Guía del usuario

AWS CloudFormation Guard

AWS CloudFormation Guard: Guía del usuario

Table of Contents

¿Qué es AWS CloudFormation Guard?

AWS CloudFormation Guard es una herramienta de evaluación de código abierto y de uso general. policy-as-code La interfaz de línea de comandos Guard (CLI) proporciona un lenguaje simple-to-use declarativo específico del dominio (DSL) que se puede utilizar para expresar la política en forma de código. Además, puede usar CLI comandos para validar YAML datos JSON o jerárquicos estructurados según esas reglas. Guard también proporciona un marco de pruebas unitarias integrado para comprobar que las reglas funcionan según lo previsto.

Guard no valida la sintaxis válida ni los valores de propiedad permitidos de las CloudFormation plantillas. Puede utilizar la herramienta cfn-lint para realizar una inspección exhaustiva de la estructura de la plantilla.

Guard no proporciona controles desde el lado del servidor. Puedes usar los CloudFormation Hooks para realizar la validación y el cumplimiento en el servidor, donde puedes bloquear o advertir una operación.

Para obtener información detallada sobre AWS CloudFormation Guard el desarrollo, consulta el repositorio de Guard GitHub .

Temas

¿Es la primera vez que utiliza Guard?

Si es la primera vez que utilizas Guard, te recomendamos que comiences leyendo las siguientes secciones:

- Configuración de Guard— En esta sección se describe cómo instalar Guard. Con Guard, puede redactar reglas de políticas con Guard DSL y validar sus datos estructurados JSON (o YAML formateados) según esas reglas.

- Reglas de Writing Guard— En esta sección se proporcionan tutoriales detallados para redactar reglas políticas.
- Reglas de Testing Guard— En esta sección se proporciona un tutorial detallado para probar las reglas y comprobar que funcionan según lo previsto y validar los datos estructurados (o YAML formateadosJSON) con arreglo a las reglas.
- Validar los datos de entrada según las reglas de Guard— Esta sección proporciona un recorrido detallado para validar sus datos estructurados (o YAML formateadosJSON) con arreglo a sus reglas.
- CLIReferencia de guardia— En esta sección se describen los comandos que están disponibles en el Guard. CLI

Características de la protección

Con Guard, puede escribir reglas de políticas para validar cualquier JSON dato estructurado YAML con formato o con cualquier dato estructurado, incluidas, entre otras, las AWS CloudFormation plantillas. Guard es compatible con todo el espectro de end-to-end evaluación de las verificaciones de políticas. Las reglas son útiles en los siguientes ámbitos empresariales:

- Gestión y cumplimiento preventivos (pruebas con turnos de izquierda): valide la infraestructura como código (IaC) o las composiciones de la infraestructura y los servicios con arreglo a las normas políticas que representen las mejores prácticas organizativas en materia de seguridad y cumplimiento. Por ejemplo, puede validar CloudFormation plantillas, conjuntos de CloudFormation cambios, archivos de configuración JSON basados en Terraform o configuraciones de Kubernetes.
- Control y conformidad de los Detectives: valide la conformidad de los recursos de la base de datos de gestión de la configuración (CMDB), como los elementos de configuración AWS Config basados (CIs). Por ejemplo, los desarrolladores pueden usar Guard Policy Against AWS Config CIs para monitorear continuamente el estado de los AWS recursos desplegados AWS y los que no lo son, detectar las infracciones de las políticas e iniciar las correcciones.
- Seguridad en el despliegue: asegúrese de que los cambios sean seguros antes del despliegue. Por ejemplo, valide los conjuntos de CloudFormation cambios según las reglas de políticas para evitar cambios que provoquen la sustitución de recursos, como cambiar el nombre de una tabla de Amazon DynamoDB.

Uso de Guard with Hooks CloudFormation

Puedes usar CloudFormation Guard para crear un Hook in CloudFormation Hooks. CloudFormation Hooks te permite hacer cumplir tus reglas de Guard de forma proactiva antes de CloudFormation crear, actualizar o eliminar operaciones y AWS Cloud Control API crear o actualizar operaciones. Hooks garantiza que las configuraciones de tus recursos cumplan con las mejores prácticas de seguridad, operativas y de optimización de costes de tu organización.

Para obtener más información sobre cómo usar Guard para crear CloudFormation Guard Hooks, consulte las reglas de Write Guard para evaluar los recursos de la Guía del usuario de Guard AWS CloudFormation Hooks in the Hooks.

Acceder a Guard

Para acceder a la Guardia DSL y a los comandos, debe instalar la GuardiaCLI. Para obtener información sobre la instalación del protectorCLI, consulteConfiguración de Guard.

Prácticas recomendadas

Escriba reglas sencillas y utilice reglas con nombre para hacer referencia a ellas en otras reglas. Las reglas complejas pueden ser difíciles de mantener y probar.

Con AWS CloudFormation Guard figuración

AWS CloudFormation Guard es una interfaz de línea de comandos de código abierto (). CLI Le proporciona un lenguaje sencillo y específico para cada dominio para escribir reglas de políticas y validar su estructura jerárquica y sus YAML datos con arreglo a JSON esas reglas. Las reglas pueden representar las directrices de las políticas de la empresa relacionadas con la seguridad, el cumplimiento y mucho más. Los datos jerárquicos estructurados pueden representar la infraestructura de la nube descrita como código. Por ejemplo, puede crear reglas para garantizar que siempre modelen los buckets cifrados de Amazon Simple Storage Service (Amazon S3) en sus plantillas. CloudFormation

En los temas siguientes se proporciona información sobre cómo instalar Guard utilizando el sistema operativo o como una AWS Lambda función que haya elegido.

Temas

Instalación de Guard para Linux y macOS

Puedes realizar la instalación AWS CloudFormation Guard para Linux y macOS mediante la versión binaria prediseñada, Cargo, o mediante Homebrew.

Instala Guard desde una versión binaria prediseñada

Utilice el siguiente procedimiento para instalar Guard desde un archivo binario prediseñado.

1. Abre una terminal y ejecuta el siguiente comando.

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

2. Ejecute el siguiente comando para configurar la PATH variable.

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

Resultados: Ha instalado correctamente Guard y ha establecido la PATH variable.

- (Opcional) Para confirmar la instalación de Guard, ejecute el siguiente comando.

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

El comando devuelve el resultado siguiente.

```
cfn-guard 3.0.0
```

Instale Guard from Cargo

Cargo es el administrador de paquetes de Rust. Complete los siguientes pasos para instalar Rust, que incluye Cargo. A continuación, instale Guard from Cargo.

1. Ejecute el siguiente comando desde una terminal y siga las instrucciones que aparecen en pantalla para instalar Rust.

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

   - (Opcional) Para entornos Ubuntu, ejecuta el siguiente comando.

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

2. Configure la variable de PATH entorno y ejecute el siguiente comando.

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

3. Con Cargo instalado, ejecute el siguiente comando para instalar Guard.

```
cargo install cfn-guard
```

   Resultados: Ha instalado Guard correctamente.

   - (Opcional) Para confirmar la instalación de Guard, ejecute el siguiente comando.

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

El comando devuelve el resultado siguiente.

```
cfn-guard 3.0.0
```

Instale Guard desde Homebrew

Homebrew es un administrador de paquetes para macOS y Linux. Complete los siguientes pasos para instalar Homebrew. Luego, instala Guard desde Homebrew.

