aws

Guida per l'utente

AWS CloudFormation Guard

AWS CloudFormation Guard: Guida per l'utente

Table of Contents

Che cos'è AWS CloudFormation Guard?

AWS CloudFormation Guard è uno strumento di valutazione open source, generico. policy-as-code L'interfaccia a riga di comando Guard (CLI) fornisce un linguaggio simple-to-use dichiarativo specifico del dominio (DSL) che è possibile utilizzare per esprimere le politiche sotto forma di codice. Inoltre, è possibile utilizzare CLI i comandi per convalidare dati o dati strutturati in base a tali regole. JSON YAML Guard fornisce anche un framework di unit testing integrato per verificare che le regole funzionino come previsto.

Guard non convalida i CloudFormation modelli per una sintassi valida o per i valori di proprietà consentiti. È possibile utilizzare lo strumento cfn-lint per eseguire un'ispezione approfondita della struttura del modello.

Guard non fornisce l'applicazione sul lato server. È possibile utilizzare gli CloudFormation Hooks per eseguire la convalida e l'applicazione sul lato server, dove è possibile bloccare o avvisare un'operazione.

Per informazioni dettagliate AWS CloudFormation Guard sullo sviluppo, consulta il repository Guard. GitHub

Argomenti

Sei un utente di Guard per la prima volta?

Se utilizzi Guard per la prima volta, ti consigliamo di iniziare leggendo le seguenti sezioni:

- Configurazione di Guard— Questa sezione descrive come installare Guard. Con Guard, puoi scrivere regole politiche utilizzando Guard DSL e convalidare i tuoi JSON dati YAML strutturati (o formattati) in base a tali regole.
- regole di Writing Guard— Questa sezione fornisce procedure dettagliate per la stesura delle regole politiche.

- Regole di Testing Guard— Questa sezione fornisce una procedura dettagliata per testare le regole per verificare che funzionino come previsto e per convalidare i dati strutturati, o YAML formattati, in base alle JSON regole.
- Convalida dei dati di input rispetto alle regole di Guard— Questa sezione fornisce una procedura dettagliata per la convalida dei dati strutturati, o formattati, in base alle regole JSON stabilite. YAML
- CLIRiferimento Guard— Questa sezione descrive i comandi disponibili in Guard. CLI

Funzionalità di Guard

Utilizzando Guard, puoi scrivere regole politiche per convalidare qualsiasi dato JSON strutturato o YAML formattato, inclusi, a titolo esemplificativo ma non esaustivo, i modelli. AWS CloudFormation Guard supporta l'intero spettro di end-to-end valutazione dei controlli delle politiche. Le regole sono utili nei seguenti settori aziendali:

- Governance e conformità preventive (shift-left testing): convalida le composizioni dell'infrastruttura come codice (IaC) o delle composizioni di infrastrutture e servizi in base alle regole politiche che rappresentano le migliori pratiche organizzative per la sicurezza e la conformità. Ad esempio, puoi convalidare CloudFormation modelli, set di CloudFormation modifiche, file di configurazione Terraform JSON basati su Terraform o configurazioni Kubernetes.
- Detective governance e conformità: convalida la conformità delle risorse del Configuration Management Database (CMDB), come gli elementi di configurazione AWS Config basati (CIs). Ad esempio, gli sviluppatori possono utilizzare Guard Policies contro AWS Config CIs per monitorare continuamente lo stato delle AWS risorse distribuite AWS e non, rilevare le violazioni delle policy e avviare la correzione.
- Sicurezza dell'implementazione: assicurati che le modifiche siano sicure prima della distribuzione. Ad esempio, convalida i set di CloudFormation modifiche in base alle regole delle policy per evitare modifiche che comportino la sostituzione di risorse, come la ridenominazione di una tabella Amazon DynamoDB.

Utilizzo di Guard with Hooks CloudFormation

Puoi usare CloudFormation Guard per creare un Hook in CloudFormation Hooks. CloudFormation Hooks ti consente di applicare in modo proattivo le regole di Guard prima di CloudFormation creare, aggiornare o eliminare operazioni e AWS Cloud Control API creare o aggiornare operazioni. Hooks

garantisce che le configurazioni delle risorse siano conformi alle migliori pratiche di sicurezza, operative e di ottimizzazione dei costi dell'organizzazione.

Per i dettagli su come utilizzare Guard per creare Guard Hooks, consulta le regole di Write CloudFormation Guard per valutare le risorse per Guard Hooks nella Guida per l'utente di Hooks.AWS CloudFormation

Accesso a Guard

Per accedere a Guard DSL e ai comandi, è necessario installare GuardCLI. Per informazioni sull'installazione di GuardCLI, consultaConfigurazione di Guard.

Best practice

Scrivi regole semplici e usa regole denominate per farvi riferimento in altre regole. Le regole complesse possono essere difficili da mantenere e testare.

Configurazione AWS CloudFormation Guard

AWS CloudFormation Guard è un'interfaccia a riga di comando open source (). CLI Fornisce un linguaggio semplice e specifico del dominio per scrivere regole politiche e convalidarne la gerarchia JSON strutturata e i dati rispetto a tali regole. YAML Le regole possono rappresentare linee guida delle politiche aziendali relative alla sicurezza, alla conformità e altro ancora. I dati gerarchici strutturati possono rappresentare l'infrastruttura cloud descritta come codice. Ad esempio, puoi creare regole per garantire che modellino sempre bucket Amazon Simple Storage Service (Amazon S3) crittografati nei loro modelli. CloudFormation

I seguenti argomenti forniscono informazioni su come installare Guard utilizzando il sistema operativo scelto o come AWS Lambda funzione.

Argomenti

Installazione di Guard per Linux e macOS

Puoi eseguire AWS CloudFormation Guard l'installazione per Linux e macOS utilizzando il binario di versione predefinito, Cargo, o tramite Homebrew.

Installa Guard da un binario di rilascio predefinito

Usa la seguente procedura per installare Guard da un binario predefinito.

1. Apri un terminale ed esegui il comando seguente.

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

2. Esegui il comando seguente per impostare la PATH variabile.

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

   Risultati: Hai installato Guard e impostato la PATH variabile con successo.

- (Facoltativo) Per confermare l'installazione di Guard, esegui il seguente comando.

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

Questo comando restituisce il seguente output.

```
cfn-guard 3.0.0
```

Installa Guard from Cargo

Cargo è il gestore di pacchetti Rust. Completa i seguenti passaggi per installare Rust, che include Cargo. Quindi, installa Guard from Cargo.

1. Esegui il seguente comando da un terminale e segui le istruzioni sullo schermo per installare Rust.

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

- (Facoltativo) Per gli ambienti Ubuntu, esegui il seguente comando.

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

2. Configura la variabile di PATH ambiente ed esegui il comando seguente.

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

3. Con Cargo installato, esegui il seguente comando per installare Guard.

```
cargo install cfn-guard
```

Risultati: Guard è stato installato correttamente.

- (Facoltativo) Per confermare l'installazione di Guard, esegui il seguente comando.

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

Questo comando restituisce il seguente output.

```
cfn-guard 3.0.0
```

Installa Guard da Homebrew

Homebrew è un gestore di pacchetti per macOS e Linux. Completa i seguenti passaggi per installare Homebrew. Quindi, installa Guard da Homebrew.

1. Esegui il seguente comando da un terminale e segui le istruzioni sullo schermo per installare Homebrew.

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

2. Con Homebrew installato, esegui il seguente comando per installare Guard.

   ```
   brew install cloudformation-guard
   ```

   Risultati: Guard è stato installato correttamente.

   - (Facoltativo) Per confermare l'installazione di Guard, esegui il seguente comando.

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

     Questo comando restituisce il seguente output.

     ```
     cfn-guard 3.0.0
     ```

Installazione di Guard per Windows

È possibile eseguire l'installazione AWS CloudFormation Guard per Windows tramite Cargo o tramite Chocolatey.

Prerequisiti

Per creare Guard dall'interfaccia a riga di comando, devi installare Build Tools for Visual Studio 2019.

1. Scarica gli strumenti di compilazione di Microsoft Visual C++ dal sito Web Build Tools for Visual Studio 2019.
2. Esegui il programma di installazione e seleziona le impostazioni predefinite.

Installa Guard da Cargo

Cargo è il gestore di pacchetti Rust. Completa i seguenti passaggi per installare Rust, che include Cargo. Quindi, installa Guard from Cargo.

1. Scarica Rust e poi esegui rustup-init.exe.
2. Dal prompt dei comandi, scegli 1, che è l'opzione predefinita.

   Questo comando restituisce il seguente output.

