시간을 기준으로 정렬

`-V, --version`

버전 정보를 인쇄합니다.

`-v, --verbose`

출력 세부 정보를 높입니다. 여러 번 지정할 수 있습니다.

상세 출력은 Guard 규칙 파일의 구조를 따릅니다. 규칙 파일의 모든 블록은 상세 출력의 블록입니다. 최상위 블록은 각 규칙입니다. 규칙에 대한 `when` 조건이 있는 경우 형제 조건 블록으로 표시됩니다.

옵션

`-d, --dir`

규칙의 루트 디렉터리를 제공합니다.

`-o, --output-format`

출력을 표시할 형식을 지정합니다.

기본값: `single-line-summary`

허용된 값: `json` | `yaml` | `single-line-summary` | `junit`

`-r, --rules-file`

규칙 파일의 이름을 제공합니다.

`-t, --test-data`

JSON 또는 YAML 형식으로 데이터 파일의 파일 또는 디렉터리 이름을 제공합니다.

예제

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

출력

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

다음 사항도 참조하세요.

Guard 규칙 테스트

검증

AWS CloudFormation Guard 규칙과 비교하여 데이터를 검증하여 성공 또는 실패를 결정합니다.

구문

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

파라미터

`-a, --alphabetical`

사전순으로 정렬된 디렉터리의 파일을 검증합니다.

`-h, --help`

도움말 정보를 인쇄합니다.

`-m, --last-modified`

마지막으로 수정된 시간으로 정렬된 디렉터리의 파일을 검증합니다.

`-P, --payload`

를 통해 규칙과 데이터를 `stdin`다음 JSON 형식으로 제공합니다.

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

예제:

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

"규칙"에서 규칙 파일의 문자열 버전 목록을 지정합니다. "data"에서 데이터 파일의 문자열 버전 목록을 지정합니다.

`--payload`가 지정`--rules`되고 지정할 수 `--data` 없는 경우.

`-p, --print-json`

출력을 JSON 형식으로 인쇄합니다.

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

요약을 포함한 절 실패를 표시합니다.

`-V, --version`

버전 정보를 인쇄합니다.

`-v, --verbose`

출력 세부 정보를 높입니다. 여러 번 지정할 수 있습니다.

`-z, --structured`

구조화되고 유효한 JSON/YAML 목록을 출력합니다. 이 인수는 verbose, print-json, show-summary: all/fail/pass/skip, output-format: single-line-summary 인수와 충돌합니다.

옵션

`-d, --data` (문자열)

JSON 또는 YAML 형식의 데이터 파일 또는 데이터 파일의 디렉터리를 제공합니다. 이 옵션을 반복적으로 사용하여 여러 값을 전달할 수 있도록 지원합니다.

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

data-dir1 위와 같은 디렉터리 인수의 경우 .yaml, .yml, .json, .jsn, .template 확장자가 있는 파일에 대해서만 스캔이 지원됩니다.

`--payload` 플래그를 지정하는 경우 `--data` 옵션을 지정하지 마십시오.

`-i, --input-parameters` (문자열)

결합된 컨텍스트로 사용할 데이터 파일과 함께 사용할 추가 파라미터를 지정하는 파라미터 파일 또는 파라미터 파일의 디렉터리를 JSON 또는 YAML로 제공합니다. 입력으로 전달된 모든 파라미터 파일이 병합되고이 결합된 컨텍스트가에 대한 인수로 전달된 각 파일과 다시 병합됩니다data. 따라서 모든 파일에는 중첩 없이 상호 배타적인 속성이 포함되어야 합니다. 이 옵션을 반복적으로 사용하여 여러 값을 전달할 수 있도록 지원합니다.

디렉터리 인수의 경우 .yaml, .yml, .json, .jsn, .template 확장자가 있는 파일에 대해서만 스캔이 지원됩니다.

`-o, --output-format` (문자열)

출력의 형식을 지정합니다.

기본값: `single-line-summary`

허용된 값: `json` | `yaml` | `single-line-summary` | `junit` | `sarif`

`-r, --rules` (문자열)

규칙 파일 또는 규칙 파일의 디렉터리를 제공합니다. 이 옵션을 반복적으로 사용하여 여러 값을 전달할 수 있도록 지원합니다.