1. Ejecute el siguiente comando desde una terminal y siga las instrucciones que aparecen en pantalla para instalar Homebrew.

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

2. Con Homebrew instalado, ejecuta el siguiente comando para instalar Guard.

   ```
   brew install cloudformation-guard
   ```

   Resultados: Ha instalado Guard correctamente.

   - (Opcional) Para confirmar la instalación de Guard, ejecute el siguiente comando.

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

     El comando devuelve el resultado siguiente.

     ```
     cfn-guard 3.0.0
     ```

Instalación de Guard para Windows

Puede realizar la instalación AWS CloudFormation Guard para Windows mediante Cargo o Chocolatey.

Requisitos previos

Para compilar Guard desde la interfaz de línea de comandos, debe instalar las herramientas de compilación para Visual Studio 2019.

1. Descargue las herramientas de compilación de Microsoft Visual C++ desde el sitio web Build Tools for Visual Studio 2019.
2. Ejecute el instalador y seleccione los valores predeterminados.

Instale Guard from Cargo

Cargo es el administrador de paquetes de Rust. Complete los siguientes pasos para instalar Rust, que incluye Cargo. A continuación, instale Guard from Cargo.

1. Descargue Rust y, a continuación, ejecute rustup-init.exe.
2. En la línea de comandos, selecciona 1, que es la opción predeterminada.

   El comando devuelve el resultado siguiente.

   ```
   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. Para finalizar la instalación, pulse la tecla Enter.
4. Con Cargo instalado, ejecute el siguiente comando para instalar Guard.

   ```
   cargo install cfn-guard
   ```

   Resultados: Ha instalado Guard correctamente.

   - (Opcional) Para confirmar la instalación de Guard, ejecute el siguiente comando.

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

     El comando devuelve el resultado siguiente.

```
cfn-guard 3.0.0
```

Instale Guard desde Chocolatey

Chocolatey es un administrador de paquetes para Windows. Complete los siguientes pasos para instalar Chocolatey. A continuación, instale Guard from Chocolatey.

1. Siga esta guía para instalar Chocolatey
2. Con Chocolatey instalado, ejecute el siguiente comando para instalar Guard.

   ```
   choco install cloudformation-guard
   ```

   Resultados: Ha instalado Guard correctamente.

   - (Opcional) Para confirmar la instalación de Guard, ejecute el siguiente comando.

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

     El comando devuelve el resultado siguiente.

     ```
     cfn-guard 3.0.0
     ```

Instalación de Guard como AWS Lambda función

Puede instalarlo AWS CloudFormation Guard a través de Cargo, el administrador de paquetes de Rust. Guard as an AWS Lambda function (`cfn-guard-lambda`) es un envoltorio ligero alrededor de Guard (`cfn-guard`) que se puede utilizar como función Lambda.

Requisitos previos

Antes de poder instalar Guard como una función Lambda, debe cumplir los siguientes requisitos previos:

- AWS Command Line Interface (AWS CLI) configurado con permisos para implementar e invocar funciones Lambda. Para obtener más información, consulte Configuración de la AWS CLI.

- Una función AWS Lambda de ejecución en AWS Identity and Access Management ()IAM. Para obtener más información, consulte función AWS Lambda de ejecución.
- En RHEL entornos CentOS/, agregue el repositorio de `musl-libc` paquetes a su configuración de yum. Para obtener más información, consulta ngompa/musl-libc.

Instale el administrador de paquetes Rust

Cargo es el administrador de paquetes de Rust. Complete los siguientes pasos para instalar Rust, que incluye Cargo.

1. Ejecute el siguiente comando desde una terminal y, a continuación, siga las instrucciones que aparecen en pantalla para instalar Rust.

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

   - (Opcional) Para entornos Ubuntu, ejecuta el siguiente comando.

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

2. Configure la variable de `PATH` entorno y ejecute el siguiente comando.

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

Instalar Guard como una función Lambda (Linux, macOS o Unix)

Para instalar Guard como una función Lambda, complete los siguientes pasos.

1. Desde su terminal de comandos, ejecute el siguiente comando.

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

   - (Opcional) Para confirmar la instalación de Guard como función Lambda, ejecute el siguiente comando.

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

     El comando devuelve el resultado siguiente.

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

2. Para instalar el `musl` soporte, ejecute el siguiente comando.

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

3. Compile con `musl` el siguiente comando y, a continuación, ejecútelo en su terminal.

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

Para un tiempo de ejecución personalizado, AWS Lambda requiere un ejecutable con el nombre del `bootstrap` archivo .zip del paquete de implementación. Cambie el nombre del `cfn-lambda` ejecutable generado a `bootstrap` y, a continuación, agréguelo al archivo.zip.

- Para entornos macOS, cree su archivo de configuración de carga en la raíz del proyecto Rust o en`~/.cargo/config`.

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

4. Cambia al directorio `cfn-guard-lambda` raíz.

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

5. Ejecute el siguiente comando en su terminal.

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

6. Ejecute el siguiente comando para enviarlo `cfn-guard` como una función Lambda a su cuenta.

```
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
```

Para crear y ejecutar Guard como una función Lambda

Para invocar la función enviada `cfn-guard-lambda` como una función Lambda, ejecute el siguiente comando.

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

Para llamar a la estructura de solicitudes de funciones Lambda

Solicita que se `cfn-guard-lambda` requieran los siguientes campos:

- `data`— La versión en cadena de la JSON plantilla YAML o
- `rules`— La versión en cadena del archivo del conjunto de reglas

Requisitos previos y descripción general para usar las reglas de Guard

En esta sección se muestra cómo puede completar las tareas principales de Guard, que consisten en escribir, probar y validar reglas con datos YAML formateados JSON o con formato. Además, contiene tutoriales detallados que muestran cómo escribir reglas que respondan a casos de uso específicos.

Temas

Requisitos previos

Para poder escribir reglas de políticas con el lenguaje específico del dominio de Guard (DSL), debe instalar la interfaz de línea de comandos de Guard (). CLI Para obtener más información, consulte Configuración de Guard.

Descripción general del uso de las reglas de Guard

Cuando se utiliza Guard, se suelen realizar los siguientes pasos:

1. Escriba JSON o YAML formatee los datos para validarlos.
2. Escribe las reglas de la política de Guard. Para obtener más información, consulte Reglas de Writing Guard.
3. Compruebe que sus reglas funcionan según lo previsto mediante el `test` comando Guard. Para obtener más información sobre las pruebas unitarias, consulteReglas de Testing Guard.
4. Utilice el `validate` comando Guard para validar sus datos YAML formateados JSON (o con formato) según sus reglas. Para obtener más información, consulte Validar los datos de entrada según las reglas de Guard.

AWS CloudFormation Guard Reglas de escritura

En AWS CloudFormation Guard, las reglas son policy-as-code reglas. Las reglas se escriben en el lenguaje específico del dominio de Guard (DSL) con las que se pueden validar los datos JSON formateados o los que están YAML formateados. Las reglas se componen de cláusulas.

Puede guardar las reglas escritas con The Guard DSL en archivos de texto sin formato que utilicen cualquier extensión de archivo.

Puede crear varios archivos de reglas y clasificarlos como un conjunto de reglas. Los conjuntos de reglas le permiten validar sus datos YAML formateados JSON (o con formato) comparándolos con varios archivos de reglas al mismo tiempo.

Temas

Cláusulas

Las cláusulas son expresiones booleanas que se evalúan como true (`PASS`) o false (`FAIL`). Las cláusulas utilizan operadores binarios para comparar dos valores u operadores unarios que funcionan con un único valor.

Ejemplos de cláusulas unarias

La siguiente cláusula unaria evalúa si la colección `TcpBlockedPorts` está vacía.