   ```
   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. Per completare l'installazione, premete il tasto Invio.
4. Con Cargo installato, esegui il seguente comando per installare Guard.

   ```
   cargo install cfn-guard
   ```

   Risultati: Guard è stato installato correttamente.

   - (Facoltativo) Per confermare l'installazione di Guard, esegui il seguente comando.

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

     Questo comando restituisce il seguente output.

     ```
     cfn-guard 3.0.0
     ```

Installa Guard da Chocolatey

Chocolatey è un gestore di pacchetti per Windows. Completa i seguenti passaggi per installare Chocolatey. Quindi, installa Guard di Chocolatey.

1. Segui questa guida per installare Chocolatey
2. Con Chocolatey installato, esegui il seguente comando per installare Guard.

   ```
   choco install cloudformation-guard
   ```

   Risultati: Hai installato Guard con successo.

   - (Facoltativo) Per confermare l'installazione di Guard, esegui il seguente comando.

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

     Questo comando restituisce il seguente output.

     ```
     cfn-guard 3.0.0
     ```

Installazione di Guard come AWS Lambda funzione

È possibile eseguire AWS CloudFormation Guard l'installazione tramite Cargo, il gestore di pacchetti Rust. Guard as an AWS Lambda function (`cfn-guard-lambda`) è un wrapper leggero per Guard (`cfn-guard`) che può essere usato come funzione Lambda.

Prerequisiti

Prima di poter installare Guard come funzione Lambda, è necessario soddisfare i seguenti prerequisiti:

- AWS Command Line Interface (AWS CLI) configurato con le autorizzazioni per distribuire e richiamare funzioni Lambda. Per ulteriori informazioni, consultare la pagina relativa alla configurazione di AWS CLI.
- Un ruolo di AWS Lambda esecuzione in (). AWS Identity and Access Management IAM Per ulteriori informazioni, vedere ruolo di AWS Lambda esecuzione.

- Negli RHEL ambienti CentOS/, aggiungi il repository dei `musl-libc` pacchetti alla tua configurazione yum. Per ulteriori informazioni, vedere ngompa/musl-libc.

Installa il gestore di pacchetti Rust

Cargo è il gestore di pacchetti Rust. Completa i seguenti passaggi per installare Rust, che include Cargo.

1. Esegui il seguente comando da un terminale, quindi segui le istruzioni sullo schermo per installare Rust.

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

   - (Facoltativo) Per gli ambienti Ubuntu, esegui il seguente comando.

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

2. Configura la variabile di `PATH` ambiente ed esegui il comando seguente.

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

Installa Guard come funzione Lambda (Linux, macOS o Unix)

Per installare Guard come funzione Lambda, completa i seguenti passaggi.

1. Dal tuo terminale di comando, esegui il comando seguente.

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

   - (Facoltativo) Per confermare l'installazione di Guard come funzione Lambda, esegui il comando seguente.

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

     Questo comando restituisce il seguente output.

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

2. Per installare il `musl` supporto, esegui il comando seguente.

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

3. Crea con`musl`, quindi esegui il comando seguente nel tuo terminale.

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

Per un runtime personalizzato, AWS Lambda richiede un eseguibile con il nome `bootstrap` nel file.zip del pacchetto di distribuzione. Rinomina l'`cfn-lambda`eseguibile generato in `bootstrap` e aggiungilo all'archivio.zip.

- Per gli ambienti macOS, crea il tuo file di configurazione cargo nella radice del progetto Rust o in. `~/.cargo/config`

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

4. Passa alla directory `cfn-guard-lambda` principale.

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

5. Esegui il seguente comando nel tuo terminale.

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

6. Esegui il comando seguente per inviarlo `cfn-guard` come funzione Lambda al tuo account.

```
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
```

Per creare ed eseguire Guard come funzione Lambda

Per richiamare l'invio `cfn-guard-lambda` come funzione Lambda, esegui il comando seguente.

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

Per chiamare la struttura della richiesta della funzione Lambda

Richieste per `cfn-guard-lambda` richiedere i seguenti campi:

- `data`— La versione in formato stringa del JSON modello YAML or
- `rules`— La versione in formato stringa del file del set di regole

Prerequisiti e panoramica per l'utilizzo delle regole di Guard

Questa sezione dimostra come completare le attività principali di Guard, ovvero la scrittura, il test e la convalida delle regole rispetto ai dati in formato JSON o YAML. Inoltre, contiene procedure dettagliate che dimostrano la scrittura di regole che rispondono a casi d'uso specifici.

Argomenti

Prerequisiti

Prima di poter scrivere regole di policy utilizzando il linguaggio DSL (Domain-Specific Language) Guard, è necessario installare l'interfaccia a riga di comando (CLI) di Guard. Per ulteriori informazioni, consulta Configurazione di Guard.

Panoramica sull'utilizzo delle regole di Guard

Quando si utilizza Guard, in genere si eseguono i seguenti passaggi:

1. Scrivi dati in formato JSON o YAML per convalidarli.
2. Regole della politica di Write Guard. Per ulteriori informazioni, consulta regole di Writing Guard.
3. Verifica che le tue regole funzionino come previsto utilizzando il `test` comando Guard. Per ulteriori informazioni sui test unitari, consultaRegole di Testing Guard.
4. Usa il `validate` comando Guard per convalidare i dati in formato JSON o YAML in base alle tue regole. Per ulteriori informazioni, consulta Convalida dei dati di input rispetto alle regole di Guard.

AWS CloudFormation Guard Regole di scrittura

In AWS CloudFormation Guard, le regole sono regole. policy-as-code Scrivi regole nel linguaggio specifico del dominio Guard (DSL) in base alle quali puoi convalidare i tuoi JSON dati (o YAML quelli formattati). Le regole sono costituite da clausole.

È possibile salvare le regole scritte utilizzando Guard DSL in file di testo semplice che utilizzano qualsiasi estensione di file.

È possibile creare più file di regole e classificarli come set di regole. I set di regole consentono di convalidare i dati JSON (o quelli YAML formattati) in base a più file di regole contemporaneamente.

Argomenti

Clausole

Le clausole sono espressioni booleane che restituiscono true () o false ()`PASS`. `FAIL` Le clausole utilizzano operatori binari per confrontare due valori o operatori unari che operano su un singolo valore.

Esempi di clausole unarie

La seguente clausola unaria valuta se la raccolta è vuota. `TcpBlockedPorts`

```
InputParameters.TcpBlockedPorts not empty
```

La seguente clausola unaria valuta se la proprietà è una stringa. `ExecutionRoleArn`

```
Properties.ExecutionRoleArn is_string
```

Esempi di clausole binarie

La clausola binaria seguente valuta se la `BucketName` proprietà contiene la stringa`encrypted`, indipendentemente dal maiuscolo.

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

La clausola binaria seguente valuta se la `ReadCapacityUnits` proprietà è inferiore o uguale a 5.000.

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

Sintassi per scrivere le clausole delle regole di Guard

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

Proprietà delle clausole delle regole Guard

`query`

Un'espressione separata da punti (`.`) scritta per attraversare dati gerarchici. Le espressioni di query possono includere espressioni di filtro destinate a un sottoinsieme di valori. Le query possono essere assegnate alle variabili in modo da poterle scrivere una sola volta e fare riferimento ad esse altrove in un set di regole, il che consentirà di accedere ai risultati delle query.

Per ulteriori informazioni sulla scrittura di interrogazioni e sui filtri, vedere. Definizione di interrogazioni e filtri

Campo obbligatorio: sì

`operator`

Un operatore unario o binario che consente di controllare lo stato della query. Il lato sinistro (LHS) di un operatore binario deve essere una query, mentre il lato destro (RHS) deve essere una query o un valore letterale.

Operatori binari supportati: == (Uguale) | `!` = (Non uguale) | > (Maggiore di) | >= (Maggiore o uguale a) | < (Minore di) | <= (Minore o uguale a) | `IN` (In un elenco di forme [x, y, z]

Operatori unari supportati: `exists` | `empty` | | `is_string` | `is_list` | `is_struct` `not(!)`

Campo obbligatorio: sì

`query|value literal`

Una query o un valore letterale supportato come `string` o. `integer(64)`

Valori letterali supportati:

- Tutti i tipi primitivi:`string`,,`integer(64)`,,`float(64)`, `bool` `char` `regex`
- Tutti i tipi di intervalli specializzati per esprimere `integer(64)` o `char` intervalli espressi come: `float(64)`
  - `r[<lower_limit>, <upper_limit>]`, che si traduce in qualsiasi valore k che soddisfi la seguente espressione: `lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`, che si traduce in qualsiasi valore k che soddisfi la seguente espressione: `lower_limit <= k < upper_limit`
  - `r(<lower_limit>, <upper_limit>]`, che si traduce in qualsiasi valore k che soddisfi la seguente espressione: `lower_limit < k <= upper_limit`
  - `r(<lower_limit>, <upper_limit>)`,che si traduce in qualsiasi valore k che soddisfi la seguente espressione: `lower_limit < k < upper_limit`
- Matrici associative (mappe) per dati di struttura chiave-valore annidati. Per esempio:

  ```
  { "my-map": { "nested-maps": [ { "key": 10, "value": 20 } ] } }
  ```
- Matrici di tipi primitivi o tipi di array associativi

Obbligatorio: condizionale; richiesto quando si utilizza un operatore binario.

`custom message`

Una stringa che fornisce informazioni sulla clausola. Il messaggio viene visualizzato negli output dettagliati dei `test` comandi `validate` and e può essere utile per comprendere o eseguire il debug della valutazione delle regole sui dati gerarchici.

Required: No

Utilizzo di interrogazioni nelle clausole

Per informazioni sulla scrittura di interrogazioni, vedere e. Definizione di interrogazioni e filtri Assegnazione e riferimento a variabili nelle regole di Guard

Utilizzo degli operatori nelle clausole

Di seguito sono riportati CloudFormation modelli di esempio e`Template-1`. `Template-2` Per dimostrare l'uso degli operatori supportati, le query e le clausole di esempio in questa sezione fanno riferimento a questi modelli di esempio.

Modello-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"
```

Modello-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
```

Esempi di clausole che utilizzano operatori unari

- `empty`— Controlla se una raccolta è vuota. È inoltre possibile utilizzarlo per verificare se una query contiene valori in un dato gerarchico, poiché le query generano una raccolta. Non è possibile utilizzarlo per verificare se le query con valori di stringa hanno una stringa vuota () definita. `""` Per ulteriori informazioni, consulta Definizione di interrogazioni e filtri.

  La clausola seguente verifica se il modello ha una o più risorse definite. Viene restituito `PASS` perché una risorsa con l'ID logico `S3Bucket` è definita in. `Template-1`