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

rules-dir1 위와 같은 디렉터리 인수의 경우 .guard, .ruleset 확장자가 있는 파일에 대해서만 스캔이 지원됩니다.

--payload 플래그를 지정하는 경우 --rules 옵션을 지정하지 마십시오.

--show-summary(문자열)

요약 테이블을 표시해야 하는지 여부를 제어합니다. --show-summary fail (기본값) 또는 --show-summary pass,fail (통과/실패한 규칙만 표시) 또는 --show-summary none (꺼짐) 또는 --show-summary all (통과, 실패 또는 건너뛰는 모든 규칙을 표시).

기본값: fail

허용된 값: none | all | pass | fail | skip

-t, --type (문자열)

입력 데이터의 형식을 제공합니다. 입력 데이터 유형을 지정하면 Guard는 출력에 CloudFormation 템플릿 리소스의 논리적 이름을 표시합니다. 기본적으로 Guard는와 같은 속성 경로와 값을 표시합니다Property [/Resources/vol2/Properties/Encrypted.

Allowed values: CFNTemplate

예제

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

출력

Guard가 템플릿을 성공적으로 검증하면 validate 명령은 종료 상태 0 ($? bash)를 반환합니다. Guard가 규칙 위반을 식별하면 validate 명령은 실패한 규칙의 상태 보고서를 반환합니다.

```
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".]
```

다음 사항도 참조하세요.

- Guard 규칙에 대한 입력 데이터 검증
- Guard 규칙과 함께 입력 파라미터 사용

의 보안 AWS CloudFormation Guard

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. Guard에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 AWS 제공 범위 내 서비스규정 준수 프로그램 .
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 여러분은 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

다음 설명서는 Guard를 AWS Lambda 함수(cfn-guard-lambda)로 설치할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다.cfn-guard-lambda

- AWS Command Line Interface 사용 설명서의 보안
- AWS Lambda 개발자 안내서의 보안
- AWS Identity and Access Management 사용 설명서의 보안

AWS CloudFormation Guard 문서 기록

다음 표에서는에 대한 설명서 릴리스를 설명합니다 AWS CloudFormation Guard.

- 최종 설명서 업데이트: 2025년 7월 30일
- 최신 버전: 3.1.2

변경 사항	설명	날짜
문서 업데이트	현재 구현에 맞게 Guard CLI 명령 참조 설명서를 업데이트했습니다. 버전 참조가 Guard 3.1.2로 업데이트되었습니다.	2025년 7월 30일
버전 3.0.0 릴리스	버전 3.0.0에는 다음과 같은 개선 사항이 도입되었습니다. • Guard 3.0.0 릴리스에 대한 소개 및 설치 주제가 업데이트되었습니다. • Homebrew 및에 대한 설치 지침이 추가되었습니다Chocolatey. • Guard 버전 3.0.0의 변경 사항을 반영하도록 Guard 규칙 마이그레이션과 관련된 정보가 업데이트되었습니다. • AWS CloudFormation Guard GitHub 리포지토리에 대한 눈에 띄는 링크가 추가되었습니다.	2023년 6월 30일
버전 2.1.3 릴리스	버전 2.1.3에는 다음과 같은 개선 사항이 도입되었습니다.	2023년 6월 9일

	Guard 2.1.3 개선 사항에 대한 정보가 추가되었습니다. Guard 2.0에 대한 참조가 Guard 2.1.3으로 업데이트되었습니다.	
버전 2.0.4 릴리스	버전 2.0.4에는 다음과 같은 개선 사항이 도입되었습니다. `--payload` 플래그가 `validate` 명령에 추가되었습니다. 자세한 내용은 Guard CLI 참조의 validate를 참조하세요.	2021년 10월 19일

버전 2.0.3 릴리스	버전 2.0.3에는 다음과 같은 개선 사항이 도입되었습니다.
	
	• 단위 테스트 파일의 각 테스트에 대한 테스트 이름을 제공할 수 있습니다. 자세한 내용은 Guard 규칙 테스트 단원을 참조하십시오.
	• `validate` 명령에 다음 옵션이 추가되었습니다.
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	
	  자세한 내용은 Guard CLI 참조의 검증을 참조하세요.
		2021년 7월 27일

최초 릴리스	AWS CloudFormation Guard 사용 설명서의 최초 릴리스입니다.	2021년 7월 15일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.