```
InputParameters.TcpBlockedPorts not empty
```

La siguiente cláusula unaria evalúa si la `ExecutionRoleArn` propiedad es una cadena.

```
Properties.ExecutionRoleArn is_string
```

Ejemplos de cláusulas binarias

La siguiente cláusula binaria evalúa si la `BucketName` propiedad contiene la cadena`encrypted`, independientemente de las mayúsculas y minúsculas.

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

La siguiente cláusula binaria evalúa si la `ReadCapacityUnits` propiedad es inferior o igual a 5000.

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

Sintaxis para escribir las cláusulas de la regla de Guard

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

Propiedades de las cláusulas de las reglas de Guard

`query`

Expresión separada por puntos (`.`) escrita para recorrer datos jerárquicos. Las expresiones de consulta pueden incluir expresiones de filtro para dirigirse a un subconjunto de valores. Las consultas se pueden asignar a variables para que pueda escribirlas una vez y hacer referencia a ellas en cualquier otro lugar de un conjunto de reglas, lo que le permitirá acceder a los resultados de las consultas.

Para obtener más información sobre cómo escribir consultas y filtrar, consulteDefinición de consultas y filtrado.

Obligatorio: sí

`operator`

Operador binario o unario que ayuda a comprobar el estado de la consulta. El lado izquierdo (LHS) de un operador binario debe ser una consulta y el lado derecho (RHS) debe ser una consulta o un valor literal.

Operadores binarios compatibles: == (Igual) | != (No igual) | > (Mayor que) | >= (Mayor o igual que) | < (Menor que) | <= (Menor que o igual a) | IN (En una lista con la forma [x, y, z]

Operadores unarios compatibles: exists | empty | is_string | is_list | is_struct not(!)

Obligatorio: sí

query|value literal

Una consulta o un valor literal admitido, como string ointeger(64).

Valores literales admitidos:

- Todos los tipos primitivos: stringinteger(64),,float(64),bool, char regex
- Todos los tipos de rangos especializados para expresar integer(64)float(64), o char rangos expresados como:
  - r[<lower_limit>, <upper_limit>], que se traduce en cualquier valor k que satisfaga la siguiente expresión: lower_limit <= k <= upper_limit
  - r[<lower_limit>, <upper_limit>), que se traduce en cualquier valor k que satisfaga la siguiente expresión: lower_limit <= k < upper_limit
  - r(<lower_limit>, <upper_limit>], que se traduce en cualquier valor k que satisfaga la siguiente expresión: lower_limit < k <= upper_limit
  - r(<lower_limit>, <upper_limit>),que se traduce en cualquier valor k que satisfaga la siguiente expresión: lower_limit < k < upper_limit
- Matrices asociativas (mapas) para datos de estructura clave-valor anidados. Por ejemplo:

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

- Matrices de tipos primitivos o tipos de matrices asociativas

Obligatorio: condicional; obligatorio cuando se utiliza un operador binario.

custom message

Cadena que proporciona información sobre la cláusula. El mensaje se muestra en las salidas detalladas de los test comandos validate y y puede resultar útil para comprender o depurar la evaluación de las reglas en datos jerárquicos.

Obligatorio: no

Uso de consultas en las cláusulas

Para obtener información sobre cómo escribir consultas, consulte Definición de consultas y filtrado yAsignación y referencia de variables en las reglas de Guard.

Uso de operadores en cláusulas

A continuación se muestran ejemplos CloudFormation de plantillas `Template-1` y`Template-2`. Para demostrar el uso de operadores compatibles, las consultas y cláusulas de ejemplo de esta sección hacen referencia a estas plantillas de ejemplo.

Plantilla-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"
```

Plantilla-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
```

Ejemplos de cláusulas que utilizan operadores unarios

- `empty`— Comprueba si una colección está vacía. También se puede utilizar para comprobar si una consulta tiene valores en un dato jerárquico, ya que las consultas dan como resultado una recopilación. No se puede usar para comprobar si las consultas con valores de cadena tienen definida una cadena vacía (""). Para obtener más información, consulte Definición de consultas y filtrado.

  La siguiente cláusula comprueba si la plantilla tiene uno o más recursos definidos. Se evalúa PASS porque un recurso con el identificador lógico `S3Bucket` está definido en`Template-1`.

  ```
  Resources !empty
  ```

  La siguiente cláusula comprueba si hay una o más etiquetas definidas para el `S3Bucket` recurso. Se evalúa como PASS porque `S3Bucket` tiene dos etiquetas definidas para la `Tags` propiedad en`Template-1`.

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

- `exists`— Comprueba si cada aparición de la consulta tiene un valor y se puede utilizar en lugar de`!= null`.

  La siguiente cláusula comprueba si la `BucketEncryption` propiedad está definida para`S3Bucket`. Se evalúa como PASS porque `BucketEncryption` se define para `S3Bucket` en`Template-1`.

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

> **Note**
>
> Las `not exists` comprobaciones `empty` y se evalúan `true` para detectar claves de propiedad faltantes al recorrer los datos de entrada. Por ejemplo, si la `Properties` sección no está definida en la plantilla de`S3Bucket`, la cláusula se `Resources.S3Bucket.Properties.Tag empty` evalúa como. `true` Las `exists` marcas y `empty` no muestran la ruta del JSON puntero dentro del documento en los mensajes de error. Ambas cláusulas suelen tener errores de recuperación que no mantienen esta información de recorrido.

- `is_string`— Comprueba si cada aparición de la consulta es de `string` tipo.

  La siguiente cláusula comprueba si se ha especificado un valor de cadena para la `BucketName` propiedad del `S3Bucket` recurso. Se evalúa `PASS` porque el valor de cadena `"MyServiceS3Bucket"` está especificado `BucketName` en`Template-1`.

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

- `is_list`— Comprueba si cada aparición de la consulta es del `list` tipo.

  La siguiente cláusula comprueba si se ha especificado una lista para la `Tags` propiedad del `S3Bucket` recurso. El resultado es `PASS` porque se especifican dos pares clave-valor en. `Tags` `Template-1`

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

- `is_struct`— Comprueba si cada aparición de la consulta son datos estructurados.

  La siguiente cláusula comprueba si los datos estructurados están especificados para la `BucketEncryption` propiedad del `S3Bucket` recurso. Se evalúa como `PASS` porque `BucketEncryption` se especifica mediante el tipo de `ServerSideEncryptionConfiguration` propiedad *`(object)`* en`Template-1`.

> **Note**
>
> Para comprobar el estado inverso, puede usar el operador (`not !`) con los operadores `is_string`is_list, `yis_struct`.

Ejemplos de cláusulas que utilizan operadores binarios

La siguiente cláusula comprueba si el valor especificado para la `BucketName` propiedad del `S3Bucket` recurso `Template-1` contiene la cadena`encrypt`, independientemente de las mayúsculas y minúsculas. Esto se PASS debe a que el nombre del depósito especificado `"MyServiceS3Bucket"` no contiene la cadena`encrypt`.

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

La siguiente cláusula comprueba si el valor especificado para la `Size` propiedad del `NewVolume` recurso `Template-2` se encuentra dentro de un rango específico: 50 <= `Size` <= 200. Se evalúa como PASS porque `100` está especificado para. `Size`

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

La siguiente cláusula comprueba si el valor especificado para la `VolumeType` propiedad del `NewVolume` recurso `Template-2` es `io1`io2, ogp3. Se evalúa como PASS porque `io1` está especificado para`NewVolume`.

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

> **Note**
>
> Las consultas de ejemplo de esta sección muestran el uso de operadores que utilizan los recursos con una IDs `S3Bucket` y `NewVolume` lógica. Los nombres de los recursos suelen estar definidos por el usuario y se pueden nombrar arbitrariamente en una plantilla de infraestructura como código (IaC). Para escribir una regla que sea genérica y se aplique a todos los `AWS::S3::Bucket` recursos definidos en la plantilla, la forma de consulta más común utilizada es. `Resources.*[ Type == 'AWS::S3::Bucket' ]` Para obtener más información, consulta Definición de consultas y filtrado los detalles sobre el uso y explora el directorio de ejemplos del `cloudformation-guard` GitHub repositorio.

Uso de mensajes personalizados en las cláusulas

En el siguiente ejemplo, las cláusulas para `Template-2` incluir un mensaje personalizado.