  ```
  Resources !empty
  ```

  La clausola seguente verifica se uno o più tag sono definiti per la `S3Bucket` risorsa. Il risultato è `PASS` perché `S3Bucket` ha due tag definiti per la `Tags` proprietà in. `Template-1`

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

- `exists`— Verifica se ogni occorrenza dell'interrogazione ha un valore e può essere utilizzata al posto di`!= null`.

  La clausola seguente verifica se la `BucketEncryption` proprietà è definita per. `S3Bucket` Restituisce `PASS` perché `BucketEncryption` è definito per `S3Bucket` in. `Template-1`

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

> **Note**
>
> I `not exists` controlli `empty` e restituiscono le chiavi `true` di proprietà mancanti durante l'attraversamento dei dati di input. Ad esempio, se la `Properties` sezione non è definita nel modello di`S3Bucket`, la clausola restituisce lo stesso risultato`Resources.S3Bucket.Properties.Tag empty`. `true` I `empty` controlli `exists` and non visualizzano il percorso del JSON puntatore all'interno del documento nei messaggi di errore. Entrambe queste clausole spesso presentano errori di recupero che non mantengono queste informazioni trasversali.

- `is_string`— Verifica se ogni occorrenza dell'interrogazione è di tipo. `string`

  La clausola seguente verifica se è specificato un valore di stringa per la `BucketName` proprietà della `S3Bucket` risorsa. Viene restituito `PASS` perché il valore della stringa `"MyServiceS3Bucket"` è specificato per `BucketName` in. `Template-1`

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

- `is_list`— Verifica se ogni occorrenza dell'interrogazione è di `list` tipo.

  La clausola seguente verifica se è specificato un elenco per la `Tags` proprietà della `S3Bucket` risorsa. Viene restituito `PASS` perché per in sono state specificate due coppie chiave-valore. `Tags` `Template-1`

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

- `is_struct`— Verifica se ogni occorrenza della query è costituita da dati strutturati.

  La clausola seguente verifica se i dati strutturati sono specificati per la `BucketEncryption` proprietà della `S3Bucket` risorsa. Viene restituito `PASS` perché `BucketEncryption` è specificato utilizzando il tipo *`(object)`* di `ServerSideEncryptionConfiguration` proprietà in. `Template-1`

> **Note**
>
> Per verificare lo stato inverso, è possibile utilizzare l'operatore (`not !`) con gli operatori `is_string`is_list, `eis_struct`.

Esempi di clausole che utilizzano operatori binari

La clausola seguente verifica se il valore specificato per la `BucketName` proprietà della `S3Bucket` risorsa in `Template-1` contiene la stringa`encrypt`, indipendentemente dal maiuscolo e minuscolo. Il risultato è che il `PASS` nome del bucket specificato `"MyServiceS3Bucket"` non contiene la stringa. `encrypt`

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

La clausola seguente verifica se il valore specificato per la `Size` proprietà della `NewVolume` risorsa in rientra in `Template-2` un intervallo specifico: 50 <= `Size` <= 200. Restituisce PASS perché `100` è specificato per. `Size`

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

La clausola seguente verifica se il valore specificato per la `VolumeType` proprietà della `NewVolume` risorsa in `Template-2` è `io1io2`, o. `gp3` Restituisce PASS perché `io1` è specificato per. `NewVolume`

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

> **Note**
>
> Le query di esempio in questa sezione dimostrano l'uso di operatori che utilizzano le risorse con logico IDs `S3Bucket` and. `NewVolume` I nomi delle risorse sono spesso definiti dall'utente e possono essere denominati arbitrariamente in un modello Infrastructure as Code (IaC). Per scrivere una regola generica e applicabile a tutte le `AWS::S3::Bucket` risorse definite nel modello, la forma di interrogazione più comune utilizzata è. `Resources.*[ Type == 'AWS::S3::Bucket' ]` Per ulteriori informazioni, consulta Definizione di interrogazioni e filtri i dettagli sull'utilizzo ed esplora la directory degli esempi nel `cloudformation-guard` GitHub repository.

Utilizzo di messaggi personalizzati nelle clausole

Nell'esempio seguente, le clausole per `Template-2` includere un messaggio personalizzato.

```
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>>
```

Combinazione di clausole

In Guard, ogni clausola scritta su una nuova riga viene combinata implicitamente con la clausola successiva utilizzando la congiunzione (logica booleana). `and` Guarda l'esempio seguente.

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

È inoltre possibile utilizzare la disgiunzione per combinare una clausola con la clausola successiva specificando alla fine della prima clausola. `or|OR`

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

In una clausola Guard, le disgiunzioni vengono valutate per prime, seguite dalle congiunzioni. Le regole Guard possono essere definite come una combinazione di clausole (e di `or|OR` s) che danno come risultato () o (). `and|AND` `true` `PASS` `false` `FAIL` È simile alla forma normale congiuntiva.

Gli esempi seguenti mostrano l'ordine di valutazione delle clausole.

```
# (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
```

Tutte le clausole basate sull'esempio `Template-1` possono essere combinate utilizzando la congiunzione. Guarda l'esempio seguente.

```
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
```

Utilizzo di blocchi con regole Guard

I blocchi sono composizioni che rimuovono la verbosità e la ripetizione da un insieme di clausole, condizioni o regole correlate. Esistono tre tipi di blocchi:

- Blocchi di query
- `when`blocchi
- blocchi con regole denominate

Blocchi di query

Di seguito sono riportate le clausole basate sull'esempio. `Template-1` La congiunzione è stata utilizzata per combinare le clausole.

```
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
```

Parti dell'espressione di interrogazione in ogni clausola vengono ripetute. È possibile migliorare la componibilità e rimuovere la verbosità e la ripetizione da un set di clausole correlate con lo stesso percorso di interrogazione iniziale utilizzando un blocco di query. È possibile scrivere lo stesso set di clausole come illustrato nell'esempio seguente.

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

```
}
```

In un blocco di query, la query che precede il blocco imposta il contesto per le clausole all'interno del blocco.

Per ulteriori informazioni sull'uso dei blocchi, vedere. Composizione di blocchi con regole denominate

whenblocchi

È possibile valutare i blocchi in modo condizionale utilizzando `when` i blocchi, che assumono la forma seguente.

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

La `when` parola chiave indica l'inizio del `when` blocco. `condition`è una regola della Guardia. Il blocco viene valutato solo se la valutazione della condizione risulta in `true` (PASS).

Di seguito è riportato un esempio di `when` blocco basato su`Template-1`.

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

La clausola all'interno del `when` blocco viene valutata solo se il valore specificato per `BucketName` è una stringa. Se il valore specificato per `BucketName` è referenziato nella `Parameters` sezione del modello, come illustrato nell'esempio seguente, la clausola all'interno del `when` blocco non viene valutata.

```
Parameters:
   S3BucketName:
     Type: String

 Resources:
   S3Bucket:
     Type: "AWS::S3::Bucket"
     Properties:
```

```
      BucketName:
        Ref: S3BucketName
    ...
```

Blocchi con regole denominate

È possibile assegnare un nome a un set di regole (set di regole) e quindi fare riferimento a questi blocchi di convalida modulari, denominati blocchi con regole denominate, in altre regole. I blocchi con regole denominate assumono la forma seguente.

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

La `rule` parola chiave indica l'inizio del blocco di regole con nome.

`rule name`è una stringa leggibile dall'uomo che identifica in modo univoco un blocco di regole con nome. È un'etichetta per il set di regole Guard che incapsula. In questo uso, il termine regola Guard include clausole, blocchi di query, blocchi e blocchi di regole denominate. `when` Il nome della regola può essere usato per fare riferimento al risultato della valutazione del set di regole che incapsula, il che rende riutilizzabili i blocchi con regole denominate. Il nome della regola fornisce inoltre un contesto sugli errori delle regole negli output e dei comandi. `validate` `test` Il nome della regola viene visualizzato insieme allo stato di valutazione del blocco (`PASS`FAIL, o`SKIP`) nell'output di valutazione del file delle regole. Guarda l'esempio seguente.

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

È inoltre possibile valutare i blocchi con regole denominate in modo condizionale specificando la `when` parola chiave seguita da una condizione dopo il nome della regola.

Di seguito è riportato il `when` blocco di esempio discusso in precedenza in questo argomento.

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

Utilizzando blocchi con regole denominate, il precedente può anche essere scritto come segue.

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

Puoi riutilizzare e raggruppare blocchi con regole denominate con altre regole di Guard. Di seguito sono riportati alcuni esempi.

```
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
    ...
```

```
}
```

Definizione delle interrogazioni e del filtraggio di Guard

Questo argomento tratta la scrittura di query e l'uso dei filtri durante la scrittura delle clausole delle regole Guard.

Prerequisiti

Il filtraggio è un concetto avanzato. AWS CloudFormation Guard Ti consigliamo di leggere i seguenti argomenti fondamentali prima di imparare a usare i filtri:

- Che cos'è AWS CloudFormation Guard?
- Regole e clausole di scrittura

Definizione delle interrogazioni

Le espressioni di query sono semplici espressioni separate da punti (`.`) scritte per attraversare dati gerarchici. Le espressioni di query possono includere espressioni di filtro destinate a un sottoinsieme di valori. Quando le query vengono valutate, danno come risultato una raccolta di valori, simile a un set di risultati restituito da una query SQL.

La seguente query di esempio cerca risorse in un AWS CloudFormation modello. `AWS::IAM::Role`

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

Le query seguono questi principi di base:

- Ogni parte dot (`.`) della query attraversa la gerarchia quando viene utilizzato un termine chiave esplicito, ad esempio `Resources` o `Properties.Encrypted`. Se una parte della query non corrisponde al dato in entrata, Guard genera un errore di recupero.
- Una parte dot (`.`) della query che utilizza un carattere jolly `*` attraversa tutti i valori della struttura a quel livello.
- Una parte dot (`.`) della query che utilizza un carattere jolly di matrice `[*]` attraversa tutti gli indici di tale array.
- Tutte le raccolte possono essere filtrate specificando i filtri all'interno di parentesi quadre. `[]` È possibile accedere alle raccolte nei seguenti modi:

- Gli array presenti naturalmente in Datum sono raccolte. Di seguito vengono riportati alcuni esempi relativi ad :

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

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

- Quando si attraversano tutti i valori di una struttura come `Resources.*`
- Qualsiasi risultato della query è di per sé una raccolta da cui i valori possono essere ulteriormente filtrati. Guarda l'esempio seguente.

```
# 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
}
```

Di seguito è riportato un frammento di CloudFormation modello di esempio.

```
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
    ...
```

In base a questo modello, il percorso percorso è `SampleRole` e il valore finale selezionato è. `Type: AWS::IAM::Role`

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

Il valore risultante della query `Resources.*[ Type == 'AWS::IAM::Role' ]` in formato YAML è illustrato nell'esempio seguente.

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

Alcuni dei modi in cui è possibile utilizzare le interrogazioni sono i seguenti:

- Assegnate un'interrogazione alle variabili in modo che sia possibile accedere ai risultati della query facendo riferimento a tali variabili.
- Segui l'interrogazione con un blocco che esegue il test rispetto a ciascuno dei valori selezionati.
- Confronta direttamente una query con una clausola di base.

Assegnazione di interrogazioni alle variabili

Guard supporta l'assegnazione di variabili in un'unica operazione all'interno di un determinato ambito. Per ulteriori informazioni sulle variabili nelle regole di Guard, vedere. Assegnazione e riferimento a variabili nelle regole di Guard

Puoi assegnare le query alle variabili in modo da poter scrivere le query una sola volta e poi farvi riferimento altrove nelle regole di Guard. Vedi il seguente esempio di assegnazione di variabili che illustra i principi di interrogazione discussi più avanti in questa sezione.

```
#
# 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'
    }
]
```

Esame diretto dei valori di una variabile assegnata a una query

Guard supporta l'esecuzione diretta sui risultati di una query. Nell'esempio seguente, il `when` blocco verifica la `AvailabilityZone` proprietà `EncryptedVolumeType`, e per ogni `AWS::EC2::Volume` risorsa trovata in un CloudFormation modello.

```
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']
        }
    }
}
```

Confronti diretti a livello di clausola

Guard supporta anche le interrogazioni come parte dei confronti diretti. Ad esempio, osserva quanto seguente.

```
let resources = Resources.*

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

Nell'esempio precedente, le due clausole (che iniziano con la `some` parola chiave) espresse nel modulo mostrato sono considerate clausole indipendenti e vengono valutate separatamente.

Modulo a clausola singola e clausola a blocchi

Nel loro insieme, le due clausole di esempio mostrate nella sezione precedente non sono equivalenti al blocco seguente.

```
let resources = Resources.*

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

Questo blocco esegue una query per ogni `Tag` valore della raccolta e confronta i valori delle proprietà con i valori delle proprietà previsti. La forma combinata delle clausole nella sezione precedente valuta le due clausole in modo indipendente. Considerate il seguente input.

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

Le clausole nel primo modulo restituiscono a. `PASS` Quando si convalida la prima clausola nella prima forma, il percorso seguente attraversa `Resources` `PropertiesTags`, e `Key` corrisponde al valore `NotPRODEnd` e non corrisponde al valore previsto. `PROD`

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

Lo stesso accade con la seconda clausola del primo modulo. Il percorso attraversa`Resources`, `PropertiesTags`, e `Value` corrisponde al valore`AppStart`. Di conseguenza, la seconda clausola è indipendente.

Il risultato complessivo è un`PASS`.

Tuttavia, il modulo a blocchi viene valutato come segue. Per ogni `Tags` valore, viene confrontato se entrambi gli `Key` e `Value` corrispondono; `NotAppStart` e `NotPRODEnd` i valori non corrispondono nell'esempio seguente.

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

Perché le valutazioni controllano entrambi e `Key == /PROD$/Value == /^App/`, la corrispondenza non è completa. Pertanto, il risultato è`FAIL`.

> **Note**
>
> Quando si lavora con le raccolte, si consiglia di utilizzare il modulo della clausola di blocco quando si desidera confrontare più valori per ogni elemento della raccolta. Utilizzate il modulo a clausola singola quando la raccolta è un insieme di valori scalari o quando intendete confrontare solo un attributo.

Risultati delle query e clausole associate

Tutte le query restituiscono un elenco di valori. Qualsiasi parte di un attraversamento, ad esempio una chiave mancante, valori vuoti per un array (`Tags: []`) quando si accede a tutti gli indici o valori mancanti per una mappa quando si incontra una map vuota (`Resources: {}`), può causare errori di recupero.

Tutti gli errori di recupero sono considerati errori nella valutazione delle clausole rispetto a tali interrogazioni. L'unica eccezione è quando nella query vengono utilizzati filtri espliciti. Quando vengono utilizzati i filtri, le clausole associate vengono ignorate.

I seguenti errori di blocco sono associati all'esecuzione delle query.

- Se un modello non contiene risorse, la query restituisce lo stesso risultato e anche le clausole a `FAIL` livello di blocco associate lo restituiscono. `FAIL`
- Quando un modello contiene un blocco di risorse vuoto, ad `FAIL` esempio`{ "Resources": {} }`, la query restituisce come risultato anche le clausole a livello di blocco associate. `FAIL`
- Se un modello contiene risorse ma nessuna corrisponde alla query, la query restituisce risultati vuoti e le clausole a livello di blocco vengono ignorate.

Utilizzo dei filtri nelle interrogazioni

I filtri nelle query sono effettivamente clausole Guard utilizzate come criteri di selezione. Di seguito è riportata la struttura di una clausola.

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

Tieni presente i seguenti punti chiave relativi AWS CloudFormation Guard Regole di scrittura all'utilizzo dei filtri:

- Combina le clausole utilizzando Conjunctive Normal Form (CNF).
- Specificate ogni clausola di congiunzione (`and`) su una nuova riga.
- Specificate le disgiunzioni (`or`) utilizzando la `or` parola chiave tra due clausole.

L'esempio seguente illustra le clausole congiuntive e disgiuntive.

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

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

Utilizzo delle clausole per i criteri di selezione

È possibile applicare filtri a qualsiasi raccolta. Il filtro può essere applicato direttamente agli attributi dell'input che sono già simili a una raccolta. `securityGroups: [....]` È inoltre possibile applicare il filtro in base a una query, che è sempre una raccolta di valori. È possibile utilizzare tutte le funzionalità delle clausole, inclusa la forma normale congiuntiva, per il filtraggio.

La seguente query comune viene spesso utilizzata quando si selezionano le risorse per tipo da un modello. CloudFormation

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

La query `Resources.*` restituisce tutti i valori presenti nella `Resources` sezione dell'input. Per il modello di esempio inserito inDefinizione delle interrogazioni, la query restituisce quanto segue.

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

Ora, applica il filtro a questa raccolta. Il criterio da abbinare è. `Type == AWS::IAM::Role` Di seguito è riportato l'output della query dopo l'applicazione del filtro.

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

Quindi, controlla le varie clausole relative alle risorse. `AWS::IAM::Role`

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

Di seguito è riportato un esempio di query di filtraggio che seleziona tutte le risorse. `AWS::IAM::Policy` `AWS::IAM::ManagedPolicy`

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

L'esempio seguente verifica se queste risorse politiche hanno un `PolicyDocument` valore specificato.

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

Elaborazione di esigenze di filtraggio più complesse

Considerate il seguente esempio di elemento di AWS Config configurazione per le informazioni sui gruppi di sicurezza in ingresso e in uscita.

```
---
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
```

Tieni presente quanto segue:

- `ipPermissions`(regole di ingresso) è una raccolta di regole all'interno di un blocco di configurazione.
- Ogni struttura di regole contiene attributi come `ipv4Ranges` e `ipv6Ranges` per specificare una raccolta di blocchi CIDR.

Scriviamo una regola che selezioni tutte le regole di ingresso che consentano le connessioni da qualsiasi indirizzo IP e verifichi che le regole non consentano l'esposizione delle porte bloccate dal protocollo TCP.

Iniziamo con la parte di query che copre IPv4, come illustrato nell'esempio seguente.

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

La some parola chiave è utile in questo contesto. Tutte le query restituiscono una raccolta di valori che corrispondono alla query. Per impostazione predefinita, Guard valuta che tutti i valori restituiti come risultato della query vengano confrontati con i controlli. Tuttavia, questo comportamento potrebbe non essere sempre quello necessario per i controlli. Considerate la parte seguente dell'input dell'elemento di configurazione.

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

Sono presenti due valori peripv4Ranges. Non tutti i ipv4Ranges valori equivalgono a un indirizzo IP indicato da0.0.0.0/0. Vuoi vedere se almeno un valore corrisponde0.0.0.0/0. Dici a Guard che non tutti i risultati restituiti da una query devono corrispondere, ma almeno un risultato deve corrispondere. La some parola chiave indica a Guard di assicurarsi che uno o più valori della query risultante corrispondano al controllo. Se nessun valore del risultato della query corrisponde, Guard genera un errore.

Quindi, aggiungi IPv6, come mostrato nell'esempio seguente.

```
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'
]
```

Infine, nell'esempio seguente, verifica che il protocollo non udp lo sia.

```
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' ]
]
```

Di seguito è riportata la regola completa.

```
rule any_ip_ingress_checks
{

    let ports = InputParameter.TcpBlockedPorts[*]

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

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

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

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

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

        }
    }
}
```

Separazione delle raccolte in base ai tipi in cui sono contenute

Quando si utilizzano modelli di configurazione infrastructure as code (IaC), è possibile che si verifichi una raccolta che contiene riferimenti ad altre entità all'interno del modello di configurazione. Di seguito è riportato un CloudFormation modello di esempio che descrive le attività di Amazon Elastic Container Service (Amazon ECS) con un riferimento locale, un riferimento `TaskRoleArn` a e un riferimento `TaskArn` diretto a una stringa.

```
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'
```

Considera la query seguente.

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

Questa query restituisce una raccolta di valori che contiene tutte e tre le `AWS::ECS::TaskDefinition` risorse mostrate nel modello di esempio. Separazioni `ecs_tasks` che contengono riferimenti `TaskRoleArn` locali dalle altre, come illustrato nell'esempio seguente.

```
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
    >>
}
```

Assegnazione e riferimento a variabili nelle regole di Guard

Puoi assegnare variabili nei tuoi file di AWS CloudFormation Guard regole per archiviare le informazioni a cui desideri fare riferimento nelle regole di Guard. Guard supporta l'assegnazione di variabili in un colpo solo. Le variabili vengono valutate pigramente, il che significa che Guard valuta le variabili solo quando vengono eseguite le regole.

Argomenti

Assegnazione di variabili

Utilizzate la `let` parola chiave per inizializzare e assegnare una variabile. Come best practice, utilizzate snake case per i nomi delle variabili. Le variabili possono memorizzare valori letterali statici o proprietà dinamiche risultanti dalle query. Nell'esempio seguente, la variabile `ecs_task_definition_task_role_arn` memorizza il valore della stringa statica. `arn:aws:iam:123456789012:role/my-role-name`

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

Nell'esempio seguente, la variabile `ecs_tasks` memorizza i risultati di una query che cerca tutte le `AWS::ECS::TaskDefinition` risorse in un AWS CloudFormation modello. È possibile fare riferimento `ecs_tasks` all'accesso alle informazioni su tali risorse quando si scrivono le regole.

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

Variabili di riferimento

Utilizzate il `%` prefisso per fare riferimento a una variabile.

In base all'esempio di `ecs_task_definition_task_role_arn` variabile inAssegnazione di variabili, è possibile fare riferimento `ecs_task_definition_task_role_arn` nella

`query|value literal` sezione di una clausola della regola Guard. L'utilizzo di tale riferimento garantisce che il valore specificato per la `TaskDefinitionArn` proprietà di qualsiasi `AWS::ECS::TaskDefinition` risorsa in un CloudFormation modello sia il valore `arn:aws:iam:123456789012:role/my-role-name` della stringa statica.

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

In base all'esempio della `ecs_tasks` variabile in[Assegnazione di variabili](), è possibile fare riferimento `ecs_tasks` in una query (ad esempio, %ECS_Tasks.properties). Innanzitutto, Guard valuta la variabile `ecs_tasks` e quindi utilizza i valori restituiti per attraversare la gerarchia. Se la variabile si `ecs_tasks` risolve in valori non di stringa, Guard genera un errore.

> **Note**
>
> Attualmente, Guard non supporta il riferimento a variabili all'interno di messaggi di errore personalizzati.

Ambito variabile

L'ambito si riferisce alla visibilità delle variabili definite in un file di regole. Un nome di variabile può essere utilizzato solo una volta all'interno di un ambito. Esistono tre livelli in cui è possibile dichiarare una variabile o tre possibili ambiti variabili:

- A livello di file: in genere dichiarate nella parte superiore del file delle regole, è possibile utilizzare variabili a livello di file in tutte le regole all'interno del file delle regole. Sono visibili all'intero file.

  Nel seguente file di regole di esempio, le variabili `ecs_task_definition_task_role_arn` e `ecs_task_definition_execution_role_arn` sono inizializzate a livello di file.