```
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>>
```

Combinación de cláusulas

En Guard, cada cláusula escrita en una nueva línea se combina implícitamente con la siguiente mediante una conjunción (lógica booleanaand). Consulte el siguiente ejemplo.

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

También puedes usar la disyunción para combinar una cláusula con la siguiente especificándola `or|OR` al final de la primera cláusula.

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

En una cláusula de guarda, las disyunciones se evalúan primero, seguidas de las conjunciones. Las reglas de protección se pueden definir como una combinación de disyunción de cláusulas (una `and|AND` de `or|OR` s) que dan como resultado () o `true` (PASS). `false` `FAIL` Es similar a la forma normal conjuntiva.

Los siguientes ejemplos muestran el orden de evaluación de las cláusulas.

```
# (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
```

Todas las cláusulas que se basan en el ejemplo se `Template-1` pueden combinar mediante la conjunción. Consulte el siguiente ejemplo.

```
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
```

Uso de bloques con reglas de guardia

Los bloques son composiciones que eliminan la verbosidad y la repetición de un conjunto de cláusulas, condiciones o reglas relacionadas. Hay tres tipos de bloques:

- Bloques de consultas
- `when`bloques
- bloques con reglas nombradas

Bloques de consultas

A continuación se muestran las cláusulas que se basan en el ejemplo`Template-1`. Se utilizó la conjunción para combinar las cláusulas.

```
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
```

Se repiten partes de la expresión de consulta de cada cláusula. Puede mejorar la componibilidad y eliminar la verbosidad y la repetición de un conjunto de cláusulas relacionadas con la misma ruta de consulta inicial mediante el uso de un bloque de consulta. Se puede escribir el mismo conjunto de cláusulas, como se muestra en el siguiente ejemplo.

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

En un bloque de consulta, la consulta que precede al bloque establece el contexto de las cláusulas del bloque.

Para obtener más información sobre el uso de bloques, consulteComposición de bloques de reglas con nombre.

whenbloques

Puede evaluar los bloques de forma condicional mediante `when` bloques, que adoptan la siguiente forma.

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

La `when` palabra clave designa el inicio del `when` bloque. `conditiones` una regla de guardia. El bloque solo se evalúa si la evaluación de la condición da como resultado `true` (`PASS`).

El siguiente es un ejemplo de `when` bloque que se basa en`Template-1`.

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

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

La cláusula del `when` bloque solo se evalúa si el valor especificado `BucketName` es una cadena. Si `BucketName` se hace referencia al valor especificado en la `Parameters` sección de la plantilla, como se muestra en el siguiente ejemplo, no se evalúa la cláusula del `when` bloque.

```
Parameters:
   S3BucketName:
     Type: String

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

Bloques de reglas con nombre

Puede asignar un nombre a un conjunto de reglas (conjunto de reglas) y, a continuación, hacer referencia a estos bloques de validación modulares, denominados bloques de reglas con nombre, en otras reglas. Los bloques de reglas con nombre tienen la siguiente forma.

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

La `rule` palabra clave designa el inicio del bloque de reglas con nombre asignado.

`rule name`es una cadena legible por humanos que identifica de forma única un bloque de reglas con nombre. Es una etiqueta para el conjunto de reglas de Guard que encapsula. En este uso, el término regla de protección incluye cláusulas, bloques de consulta, bloques y `when` bloques de reglas con nombre. El nombre de la regla se puede usar para hacer referencia al resultado de la evaluación del conjunto de reglas que encapsula, lo que hace que los bloques de reglas con nombre asignado sean reutilizables. El nombre de la regla también proporciona un contexto sobre los errores de las reglas en las salidas de los comandos `validate` y`test`. El nombre de la regla se muestra

junto con el estado de evaluación del bloque (PASSFAIL, oSKIP) en el resultado de la evaluación del archivo de reglas. Consulte el siguiente ejemplo.

```
# 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**
```

También puede evaluar los bloques de reglas con nombre de forma condicional especificando la when palabra clave seguida de una condición después del nombre de la regla.

A continuación se muestra el when bloque de ejemplo que se analizó anteriormente en este tema.

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

Al usar bloques de reglas con nombre, lo anterior también se puede escribir de la siguiente manera.

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

Puedes reutilizar y agrupar bloques de reglas con nombre propio con otras reglas de Guard. A continuación se muestran algunos ejemplos.

```
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
    ...
}
```

Definición de consultas y filtrado de Guard

En este tema se describe la redacción de consultas y el uso de filtros al escribir cláusulas de reglas de Guard.

Requisitos previos

El filtrado es un AWS CloudFormation Guard concepto avanzado. Le recomendamos que revise los siguientes temas fundamentales antes de aprender sobre el filtrado:

- ¿Qué es AWS CloudFormation Guard?
- Redacción de reglas y cláusulas

Definición de consultas

Las expresiones de consulta son simples expresiones separadas por puntos (`.`) que se escriben para recorrer datos jerárquicos. Las expresiones de consulta pueden incluir expresiones de filtro para dirigirse a un subconjunto de valores. Cuando se evalúan las consultas, dan como resultado una colección de valores, similar a un conjunto de resultados devuelto por una SQL consulta.

El siguiente ejemplo de consulta busca `AWS::IAM::Role` recursos en una AWS CloudFormation plantilla.

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

Las consultas siguen estos principios básicos:

- Cada punto (`.`) de la consulta desciende en la jerarquía cuando se utiliza un término clave explícito, por ejemplo, `Properties.Encrypted.` si `Resources` alguna parte de la consulta no coincide con el dato entrante, Guard arroja un error de recuperación.
- Una parte de la consulta con puntos (`.`) que utiliza un comodín `*` recorre todos los valores de la estructura en ese nivel.
- Una parte con puntos (`.`) de la consulta que utiliza un comodín de matriz `[*]` recorre todos los índices de esa matriz.
- Todas las colecciones se pueden filtrar especificando los filtros entre corchetes. `[]` Las colecciones se pueden encontrar de las siguientes maneras:
  - Las matrices que se encuentran de forma natural en los datos son colecciones. A continuación se muestran algunos ejemplos realizados con la :

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

    Etiquetas: `[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - Al recorrer todos los valores de una estructura como `Resources.*`
  - El resultado de cualquier consulta es en sí mismo una colección desde la que se pueden filtrar aún más los valores. Consulte el siguiente ejemplo.

    ```
    let all_resources = Resource.* # query let iam_resources = %resources[ Type == /
    IAM/ ] # filter from query results let managed_policies = %iam_resources[ Type == /
    ManagedPolicy/ ] # further refinements %managed_policies { # traversing each value
     # do something with each }
    ```

A continuación se muestra un ejemplo de fragmento CloudFormation de plantilla.

```
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
    ...
```

Según esta plantilla, la ruta recorrida es `SampleRole` y el valor final seleccionado es. `Type: AWS::IAM::Role`

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

El valor resultante de la consulta `Resources.*[ Type == 'AWS::IAM::Role' ]` en YAML formato se muestra en el siguiente ejemplo.

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

Algunas de las formas en que puede utilizar las consultas son las siguientes:

- Asigne una consulta a las variables para poder acceder a los resultados de la consulta haciendo referencia a esas variables.
- Siga la consulta con un bloque que compare cada uno de los valores seleccionados.
- Compara una consulta directamente con una cláusula básica.

Asignación de consultas a variables

Guard admite la asignación de variables de una sola vez dentro de un ámbito determinado. Para obtener más información sobre las variables de las reglas de Guard, consulteAsignación y referencia de variables en las reglas de Guard.

Puede asignar consultas a las variables para poder escribirlas una vez y, después, hacer referencia a ellas en cualquier otro lugar de las reglas de Guard. Consulte los siguientes ejemplos de

asignaciones de variables, que muestran los principios de consulta que se describen más adelante en esta sección.

```
#
# 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'
    }
]
```

Recorrer directamente los valores de una variable asignada a una consulta

Guard permite comparar directamente los resultados de una consulta. En el siguiente ejemplo, el `when` bloque se compara con la `AvailabilityZone` propiedad `EncryptedVolumeType`, y de cada `AWS::EC2::Volume` recurso que se encuentra en una CloudFormation plantilla.

```
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']
        }
    }
}
```

Comparaciones directas a nivel de cláusula

Guard también admite consultas como parte de las comparaciones directas. Consulte los ejemplos siguientes.

```
let resources = Resources.*

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

En el ejemplo anterior, las dos cláusulas (que comienzan con la `some` palabra clave) expresadas en la forma que se muestra se consideran cláusulas independientes y se evalúan por separado.

Formulario de cláusula única y cláusula de bloque

En conjunto, las dos cláusulas de ejemplo que se muestran en la sección anterior no equivalen al bloque siguiente.

```
let resources = Resources.*

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

Este bloque consulta cada `Tag` valor de la colección y compara los valores de sus propiedades con los valores de propiedad esperados. La forma combinada de las cláusulas de la sección anterior evalúa las dos cláusulas de forma independiente. Tenga en cuenta la siguiente entrada.

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

Las cláusulas del primer formulario se evalúan como`PASS`. Al validar la primera cláusula del primer formulario, la siguiente ruta cruza`Resources`, `PropertiesTags`, y `Key` coincide con el valor `NotPRODEnd` y no coincide con el valor `PROD` esperado.

```
Resources:
```

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

Lo mismo ocurre con la segunda cláusula del primer formulario. La ruta que cruza `Resources`,`Properties`,`Tags`, y `Value` coincide con el valor`AppStart`. Como resultado, la segunda cláusula de forma independiente.

El resultado general es un`PASS`.

Sin embargo, el formulario de bloque se evalúa de la siguiente manera. Para cada `Tags` valor, se compara si ambos valores coinciden `Key` y`Value`, `NotAppStart` en el ejemplo siguiente, `NotPRODEnd` los valores no coinciden.

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

Porque las evaluaciones comprueban ambos y `Key == /PROD$/Value == /^App/`, además, la coincidencia no está completa. Por lo tanto, el resultado es`FAIL`.

> **Note**
>
> Cuando trabaje con colecciones, le recomendamos que utilice el formulario de cláusula de bloque cuando desee comparar varios valores de cada elemento de la colección. Utilice el formulario de cláusula única cuando la colección sea un conjunto de valores escalares o cuando desee comparar solo un atributo.

Resultados de la consulta y cláusulas asociadas

Todas las consultas devuelven una lista de valores. Cualquier parte de un recorrido, como la falta de una clave, los valores vacíos de una matriz (`Tags: []`) al acceder a todos los índices o la falta de valores en un mapa al encontrar un mapa vacío (`Resources: {}`), puede provocar errores de recuperación.

Todos los errores de recuperación se consideran errores al evaluar las cláusulas en función de dichas consultas. La única excepción se produce cuando se utilizan filtros explícitos en la consulta. Cuando se utilizan filtros, se omiten las cláusulas asociadas.

Los siguientes errores de bloqueo están asociados a la ejecución de consultas.

- Si una plantilla no contiene recursos, la consulta se evalúa y las cláusulas de nivel de bloque asociadas también se evalúan como tal. `FAIL` `FAIL`
- Cuando una plantilla contiene un bloque de recursos vacío`{ "Resources": {} }`, la consulta se evalúa como`FAIL`, y las cláusulas de nivel de bloque asociadas, también dan como resultado. `FAIL`
- Si una plantilla contiene recursos pero ninguno coincide con la consulta, la consulta devuelve resultados vacíos y se omiten las cláusulas a nivel de bloque.

Uso de filtros en las consultas

Los filtros en las consultas son, en efecto, cláusulas de protección que se utilizan como criterios de selección. A continuación se muestra la estructura de una cláusula.

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

Tenga en cuenta los siguientes puntos clave AWS CloudFormation Guard Reglas de escritura al trabajar con filtros:

- Combine las cláusulas mediante la forma normal conjuntiva () CNF.
- Especifique cada cláusula de conjunción (`and`) en una línea nueva.
- Especifique las disyunciones (`or`) mediante la `or` palabra clave entre dos cláusulas.

El siguiente ejemplo muestra las cláusulas conjuntivas y disyuntivas.

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

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

Uso de cláusulas como criterios de selección

Puede aplicar el filtrado a cualquier colección. El filtrado se puede aplicar directamente a los atributos de la entrada que ya son similares a una colección`securityGroups: [....]`. También puede aplicar el filtrado a una consulta, que siempre es una colección de valores. Puede utilizar todas las características de las cláusulas, incluida la forma normal conjuntiva, para filtrar.

La siguiente consulta común se utiliza a menudo al seleccionar recursos por tipo de una CloudFormation plantilla.

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

La consulta `Resources.*` devuelve todos los valores presentes en la `Resources` sección de la entrada. En el caso de la plantilla de ejemplo introducidaDefinición de consultas, la consulta devuelve lo siguiente.