  ```
  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
}
```

- A livello di regola: dichiarate all'interno di una regola, le variabili a livello di regola sono visibili solo per quella regola specifica. Qualsiasi riferimento esterno alla regola genera un errore.

  Nel seguente file di regole di esempio, le variabili `ecs_task_definition_task_role_arn` e `ecs_task_definition_execution_role_arn` vengono inizializzate a livello di regola. `ecs_task_definition_task_role_arn`È possibile fare riferimento solo all'interno della regola denominata. `check_ecs_task_definition_task_role_arn` È possibile fare riferimento solo alla `ecs_task_definition_execution_role_arn` variabile all'interno della regola `check_ecs_task_definition_execution_role_arn` denominata.

  ```
  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
  }
  ```

- A livello di blocco: dichiarate all'interno di un blocco, ad esempio una `when` clausola, le variabili a livello di blocco sono visibili solo per quel blocco specifico. Qualsiasi riferimento esterno al blocco genera un errore.

  Nel seguente file di regole di esempio, le variabili `ecs_task_definition_task_role_arn` e `ecs_task_definition_execution_role_arn` sono inizializzate a livello di blocco all'interno del `AWS::ECS::TaskDefinition` blocco di tipo. È possibile fare riferimento solo alle `ecs_task_definition_execution_role_arn` variabili `ecs_task_definition_task_role_arn` and all'interno dei blocchi `AWS::ECS::TaskDefinition` di tipo per le rispettive regole.

```
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
    }
}
```

Esempi di variabili nei file delle regole di Guard

Le sezioni seguenti forniscono esempi di assegnazione statica e dinamica delle variabili.

Assegnazione statica

Di seguito è riportato un CloudFormation modello di esempio.

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

Sulla base di questo modello, è possibile scrivere una regola denominata `check_ecs_task_definition_task_role_arn` che garantisce che la `TaskRoleArn` proprietà di tutte le risorse del `AWS::ECS::TaskDefinition` modello sia`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
}
```

Nell'ambito della regola, è possibile inizializzare una variabile chiamata `ecs_task_definition_task_role_arn` e assegnarle il valore di stringa statico. `'arn:aws:iam::123456789012:role/my-role-name'` La clausola della regola verifica se il valore specificato per la `TaskRoleArn` proprietà della `EcsTask` risorsa proviene `arn:aws:iam::123456789012:role/my-role-name` dal riferimento alla `ecs_task_definition_task_role_arn` variabile nella sezione. `query|value literal`

Assegnazione dinamica

Di seguito è riportato un CloudFormation modello di esempio.

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

Sulla base di questo modello, è possibile inizializzare una variabile chiamata nell'`ecs_tasks`ambito del file e assegnarle la query. `Resources.*[ Type == 'AWS::ECS::TaskDefinition'` Guard interroga tutte le risorse nel modello di input e memorizza le informazioni su di esse. `ecs_tasks` Puoi anche scrivere una regola chiamata `check_ecs_task_definition_task_role_arn` che assicura che la `TaskRoleArn` proprietà di tutte le risorse del `AWS::ECS::TaskDefinition` modello sia `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'
}
```

La clausola della regola verifica se il valore specificato per la `TaskRoleArn` proprietà della `EcsTask` risorsa proviene `arn:aws:iam::123456789012:role/my-role-name` dal riferimento alla `ecs_task_definition_task_role_arn` variabile nella `query` sezione.

Applicazione AWS CloudFormation della configurazione del modello

Esaminiamo un esempio più complesso di caso d'uso in produzione. In questo esempio, scriviamo regole Guard per garantire controlli più rigorosi su come vengono definite ECS le attività di Amazon.

Di seguito è riportato un CloudFormation modello di esempio.

```
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'
```

Sulla base di questo modello, scriviamo le seguenti regole per garantire che questi requisiti siano soddisfatti:

- A ogni `AWS::ECS::TaskDefinition` risorsa del modello sono associati sia un ruolo di attività che un ruolo di esecuzione.
- I ruoli di attività e i ruoli di esecuzione sono ruoli AWS Identity and Access Management (IAM).
- I ruoli sono definiti nel modello.
- La `PermissionsBoundary` proprietà è specificata per ogni ruolo.

```
# 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
    }
}
```

Composizione di blocchi con regole denominate in AWS CloudFormation Guard

Quando si scrivono blocchi con regole denominate utilizzando AWS CloudFormation Guard, è possibile utilizzare i due stili di composizione seguenti:

- Dipendenza condizionale
- Dipendenza correlazionale

L'uso di uno di questi stili di composizione delle dipendenze aiuta a promuovere la riusabilità e riduce la verbosità e la ripetizione nei blocchi con regole denominate.

Argomenti

Prerequisiti

Scopri i blocchi con regole denominate in Writing rules.

Composizione delle dipendenze condizionali

In questo stile di composizione, la valutazione di un `when` blocco o di un blocco con regole denominate dipende condizionatamente dal risultato della valutazione di uno o più altri blocchi o clausole con regole denominate. Il seguente file di regole Guard contiene blocchi con regole denominate che dimostrano dipendenze condizionali.

```
# 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
    ...
}
```

Nel file di regole di esempio precedente, `Example-1` ha i seguenti risultati possibili:

- Se `rule_name_A` restituisce a`PASS`, vengono valutate le regole Guard incapsulate da. `rule_name_B`
- Se `rule_name_A` restituisce a`FAIL`, le regole Guard incapsulate da non vengono valutate. `rule_name_B` `rule_name_B`valuta `SKIP` in.
- Se `rule_name_A` restituisce a`SKIP`, le regole Guard incapsulate da `rule_name_B` non vengono valutate. `rule_name_B`valuta `SKIP` in.

> **Note**
>
> Questo caso si verifica se dipende `rule_name_A` condizionatamente da una regola che restituisce `FAIL` e determina la valutazione a. `rule_name_A` `SKIP`

Di seguito è riportato un esempio di elemento di configurazione del database di gestione della configurazione (CMDB) tratto da un AWS Config elemento per le informazioni sui gruppi di sicurezza in ingresso e in uscita. Questo esempio dimostra la composizione delle dipendenze condizionali.

```
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
                }
            }
        }
    }
}
```

Nell'esempio precedente, `check_parameter_validity` dipende condizionatamente da `check_resource_type_and_parameter` e `check_ip_procotol_and_port_range_validity` dipende condizionatamente da. `check_parameter_validity` Quanto segue è un elemento di configurazione del database di gestione della configurazione (CMDB) conforme alle regole precedenti.

```
---
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
```

Composizione delle dipendenze correlazionali

In questo stile di composizione, la valutazione di un `when` blocco o di un blocco con regole denominate dipende in modo correlazionale dal risultato della valutazione di una o più altre regole di Guard. La dipendenza correlazionale può essere ottenuta come segue.

```
# 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
    ...
}
```

Per aiutarti a comprendere la composizione delle dipendenze correlazionali, consulta il seguente esempio di un file di regole di Guard.

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

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

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

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

Nel file di regole precedente, `ensure_elbs_are_internal_and_secure` ha una dipendenza correlazionale da. `ensure_all_elbs_are_secure` Di seguito è riportato un CloudFormation modello di esempio conforme alle regole precedenti.

```
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...'
```

Scrittura di clausole per eseguire valutazioni basate sul contesto

AWS CloudFormation Guard le clausole vengono valutate sulla base di dati gerarchici. Il motore di valutazione Guard risolve le interrogazioni sui dati in entrata seguendo i dati gerarchici come specificato, utilizzando una semplice notazione punteggiata. Spesso sono necessarie più clausole per la valutazione rispetto a una mappa di dati o a una raccolta. Guard fornisce una sintassi comoda per scrivere tali clausole. Il motore è contestualmente consapevole e utilizza i dati corrispondenti associati per le valutazioni.

Di seguito è riportato un esempio di configurazione di Kubernetes Pod con contenitori, a cui è possibile applicare valutazioni sensibili al contesto.

```
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
```

Puoi creare clausole Guard per valutare questi dati. Quando si valuta un file di regole, il contesto è l'intero documento di input. Di seguito sono riportate alcune clausole di esempio che convalidano l'applicazione dei limiti per i contenitori specificati in un Pod.

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

#
# Our rule only evaluates for apiVersion == v1 and K8s kind is Pod
#
rule ensure_container_limits_are_enforced
    when apiVersion == 'v1'
        kind == 'Pod'
{
    spec.containers[*] {
        resources.limits {
            #
            # Ensure that cpu attribute is set
            #
            cpu exists
            <<
                Id: K8S_REC_18
                Description: CPU limit must be set for the container
            >>

            #
```

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

Comprensione nelle valutazioni **`context`**

A livello di blocco di regole, il contesto in entrata è il documento completo. Le valutazioni della `when` condizione vengono eseguite in base a questo contesto radice in entrata in cui si trovano gli attributi `apiVersion` and`kind`. Nell'esempio precedente, queste condizioni restituiscono `a``true`.