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

Ahora, aplique el filtro a esta colección. El criterio que debe coincidir es`Type == AWS::IAM::Role`. A continuación se muestra el resultado de la consulta después de aplicar el filtro.

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

```
  ...
```

A continuación, consulte varias cláusulas para ver `AWS::IAM::Role` los recursos.

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

El siguiente es un ejemplo de consulta de filtrado que selecciona todos los `AWS::IAM::ManagedPolicy` recursos `AWS::IAM::Policy` y.

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

El siguiente ejemplo comprueba si estos recursos de políticas tienen un valor `PolicyDocument` específico.

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

Definir necesidades de filtrado más complejas

Considere el siguiente ejemplo de un elemento de AWS Config configuración para la información de los grupos de seguridad de entrada y salida.

```
---
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
```

Tenga en cuenta lo siguiente:

- `ipPermissions`(reglas de entrada) es un conjunto de reglas dentro de un bloque de configuración.
- Cada estructura de reglas contiene atributos tales como `ipv4Ranges` y `ipv6Ranges` para especificar un conjunto de CIDR bloques.

Escribamos una regla que seleccione las reglas de entrada que permitan las conexiones desde cualquier dirección IP y compruebe que las reglas no permiten exponer los puertos TCP bloqueados.

Comience con la parte de consulta que cubreIPv4, como se muestra en el siguiente ejemplo.

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

La `some` palabra clave es útil en este contexto. Todas las consultas devuelven un conjunto de valores que coinciden con la consulta. De forma predeterminada, Guard evalúa que todos los valores devueltos como resultado de la consulta se comparan con las comprobaciones. Sin embargo, es posible que este comportamiento no siempre sea lo que necesita para las comprobaciones. Tenga en cuenta la siguiente parte de la entrada del elemento de configuración.

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

Hay dos valores presentes para`ipv4Ranges`. No todos los `ipv4Ranges` valores son iguales a una dirección IP indicada por`0.0.0.0/0`. Desea comprobar si al menos un valor coincide`0.0.0.0/0`. Le dices a Guard que no es necesario que coincidan todos los resultados devueltos por una consulta, pero que al menos uno de ellos debe coincidir. La `some` palabra clave indica a Guard que se asegure de que uno o más valores de la consulta resultante coincidan con la comprobación. Si ningún valor del resultado de la consulta coincide, Guard arroja un error.

A continuación, añadaIPv6, como se muestra en el siguiente ejemplo.

```
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'
]
```

Por último, en el siguiente ejemplo, valide que el protocolo no lo seaudp.

```
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' ]
]
```

La siguiente es la regla 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
                    >>
                }
            }

        }
    }
}
```

Separar las colecciones en función de los tipos de contenido

Al utilizar plantillas de configuración de infraestructura como código (IaC), es posible que encuentre una colección que contenga referencias a otras entidades dentro de la plantilla de configuración. La siguiente es una CloudFormation plantilla de ejemplo que describe las tareas de Amazon Elastic Container Service (AmazonECS) con una referencia local a`TaskRoleArn`, una referencia a `TaskArn` y una referencia de cadena directa.

```
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'
```

Analice la siguiente consulta.

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

Esta consulta devuelve un conjunto de valores que contiene los tres `AWS::ECS::TaskDefinition` recursos que se muestran en la plantilla de ejemplo. Separe `ecs_tasks` los que contienen referencias `TaskRoleArn` locales de los demás, como se muestra en el siguiente ejemplo.

```
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
    >>
}
```

Asignación y referencia de variables en las reglas de Guard

Puedes asignar variables en tus archivos de AWS CloudFormation Guard reglas para almacenar la información a la que quieras hacer referencia en tus reglas de Guard. Guard admite la asignación de variables con un solo disparo. Las variables se evalúan de forma perezosa, lo que significa que Guard solo evalúa las variables cuando se ejecutan reglas.

Temas

Asignación de variables

Use la `let` palabra clave para inicializar y asignar una variable. Como práctica recomendada, utilice las mayúsculas y minúsculas para los nombres de las variables. Las variables pueden almacenar literales estáticos o propiedades dinámicas resultantes de las consultas. En el siguiente ejemplo, la variable `ecs_task_definition_task_role_arn` almacena el valor `arn:aws:iam:123456789012:role/my-role-name` de cadena estática.

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

En el siguiente ejemplo, la variable `ecs_tasks` almacena los resultados de una consulta que busca todos los `AWS::ECS::TaskDefinition` recursos de una AWS CloudFormation plantilla. Puede hacer referencia `ecs_tasks` a la información sobre esos recursos para acceder a ellos al escribir reglas.

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

```
]
```

Variables de referencia

Use el % prefijo para hacer referencia a una variable.

Según el ejemplo de `ecs_task_definition_task_role_arn` variable que aparece enAsignación de variables, puede hacer referencia `ecs_task_definition_task_role_arn` a la `query|value literal` sección de una cláusula de regla de guardia. El uso de esa referencia garantiza que el valor especificado para la `TaskDefinitionArn` propiedad de cualquier `AWS::ECS::TaskDefinition` recurso de una CloudFormation plantilla sea el valor de cadena estática`arn:aws:iam:123456789012:role/my-role-name`.

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

Según el ejemplo de `ecs_tasks` variable deAsignación de variables, puede hacer referencia `ecs_tasks` en una consulta (por ejemplo, %ECS_Tasks.properties). En primer lugar, Guard evalúa la variable `ecs_tasks` y, a continuación, utiliza los valores devueltos para recorrer la jerarquía. Si la variable se `ecs_tasks` resuelve en valores que no son cadenas, Guard arroja un error.

> **Note**
>
> Actualmente, Guard no admite la referencia a variables dentro de los mensajes de error personalizados.

Ámbito variable

El alcance se refiere a la visibilidad de las variables definidas en un archivo de reglas. El nombre de una variable solo se puede usar una vez dentro de un ámbito. Hay tres niveles en los que se puede declarar una variable, o tres posibles ámbitos de variables:

- A nivel de archivo: normalmente se declaran en la parte superior del archivo de reglas, pero se pueden usar variables a nivel de archivo en todas las reglas del archivo de reglas. Están visibles en todo el archivo.

  En el siguiente archivo de reglas de ejemplo, las variables `ecs_task_definition_task_role_arn` y `ecs_task_definition_execution_role_arn` se inicializan a nivel de archivo.

```
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
}
```

- Nivel de regla: declaradas dentro de una regla, las variables de nivel de regla solo son visibles para esa regla específica. Cualquier referencia fuera de la regla produce un error.

  En el siguiente archivo de reglas de ejemplo, las variables `ecs_task_definition_task_role_arn` y `ecs_task_definition_execution_role_arn` se inicializan en el nivel de la regla. Solo se `ecs_task_definition_task_role_arn` puede hacer referencia a ellas dentro de la `check_ecs_task_definition_task_role_arn` regla nombrada. Solo puede hacer referencia a la `ecs_task_definition_execution_role_arn` variable dentro de la regla `check_ecs_task_definition_execution_role_arn` nombrada.