Ora, attraversate la gerarchia `spec.containers[*]` mostrata nell'esempio precedente. Per ogni attraversamento della gerarchia, il valore del contesto cambia di conseguenza. Al termine dell'attraversamento del `spec` blocco, il contesto cambia, come illustrato nell'esempio seguente.

```
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
```

Dopo aver attraversato l'`containers`attributo, il contesto viene mostrato nell'esempio seguente.

```
- 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
```

Comprensione dei loop

È possibile utilizzare l'espressione `[*]` per definire un ciclo per tutti i valori contenuti nell'array per l'`containers`attributo. Il blocco viene valutato per ogni elemento al suo interno`containers`. Nel frammento di regola dell'esempio precedente, le clausole contenute all'interno del blocco definiscono i controlli da convalidare rispetto a una definizione di contenitore. Il blocco di clausole contenuto all'interno viene valutato due volte, una volta per ogni definizione di contenitore.

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

Per ogni iterazione, il valore di contesto è il valore in corrispondenza dell'indice corrispondente.

> **Note**
>
> L'unico formato di accesso all'indice supportato è `[<integer>]` o`[*]`. Attualmente, Guard non supporta intervalli come`[2..4]`.

Matrici

Spesso nei luoghi in cui viene accettato un array, vengono accettati anche valori singoli. Ad esempio, se è presente un solo contenitore, l'array può essere eliminato e viene accettato il seguente input.

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

Se un attributo può accettare un array, assicurati che la regola utilizzi il modulo di matrice. Nell'esempio precedente, si usa `containers[*]` e non`containers`. Guard valuta correttamente quando attraversa i dati quando incontra solo la forma a valore singolo.

> **Note**
>
> Usa sempre il modulo di matrice quando esprimi l'accesso a una clausola di regola quando un attributo accetta un array. Guard valuta correttamente anche nel caso in cui venga utilizzato un solo valore.

Utilizzando il modulo **`spec.containers[*]`** invece di **`spec.containers`**

Le query Guard restituiscono una raccolta di valori risolti. Quando si utilizza il modulo`spec.containers`, i valori risolti per la query contengono l'array a cui si fa riferimento`containers`, non gli elementi al suo interno. Quando si utilizza il modulo`spec.containers[*]`, si fa riferimento a ogni singolo elemento contenuto. Ricordatevi di usare il `[*]` modulo ogni volta che intendete valutare ogni elemento contenuto nell'array.

Usato **`this`** per fare riferimento al valore di contesto corrente

Quando si crea una regola Guard, è possibile fare riferimento al valore di contesto utilizzando`this`. Spesso `this` è implicito perché è legato al valore del contesto. Ad esempio, `this.apiVersion`this.`kind`, e `this.spec` sono associati alla radice o al documento. Al contrario, `this.resources` è associato a ciascun valore per`containers`, ad esempio `/spec/containers/0/` e`/spec/containers/1`. Allo stesso modo, `this.cpu` e `this.memory` mappa fino ai limiti, in particolare `/spec/containers/0/resources/limits` e`/spec/containers/1/resources/limits`.

Nel prossimo esempio, la regola precedente per la configurazione di Kubernetes Pod viene riscritta per essere utilizzata in modo esplicito. `this`

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

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

Non è necessario utilizzarlo in modo esplicito. `this` Tuttavia, il `this` riferimento può essere utile quando si lavora con gli scalari, come mostrato nell'esempio seguente.

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

Nell'esempio precedente, `this` viene utilizzato per fare riferimento a ciascun numero di porta.

Potenziali errori con l'uso di dati impliciti **`this`**

Quando si creano regole e clausole, si verificano alcuni errori comuni quando si fa riferimento a elementi del valore di contesto implicito. `this` Ad esempio, considerate il seguente dato di input rispetto al quale effettuare la valutazione (questo deve passare).

```
resourceType: 'AWS::EC2::SecurityGroup'
InputParameters:
  TcpBlockedPorts: [21, 22, 110]
configuration:
  ipPermissions:
  - fromPort: 172
    ipProtocol: tcp
    ipv6Ranges: []
    prefixListIds: []
    toPort: 172
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: "0.0.0.0/0"
  - fromPort: 89
    ipProtocol: tcp
    ipv6Ranges:
      - cidrIpv6: "::/0"
    prefixListIds: []
    toPort: 109
    userIdGroupPairs: []
    ipv4Ranges:
      - cidrIp: 10.2.0.0/24
```

Quando viene testata rispetto al modello precedente, la regola seguente genera un errore perché presuppone erroneamente di sfruttare l'implicito. `this`

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

        ipProtocol != 'udp' ]

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

Per illustrare questo esempio, salvate il file delle regole precedenti con il nome `any_ip_ingress_check.guard` e i dati con il nome del file. `ip_ingress.yaml` Quindi, esegui il `validate` comando seguente con questi file.

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

Nell'output seguente, il motore indica che il suo tentativo di recuperare una proprietà `InputParameters.TcpBlockedPorts[*]` sul valore `/configuration/ipPermissions/1` è `/configuration/ipPermissions/0` fallito.

```
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[*]
```

Per facilitare la comprensione di questo risultato, riscrivi la regola utilizzando riferimenti `this` espliciti.

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

        ipProtocol != 'udp' ]

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

`this.InputParameters`fa riferimento a ogni valore contenuto nella variabile. `any_ip_permissions` L'interrogazione assegnata alla variabile seleziona

`configuration.ipPermissions` i valori che corrispondono. L'errore indica un tentativo di recupero `InputParamaters` in questo contesto, ma si `InputParameters` è verificato nel contesto principale.

Il blocco interno fa riferimento anche a variabili che non rientrano nell'ambito, come illustrato nell'esempio seguente.

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

`this`si riferisce a ogni valore di porta in`[21, 22, 110]`, ma si riferisce anche a `fromPort` and`toPort`. Entrambi appartengono all'ambito del blocco esterno.

Risoluzione degli errori con l'uso implicito di **`this`**

Utilizzate le variabili per assegnare e fare riferimento in modo esplicito ai valori. Innanzitutto, `InputParameter.TcpBlockedPorts` fa parte del contesto di input (root). `InputParameter.TcpBlockedPorts`Esci dal blocco interno e assegnalo in modo esplicito, come mostrato nell'esempio seguente.

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

Quindi, fate riferimento a questa variabile in modo esplicito.

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

Fate lo stesso per i `this` riferimenti interni`%ports`.

Tuttavia, non tutti gli errori sono ancora stati corretti perché il loop interno contiene `ports` ancora un riferimento errato. L'esempio seguente mostra la rimozione del riferimento errato.

```
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
                    >>
                }
```

```
            }
        }
    }
}
```

Quindi, esegui nuovamente il `validate` comando. Questa volta, passa.

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

Quanto segue è l'output del `validate` comando.

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

Per testare questo approccio in caso di errori, l'esempio seguente utilizza una modifica del payload.

```
resourceType: 'AWS::EC2::SecurityGroup'
InputParameters:
  TcpBlockedPorts: [21, 22, 90, 110]
configuration:
  ipPermissions:
    - fromPort: 172
      ipProtocol: tcp
      ipv6Ranges: []
      prefixListIds: []
      toPort: 172
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: "0.0.0.0/0"
    - fromPort: 89
      ipProtocol: tcp
      ipv6Ranges:
        - cidrIpv6: "::/0"
      prefixListIds: []
      toPort: 109
      userIdGroupPairs: []
      ipv4Ranges:
        - cidrIp: 10.2.0.0/24
```

90 rientra nell'intervallo compreso tra 89 e 109 a cui è consentito qualsiasi indirizzo. IPv6 Di seguito è riportato l'output del `validate` comando dopo averlo eseguito nuovamente.

```
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 Regole di test

Puoi utilizzare il framework di unit testing AWS CloudFormation Guard integrato per verificare che le regole di Guard funzionino come previsto. Questa sezione fornisce una procedura dettagliata su come scrivere un file di unit testing e su come utilizzarlo per testare il file delle regole con il `test` comando.

Il file di unit test deve avere una delle seguenti estensioni:`.json`,,`.JSON`, `.jsn` `.yaml`.YAML, o. `.yml`

Argomenti

- Prerequisiti
- Panoramica dei file di test delle unità Guard
- Procedura dettagliata per la scrittura di un file di test unitario delle regole di Guard

Prerequisiti

Scrivi le regole di Guard in base alle quali valutare i dati di input. Per ulteriori informazioni, consulta regole di Writing Guard.

Panoramica dei file di test delle unità Guard

I file di test delle unità di Guard sono JSON file in formato o YAML formattato che contengono più input e i risultati attesi per le regole scritte all'interno di un file di regole di Guard. Possono esserci più esempi per valutare aspettative diverse. Ti consigliamo di iniziare verificando la presenza di input vuoti e quindi di aggiungere progressivamente informazioni per valutare varie regole e clausole.

Inoltre, si consiglia di denominare i file di unit test utilizzando il suffisso o. `_test.json` `_tests.yaml` Ad esempio, se avete un file di regole denominato`my_rules.guard`, assegnate un nome al file `my_rules_tests.yaml` di unit testing.

Sintassi

Di seguito viene illustrata la sintassi di un file di unit test in YAML formato.

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

Proprietà

Di seguito sono riportate le proprietà di un file di test Guard.

`input`

Dati in base ai quali testare le tue regole. È consigliabile che il primo test utilizzi un input vuoto, come illustrato nell'esempio seguente.

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

Per i test successivi, aggiungi i dati di input al test.

Campo obbligatorio: sì

`expectations`

Il risultato previsto quando regole specifiche vengono valutate rispetto ai dati di input. Specificate una o più regole da testare oltre al risultato previsto per ogni regola. Il risultato previsto deve essere uno dei seguenti:

- `PASS`— Quando vengono eseguite sulla base dei dati di input, le regole restituiscono lo stesso risultato`true`.
- `FAIL`— Quando vengono eseguite in base ai dati di input, le regole restituiscono lo stesso risultato`false`.
- `SKIP`— Quando viene eseguita sui dati di input, la regola non viene attivata.

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

Campo obbligatorio: sì

Procedura dettagliata per la scrittura di un file di test unitario delle regole di Guard

Di seguito è riportato un file di regole denominato. `api_gateway_private.guard` L'intento di questa regola è verificare se tutti i tipi di risorse Amazon API Gateway definiti in un CloudFormation modello sono distribuiti solo per l'accesso privato. Verifica inoltre se almeno una dichiarazione di policy consente l'accesso da un cloud privato virtuale ()VPC.

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

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

# 2) All AWS::ApiGateway::RestApi resources deployed must have at least one AWS
 Identity and Access Management (IAM) policy condition key to allow access from a VPC.
#
# Expectations:
```

```
# 1) SKIP when there are no AWS::ApiGateway::RestApi resources in the template.
# 2) PASS when:
#     ALL AWS::ApiGateway::RestApi resources in the template have
 the EndpointConfiguration property set to Type: PRIVATE.
#     ALL AWS::ApiGateway::RestApi resources in the template have one IAM condition key
 specified in the Policy property with aws:sourceVpc or :SourceVpc.
# 3) FAIL otherwise.

#
#

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

    }
}

rule check_rest_api_has_vpc_access when check_rest_api_is_private {
    %api_gws {
        Properties {
            #
            # ALL AWS::ApiGateway::RestApi resources in the template have one IAM
 condition key specified in the Policy property with
            #     aws:sourceVpc or :SourceVpc
            #
            some Policy.Statement[*] {
                Condition.*[ keys == /aws:[sS]ource(Vpc|VPC|Vpce|VPCE)/ ] !empty
            }
        }
    }
}
```

Questa procedura dettagliata verifica l'intento della prima regola: tutte le `AWS::ApiGateway::RestApi` risorse distribuite devono essere private.