```
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
}
```

- Nivel de bloque: declaradas dentro de un bloque, como una `when` cláusula, las variables de nivel de bloque solo son visibles para ese bloque específico. Cualquier referencia fuera del bloque produce un error.

  En el siguiente archivo de reglas de ejemplo, las variables `ecs_task_definition_task_role_arn` y `ecs_task_definition_execution_role_arn` se inicializan a nivel de bloque dentro del `AWS::ECS::TaskDefinition` bloque de tipos. Solo puede hacer referencia a `ecs_task_definition_execution_role_arn` las variables `ecs_task_definition_task_role_arn` y dentro de los bloques de `AWS::ECS::TaskDefinition` tipos para sus reglas respectivas.

```
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
    }
}
```

Ejemplos de variables en los archivos de reglas de Guard

En las siguientes secciones se proporcionan ejemplos de asignación estática y dinámica de variables.

Asignación estática

A continuación se muestra un ejemplo CloudFormation de plantilla.

```
Resources:
```

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

Basándose en esta plantilla, puede escribir una regla denominada `check_ecs_task_definition_task_role_arn` que garantice que la `TaskRoleArn` propiedad de todos los recursos de la `AWS::ECS::TaskDefinition` plantilla es`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
}
```

Dentro del ámbito de la regla, puede inicializar una variable llamada `ecs_task_definition_task_role_arn` y asignarle el valor `'arn:aws:iam::123456789012:role/my-role-name'` de cadena estática. La cláusula de regla comprueba si el valor especificado para la `TaskRoleArn` propiedad del `EcsTask` recurso es `arn:aws:iam::123456789012:role/my-role-name` haciendo referencia a la `ecs_task_definition_task_role_arn` variable de la sección. `query|value literal`

Asignación dinámica

A continuación se muestra un ejemplo CloudFormation de plantilla.

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

Basándose en esta plantilla, puede inicializar una variable llamada `ecs_tasks` dentro del ámbito del archivo y asignarle la consulta`Resources.*[ Type == 'AWS::ECS::TaskDefinition'`. Guard consulta todos los recursos de la plantilla de entrada y almacena información sobre ellos en `ecs_tasks` ella. También puede escribir una regla llamada `check_ecs_task_definition_task_role_arn` que garantice que la

`TaskRoleArn` propiedad de todos los recursos de la `AWS::ECS::TaskDefinition` plantilla sea `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 cláusula de regla comprueba si el valor especificado para la `TaskRoleArn` propiedad del `EcsTask` recurso es `arn:aws:iam::123456789012:role/my-role-name` haciendo referencia a la `ecs_task_definition_task_role_arn` variable de la `query` sección.

Hacer cumplir la configuración de la plantilla AWS CloudFormation

Veamos un ejemplo más complejo de un caso de uso de producción. En este ejemplo, escribimos reglas de Guard para garantizar controles más estrictos sobre cómo se definen ECS las tareas de Amazon.

El siguiente es un ejemplo de CloudFormation plantilla.

```
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'
```

Basándonos en esta plantilla, escribimos las siguientes reglas para garantizar que se cumplan estos requisitos:

- Cada `AWS::ECS::TaskDefinition` recurso de la plantilla tiene un rol de tarea y un rol de ejecución adjunto.
- Los roles de tarea y los roles de ejecución son AWS Identity and Access Management (IAM) roles.
- Los roles se definen en la plantilla.
- La `PermissionsBoundary` propiedad se especifica para cada función.

```
# 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
    }
}
```

Componer bloques de reglas con nombre en AWS CloudFormation Guard

Al escribir bloques de reglas con nombre AWS CloudFormation Guard, puede usar los dos estilos de composición siguientes:

- Dependencia condicional
- Dependencia correlacional

El uso de cualquiera de estos estilos de composición de dependencias ayuda a promover la reutilización y reduce la verbosidad y la repetición en los bloques de reglas nombradas.

Temas

Requisitos previos

Obtén más información sobre los bloques de reglas con nombre en Cómo escribir reglas.

Composición de dependencias condicionales

En este estilo de composición, la evaluación de un `when` bloque o bloque de reglas nominadas depende condicionalmente del resultado de la evaluación de uno o más bloques o cláusulas de reglas nombradas. El siguiente archivo de reglas de Guard de ejemplo contiene bloques de reglas con nombres que muestran las dependencias condicionales.

```
# 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
    ...
}
```

En el ejemplo anterior, el archivo de reglas `Example-1` tiene los siguientes resultados posibles:

- Si `rule_name_A` se evalúa así`PASS`, se evalúan las reglas de Guard `rule_name_B` encapsuladas por.
- Si `rule_name_A` se evalúa así`FAIL`, las reglas de Guard encapsuladas por `rule_name_B` no se evalúan. `rule_name_B`se evalúa como. `SKIP`
- Si `rule_name_A` se evalúa como tal`SKIP`, las reglas de Guard encapsuladas por `rule_name_B` no se evalúan. `rule_name_B`se evalúa como. `SKIP`

> **Note**
>
> Este caso ocurre si depende `rule_name_A` condicionalmente de una regla que evalúa `FAIL` y da como resultado una evaluación de`rule_name_A`. `SKIP`

A continuación se muestra un ejemplo de un elemento de configuración de una base de datos de administración de la configuración (CMDB) a partir de un AWS Config elemento para la información de grupos de seguridad de entrada y salida. En este ejemplo se muestra la composición de dependencias condicionales.

```
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
                }
            }
        }
    }
}
```

En el ejemplo anterior, `check_parameter_validity` depende condicionalmente de `check_resource_type_and_parameter` y `check_ip_procotol_and_port_range_validity` depende condicionalmente de. `check_parameter_validity` El siguiente es un elemento de configuración de la base de datos de administración de la configuración (CMDB) que cumple con las reglas anteriores.

```
---
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
```

Composición de dependencias correlacionales

En este estilo de composición, la evaluación de un `when` bloque o de un bloque de reglas nominadas tiene una dependencia correlacional del resultado de la evaluación de una o más reglas de Guard distintas. La dependencia correlacional se puede lograr de la siguiente manera.

```
# 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
    ...
}
```

Para ayudarle a entender la composición de la dependencia correlacional, revise el siguiente ejemplo de un archivo de reglas de 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'
}
```

En el archivo de reglas anterior, `ensure_elbs_are_internal_and_secure` tiene una dependencia correlacional de. `ensure_all_elbs_are_secure` A continuación se muestra un ejemplo de CloudFormation plantilla que se ajusta a las reglas anteriores.

```
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...'
```

Redacción de cláusulas para realizar evaluaciones sensibles al contexto

AWS CloudFormation Guard las cláusulas se evalúan en función de los datos jerárquicos. El motor de evaluación Guard resuelve las consultas con los datos entrantes siguiendo los datos jerárquicos según se especifique, mediante una simple notación punteada. Con frecuencia, se necesitan varias cláusulas para realizar la evaluación en función de un mapa de datos o de una colección. Guard proporciona una sintaxis práctica para escribir dichas cláusulas. El motor tiene en cuenta el contexto y utiliza los datos correspondientes asociados para las evaluaciones.

El siguiente es un ejemplo de una configuración de Kubernetes Pod con contenedores, a la que puedes aplicar evaluaciones contextuales.

```
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
```

Puedes crear cláusulas Guard para evaluar estos datos. Al evaluar un archivo de reglas, el contexto es todo el documento de entrada. A continuación, se muestran ejemplos de cláusulas que validan la aplicación de los límites a los contenedores especificados en 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
            >>
        }
    }
}
```

Comprensión **`context`** en las evaluaciones

A nivel de bloque de reglas, el contexto entrante es el documento completo. Las evaluaciones de la `when` condición se realizan en relación con este contexto raíz entrante en el que se encuentran `kind` los atributos `apiVersion` y. En el ejemplo anterior, estas condiciones se evalúan como`true`.