1. Crea un file di unit test chiamato `api_gateway_private_tests.yaml` che contiene il seguente test iniziale. Con il test iniziale, aggiungi un input vuoto e aspettati che la regola `check_rest_api_is_private` venga ignorata perché non ci sono `AWS::ApiGateway::RestApi` risorse come input.

   ```
   ---
   ```

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

2. Esegui il primo test nel tuo terminale usando il `test` comando. Per il `--rules-file` parametro, specifica il file delle regole. Per il `--test-data` parametro, specificate il file di test unitario.

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

Il risultato del primo test èPASS.

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

3. Aggiungi un altro test al tuo file di unit test. Ora estendi il test per includere risorse vuote. Di seguito è riportato il `api_gateway_private_tests.yaml` file aggiornato.

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

4. Esegui `test` con il file di unit test aggiornato.

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

Il risultato del secondo test èPASS.

```
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. Aggiungi altri due test al tuo file di unit testing. Estendi il test per includere quanto segue:

   - Una `AWS::ApiGateway::RestApi` risorsa senza proprietà specificate.

     > **Note**
     >
     > Questo non è un CloudFormation modello valido, ma è utile per verificare se la regola funziona correttamente anche per input non validi.

     Aspettatevi che questo test abbia esito negativo perché la `EndpointConfiguration` proprietà non è specificata e quindi non è impostata su. `PRIVATE`

   - Una `AWS::ApiGateway::RestApi` risorsa che soddisfa il primo intento con la `EndpointConfiguration` proprietà impostata su `PRIVATE` ma non soddisfa il secondo intento perché non ha istruzioni politiche definite. Aspettatevi che questo test venga superato.

   Di seguito è riportato il file di unit test aggiornato.

```
---
- 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. Esegui `test` con il file di unit test aggiornato.

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

Il terzo risultato è`FAIL`, e il quarto risultato è`PASS`.

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

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

Test Case #3
Name: "MyTest3"
  PASS Rules:
```

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

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

7. Commenta i test 1-3 nel tuo file di unit testing. Accedi al contesto dettagliato solo per il quarto test. Di seguito è riportato il file di unit test aggiornato.

```
---
#- 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. Controlla i risultati della valutazione eseguendo il `test` comando nel tuo terminale, usando il `--verbose` flag. Il contesto verboso è utile per comprendere le valutazioni. In questo caso, fornisce informazioni dettagliate sul motivo per cui il quarto test ha avuto esito positivo. PASS

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

Ecco l'output di quella corsa.

```
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)
```

L'osservazione chiave dell'output è la riga`Clause(Location[file:api_gateway_private.guard, line:22,`

`column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`, che indica che il controllo è stato superato. L'esempio ha mostrato anche il caso in cui ci si `Types` aspettava che fosse un array, ma è stato fornito un singolo valore. In tal caso, Guard ha continuato a valutare e ha fornito un risultato corretto.

9. Aggiungi un test case come il quarto test case al tuo file di unit testing per una `AWS::ApiGateway::RestApi` risorsa con la `EndpointConfiguration` proprietà specificata. Il test case fallirà invece di essere superato. Di seguito è riportato il file di unit test aggiornato.

```
---
#- 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. Esegui il `test` comando con il file di unit test aggiornato usando il `--verbose` flag.

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

Il risultato è quello `FAIL` previsto perché `REGIONAL` è stato specificato `EndpointConfiguration` ma non è previsto.

```
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)
```

L'output dettagliato del `test` comando segue la struttura del file delle regole. Ogni blocco nel file delle regole è un blocco nell'output dettagliato. Il blocco più in alto è ogni regola. Se sono presenti `when` condizioni contrarie alla regola, queste vengono visualizzate in un blocco di condizioni di pari livello. Nell'esempio seguente, la condizione `%api_gws !empty` viene testata e viene soddisfatta.

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

Una volta superata la condizione, testiamo le clausole della regola.

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

`%api_gws`è una regola di blocco che corrisponde al `BlockClause` livello dell'output (linea:21). La clausola rule è un insieme di clausole di congiunzione (AND), in cui ogni clausola di congiunzione è un insieme di disgiunzioni. `OR` La `Properties.EndpointConfiguration.Types[*] == "PRIVATE"` congiunzione ha una sola clausola,. Pertanto, l'output dettagliato mostra una singola clausola. Il percorso `/Resources/apiGw/Properties/EndpointConfiguration/Types/1` mostra quali valori dell'input vengono confrontati, che in questo caso è l'elemento `Types` indicizzato a 1.

In[Convalida dei dati di input rispetto alle regole di Guard](), è possibile utilizzare gli esempi in questa sezione per utilizzare il `validate` comando per valutare i dati di input in base alle regole.

Utilizzo dei parametri di input con AWS CloudFormation Guard regole

AWS CloudFormation Guard consente di utilizzare i parametri di input per le ricerche dinamiche dei dati durante la convalida. Questa funzionalità è particolarmente utile quando è necessario fare riferimento a dati esterni nelle regole. Tuttavia, quando si specificano le chiavi dei parametri di input, Guard richiede che non vi siano percorsi in conflitto.

Come usare

1. Utilizzate il `-i` flag `--input-parameters` o per specificare i file contenenti i parametri di input. È possibile specificare più file di parametri di input che verranno combinati per formare un contesto comune. Le chiavi dei parametri di input non possono avere percorsi in conflitto.
2. Utilizzate il `-d` flag `--data` o per specificare il file modello effettivo da convalidare.

Esempio di utilizzo

1. Create un file di parametri di input (ad esempio,`network.yaml`):

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

2. Fai riferimento a questi parametri nel tuo file guard rule (ad esempio,`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. Crea un modello di dati non valido (ad esempio,`security_groups_fail.yaml`):

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

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

4. Esegui il comando validate:

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

   In questo comando:

   - `-r`specifica il file delle regole
   - `-i`specifica il file dei parametri di input
   - `-d`specifica il file di dati (modello) da convalidare

Parametri di input multipli

È possibile specificare più file di parametri di input:

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

Tutti i file specificati con `-i` verranno combinati per formare un unico contesto per la ricerca dei parametri.

Convalida dei dati di input rispetto alle regole AWS CloudFormation Guard

È possibile utilizzare il AWS CloudFormation Guard `validate` comando per convalidare i dati in base alle regole di Guard. Per ulteriori informazioni sul `validate` comando, inclusi i parametri e le opzioni, consulta validate.

Prerequisiti

- Scrivi le regole di Guard in base alle quali convalidare i dati di input. Per ulteriori informazioni, consulta regole di Writing Guard.
- Verifica le tue regole per assicurarti che funzionino come previsto. Per ulteriori informazioni, consulta Regole di Testing Guard.

Utilizzo del **`validate`** comando

Per convalidare i dati di input in base alle regole di Guard, ad esempio un AWS CloudFormation modello, esegui il `validate` comando Guard. Per il `--rules` parametro, specificate il nome di un file di regole. Per il `--data` parametro, specificate il nome del file di dati di input.

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

Se Guard convalida correttamente i modelli, il `validate` comando restituisce uno stato di uscita di `0` ($?in bash). Se Guard identifica una violazione della regola, il `validate` comando restituisce un rapporto sullo stato delle regole che non sono riuscite. Usa il flag di riepilogo (`-s all`) per visualizzare l'albero di valutazione dettagliato che mostra come Guard ha valutato ciascuna regola.

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

Convalida di più regole rispetto a più file di dati

Per aiutare a mantenere le regole, puoi scrivere regole in più file e organizzarle come preferisci. Quindi, puoi convalidare più file di regole rispetto a uno o più file di dati. Il `validate` comando può

richiedere una directory di file per le `--rules` opzioni `--data` e. Ad esempio, è possibile eseguire il comando seguente dove `/path/to/dataDirectory` contiene uno o più file di dati e `/path/to/ruleDirectory` contiene uno o più file di regole.

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

È possibile scrivere regole per verificare se varie risorse definite in più CloudFormation modelli dispongono delle assegnazioni di proprietà appropriate per garantire la crittografia a riposo. Per semplificare la ricerca e la manutenzione, è possibile disporre di regole per il controllo della crittografia inattiva in ogni risorsa in file separati `s3_bucket_encryption.guard`, `ec2_volume_encryption.guard`, denominati e `rds_dbinstance_encrytion.guard` in una directory con il percorso`~/GuardRules/encryption_at_rest`. I CloudFormation modelli da convalidare si trovano in una directory con il percorso`~/CloudFormation/templates`. In questo caso, esegui il `validate` comando come segue.

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

Risoluzione dei problemi AWS CloudFormation Guard

Se riscontri problemi durante l'utilizzo AWS CloudFormation Guard, consulta gli argomenti di questa sezione.

Argomenti

La clausola fallisce quando non sono presenti risorse del tipo selezionato

Quando una query utilizza un filtro come`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`, ad esempio, se non ci sono `AWS::ApiGateway::RestApi` risorse nell'input, la clausola restituisce lo stesso risultato. `FAIL`

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

Per evitare questo risultato, assegnate filtri alle variabili e utilizzate il `when` controllo delle condizioni.

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

Guard non valuta i CloudFormation modelli con riferimenti in formato breve Fn::GetAtt

Guard non supporta le forme brevi delle funzioni intrinseche. Ad esempio, l'utilizzo di`!Join`, `!Sub` in un AWS CloudFormation modello in YAML formato -non è supportato. Utilizza invece le forme espanse delle funzioni CloudFormation intrinseche. Ad esempio`Fn::Join`, usa CloudFormation modelli `Fn::Sub` in YAML formato -per valutarli rispetto alle regole di Guard.

Per ulteriori informazioni sulle funzioni intrinseche, vedere il riferimento alla funzione intrinseca nella Guida per l'utente.AWS CloudFormation

Argomenti generali per la risoluzione

- Verifica che `string` i valori letterali non contengano stringhe di escape incorporate. Attualmente, Guard non supporta stringhe di escape incorporate nei valori letterali. `string`
- Verifica che i tuoi `!=` confronti confrontino tipi di dati compatibili. Ad esempio, a `string` e an non `int` sono tipi di dati compatibili per il confronto. Quando si esegue il `!=` confronto, se i valori sono incompatibili, si verifica un errore interno. Attualmente, l'errore viene soppresso e convertito in modo da `false` soddisfare la PartialEqcaratteristica di Rust.

AWS CloudFormation Guard CLIparametri e riferimenti ai comandi

I seguenti parametri e comandi globali sono disponibili tramite l'interfaccia a riga di AWS CloudFormation Guard comando (CLI).

Argomenti

Parametri CLI globali di Guard

È possibile utilizzare i seguenti parametri con qualsiasi AWS CloudFormation Guard CLI comando.

`-h`, `--help`

Stampa le informazioni di aiuto.

`-V`, `--version`

Stampa le informazioni sulla versione.

parse-tree

Genera un albero di analisi per le AWS CloudFormation Guard regole definite in un file di regole.

Sintassi

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

Parametri

`-h`, `--help`

Stampa le informazioni di aiuto.

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

Stampa l'output in JSON formato.

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

Stampa l'output in YAML formato.

`-V`, `--version`

Stampa le informazioni sulla versione.

Opzioni

`-o`, `--output`

Scrive l'albero generato in un file di output.

`-r`, `--rules`

Fornisce un file di regole.

Esempi

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

rulegen

Accetta un file AWS CloudFormation modello in YAML formato JSON - o - e genera automaticamente un set di AWS CloudFormation Guard regole che corrispondono alle proprietà delle risorse del modello. Questo comando è utile per iniziare a scrivere regole o per creare ready-to-use regole a partire da modelli noti e validi.

Sintassi

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

Parametri

`-h`, `--help`

Stampa informazioni di aiuto.

`-V`, `--version`

Stampa le informazioni sulla versione.

Opzioni

`-o`, `--output`

Scrive le regole generate in un file di output. Data la possibilità che emergano centinaia o addirittura migliaia di regole, consigliamo di utilizzare questa opzione.

`-t`, `--template`

Fornisce il percorso di un file CloudFormation modello in JSON o YAML formato.

Esempi

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

test

Convalida un file di AWS CloudFormation Guard regole rispetto a un file di test delle unità Guard in JSON o in YAML formato per determinare il successo delle singole regole.

Sintassi

```
cfn-guard test
```

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

Parametri

`-h`, `--help`

Stampa informazioni di aiuto.

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

Ordina in base all'ora dell'ultima modifica all'interno di una directory

`-V`, `--version`

Stampa le informazioni sulla versione.

`-v`, `--verbose`

Aumenta la verbosità dell'output. Può essere specificato più volte.

L'output dettagliato segue la struttura del file delle regole di Guard. Ogni blocco nel file delle regole è un blocco nell'output dettagliato. Il blocco più in alto è ogni regola. Se sono presenti `when` condizioni contrarie alla regola, queste vengono visualizzate come un blocco di condizioni di pari livello.

Opzioni

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

Fornisce il nome di un file di regole.

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

Fornisce il nome di un file o di una directory per i file di dati in uno JSON o più YAML formati.

args

<alphabetical>

Ordina alfabeticamente all'interno di una cartella.

Esempi

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

Output

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

Consulta anche

Regole di Testing Guard

validate

Convalida i dati in AWS CloudFormation Guard base alle regole per determinare il successo o il fallimento.

Sintassi

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

Parametri

`-a, --alphabetical`

Convalida i file in una directory in ordine alfabetico.

`-h, --help`

Stampa le informazioni di aiuto.

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

Convalida i file in una directory ordinata in base all'ora dell'ultima modifica.

`-P`, `--payload`

Consente di fornire regole e dati nel seguente JSON formato tramite: `stdin`

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

Per esempio:

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

Per «regole», specifica un elenco di stringhe di file di regole. Per «dati», specifica un elenco di stringhe di file di dati.

Se specificate il `--payload` flag, non specificate le `--data` opzioni `--rules` o.

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

Stampa l'output in JSON formato.

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

Mostra l'errore della clausola, incluso un riepilogo.

`-V`, `--version`

Stampa le informazioni sulla versione.

`-v`, `--verbose`

Aumenta la verbosità dell'output. Può essere specificato più volte.

Opzioni

`-d`, `--data` (stringa)

Fornisce il nome di un file o di una directory per i file di dati in uno JSON o più YAML formati. Se fornite una directory, Guard valuta le regole specificate rispetto a tutti i file di dati nella directory. La directory deve contenere solo file di dati; non può contenere sia file di dati che file di regole.

Se specificate il `--payload` flag, non specificate l'`--data`opzione.

`-o`, `--output-format` (stringa)

Scrive su un file di output.

Default: `single-line-summary`

Valori consentiti: `json` | `yaml` | `single-line-summary`

`-r`, `--rules` (stringa)

Fornisce il nome di un file di regole o di una directory di file di regole. Se fornisci una directory, Guard valuta tutte le regole della directory rispetto ai dati specificati. La directory deve contenere solo file di regole; non può contenere sia file di dati che file di regole.

Se specificate il `--payload` flag, non specificate l'`--rules`opzione.

`--show-summary` (Stringa)

Specifica la verbosità del riepilogo di valutazione delle regole Guard. Se si specifica`all`, Guard visualizza il riepilogo completo. Se lo specifichi`pass,fail`, Guard visualizza solo le informazioni di riepilogo relative alle regole approvate o non riuscite. Se lo specifichi`none`, Guard non visualizza informazioni di riepilogo. Per impostazione predefinita, `all` è specificato.

Valori consentiti: `all` | `pass,fail` | `none`

`-t`, `--type` (stringa)

Fornisce il formato dei dati di input. Quando si specifica il tipo di dati di input, Guard visualizza i nomi logici delle risorse del CloudFormation modello nell'output. Per impostazione predefinita, Guard visualizza i percorsi e i valori delle proprietà, ad esempio`Property [/Resources/vol2/Properties/Encrypted`.

Allowed values: `CFNTemplate`

Esempi

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

Output

Se Guard convalida correttamente i modelli, il `validate` comando restituisce uno stato di uscita di `0` (`$?`in bash). Se Guard identifica una violazione della regola, il `validate` comando restituisce un rapporto sullo stato delle regole che non sono riuscite. Usa il flag verbose (`-v`) per visualizzare l'albero di valutazione dettagliato che mostra come Guard ha valutato ciascuna regola.

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

Consulta anche

Convalida dei dati di input rispetto alle regole di Guard

Sicurezza in AWS CloudFormation Guard

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo modello come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per ulteriori informazioni sui programmi di conformità che si applicano a Guard, consulta AWS Services in Scope by Compliance Program AWS .
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i tuoi requisiti aziendali e le leggi e le normative applicabili

La seguente documentazione ti aiuta a capire come applicare il modello di responsabilità condivisa durante l'installazione di Guard come AWS Lambda funzione (cfn-guard-lambda):

- La sicurezza nella guida AWS Command Line Interface per l'utente
- La sicurezza nella Guida per AWS Lambda gli sviluppatori
- La sicurezza nella guida AWS Identity and Access Management per l'utente

AWS CloudFormation Guard Cronologia dei documenti

La tabella seguente descrive le versioni della documentazione per AWS CloudFormation Guard.

- Ultimo aggiornamento della documentazione: 30 giugno 2023
- Ultima versione: 3.0.0

Modifica	Descrizione	Data
Rilascio della versione 3.0.0	La versione 3.0.0 introduce i seguenti miglioramenti: • Argomenti di introduzione e installazione aggiornati per la versione di Guard 3.0.0. • Sono state aggiunte istruzioni di installazione per Homebrew e Chocolatey. • Informazioni relative alla migrazione delle regole di Guard aggiornate per riflettere le modifiche nella versione 3.0.0 di Guard. • Aggiunto un collegamento importante al repository. AWS CloudFormation Guard GitHub	30 giugno 2023
Rilascio della versione 2.1.3	La versione 2.1.3 introduce i seguenti miglioramenti: Sono state aggiunte informazioni sui miglioramenti di Guard 2.1.3. I riferimenti a Guard 2.0	9 giugno 2023

	sono stati aggiornati a Guard 2.1.3.	
Rilascio della versione 2.0.4	La versione 2.0.4 introduce i seguenti miglioramenti: La `--payload` bandiera è stata aggiunta al `validate` comando. Per ulteriori informazioni, consulta validate nel CLI riferimento Guard.	19 ottobre 2021

Rilascio della versione 2.0.3	La versione 2.0.3 introduce i seguenti miglioramenti:
	
	• Puoi fornire nomi di test per ogni test nel tuo file di unit testing. Per ulteriori informazioni, consulta Regole di Testing Guard.
	• Le seguenti opzioni sono state aggiunte al `validate` comando:
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	
	Per ulteriori informazioni, consulta validate nel CLI riferimento Guard.
		27 luglio 2021

Versione iniziale	Versione iniziale della Guida per l' AWS CloudFormation Guard utente.	15 luglio 2021

AWS Glossario

Per la AWS terminologia più recente, consultate il AWS glossario nella sezione Reference.Glossario AWS

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.