Ahora, recorra la jerarquía que `spec.containers[*]` se muestra en el ejemplo anterior. Para cada recorrido de la jerarquía, el valor del contexto cambia en consecuencia. Una vez finalizado el recorrido del `spec` bloque, el contexto cambia, como se muestra en el siguiente ejemplo.

```
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
```

Tras recorrer el `containers` atributo, el contexto se muestra en el siguiente ejemplo.

```
- 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
```

Entender los bucles

Puede utilizar la expresión `[*]` para definir un bucle para todos los valores contenidos en la matriz del `containers` atributo. El bloque se evalúa para cada uno de los elementos que contiene`containers`. En el fragmento de regla del ejemplo anterior, las cláusulas contenidas en el bloque definen las comprobaciones que deben validarse con respecto a una definición de contenedor. El bloque de cláusulas que contiene se evalúa dos veces, una para cada definición de contenedor.

```
{
```

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

Para cada iteración, el valor de contexto es el valor del índice correspondiente.

> **Note**
>
> El único formato de acceso al índice admitido es `[<integer>]` o`[*]`. Actualmente, Guard no admite rangos como`[2..4]`.

Matrices

A menudo, en los lugares donde se acepta una matriz, también se aceptan valores individuales. Por ejemplo, si solo hay un contenedor, se puede eliminar la matriz y aceptar la siguiente entrada.

```
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
```

Si un atributo puede aceptar una matriz, asegúrese de que la regla utilice la forma de matriz. En el ejemplo anterior, se utiliza `containers[*]` y no`containers`. Guard evalúa correctamente al recorrer los datos cuando encuentra solo el formulario de un solo valor.

> **Note**
>
> Utilice siempre la forma matricial al expresar el acceso a una cláusula de regla cuando un atributo acepte una matriz. Guard evalúa correctamente incluso en el caso de que se utilice un único valor.

Utilizar el formulario **`spec.containers[*]`** en lugar de **`spec.containers`**

Las consultas de protección devuelven una colección de valores resueltos. Al utilizar el formulario`spec.containers`, los valores resueltos de la consulta contienen la matriz a la que se hace referencia`containers`, no los elementos que contiene. Al utilizar el formulario`spec.containers[*]`, se hace referencia a cada elemento individual contenido. Recuerde utilizar el `[*]` formulario siempre que desee evaluar cada elemento contenido en la matriz.

Se utiliza **`this`** para hacer referencia al valor del contexto actual

Al crear una regla de protección, puede hacer referencia al valor de contexto mediante`this`. A menudo, `this` está implícita porque está vinculada al valor del contexto. Por ejemplo, `this.apiVersion`this.kind, y `this.spec` están enlazados a la raíz o al documento. Por el contrario, `this.resources` está enlazado a cada valor de`containers`, como `/spec/containers/0/` y`/spec/containers/1`. Del mismo modo, `this.cpu` y `this.memory` mapea los límites, específicamente `/spec/containers/0/resources/limits` y`/spec/containers/1/resources/limits`.

En el siguiente ejemplo, la regla anterior para la configuración del Kubernetes Pod se reescribió para utilizarla de forma explícita. `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
            >>
        }
    }
}
```

No es necesario que la utilices de forma explícita. `this` Sin embargo, la `this` referencia puede resultar útil cuando se trabaja con escalares, como se muestra en el siguiente ejemplo.

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

En el ejemplo anterior, `this` se utiliza para hacer referencia a cada número de puerto.

Posibles errores relacionados con el uso del término implícito **`this`**

Al crear reglas y cláusulas, hay algunos errores comunes al hacer referencia a elementos del valor de contexto implícito`this`. Por ejemplo, considere el siguiente dato de entrada para realizar la evaluación (debe aprobarse).

```
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
```

Cuando se compara con la plantilla anterior, la siguiente regla produce un error porque supone erróneamente que se aprovecha lo implícito. `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
                >>
            }
        }
```

```
    }
}
```

Para ver este ejemplo, guarde el archivo de reglas anterior con el nombre `any_ip_ingress_check.guard` y los datos con el nombre del archivo. `ip_ingress.yaml` A continuación, ejecute el siguiente `validate` comando con estos archivos.

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

En el siguiente resultado, el motor indica que su intento de recuperar una propiedad `InputParameters.TcpBlockedPorts[*]` del valor `/configuration/ipPermissions/0` / `configuration/ipPermissions/1` ha fallado.

```
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[*]
```

Para entender mejor este resultado, reescribe la regla utilizando referencias `this` explícitas.

```
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`hace referencia a cada valor contenido en la variable`any_ip_permissions`. La consulta asignada a la variable selecciona `configuration.ipPermissions` los valores que coinciden. El error indica un intento de recuperación `InputParamaters` en este contexto, pero `InputParameters` estaba en el contexto raíz.

El bloque interno también hace referencia a variables que están fuera del alcance, como se muestra en el siguiente ejemplo.

```
{
    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`hace referencia a cada valor de puerto de`[21, 22, 110]`, pero también hace referencia a `fromPort` y`toPort`. Ambos pertenecen al ámbito del bloque exterior.

Resolver errores con el uso implícito de **`this`**

Utilice variables para asignar valores y hacer referencia a ellos de forma explícita. En primer lugar, `InputParameter.TcpBlockedPorts` forma parte del contexto de entrada (raíz). `InputParameter.TcpBlockedPorts`Salga del bloque interno y asígnelo de forma explícita, como se muestra en el siguiente ejemplo.

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

A continuación, consulte esta variable de forma explícita.

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

Haga lo mismo con `this` las referencias internas internas`%ports`.

Sin embargo, aún no se han corregido todos los errores porque el bucle interior `ports` todavía tiene una referencia incorrecta. En el siguiente ejemplo, se muestra la eliminación de la referencia incorrecta.

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

A continuación, vuelva a ejecutar el `validate` comando. Esta vez, pasa.

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

El siguiente es el resultado del `validate` comando.

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

Para probar este enfoque en busca de errores, en el siguiente ejemplo se utiliza un cambio de carga útil.

```
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 está dentro del rango de 89 a 109, y todas IPv6 las direcciones están permitidas. El siguiente es el resultado del `validate` comando después de volver a ejecutarlo.

```
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 Reglas de prueba

Puede utilizar el marco de pruebas unitarias AWS CloudFormation Guard integrado para comprobar que las reglas de Guard funcionan según lo previsto. En esta sección se proporciona un tutorial

sobre cómo escribir un archivo de pruebas unitarias y cómo usarlo para probar el archivo de reglas con el `test` comando.

El archivo de prueba unitaria debe tener una de las siguientes extensiones: `.json`.`JSON`,`.jsn`, `.yaml`.`YAML`, `o`.`yml`.

Temas

- Requisitos previos
- Descripción general de los archivos de pruebas de las unidades Guard
- Tutorial sobre cómo escribir un archivo de pruebas unitarias de Guard Rules

Requisitos previos

Escribe reglas de Guard para evaluar tus datos de entrada. Para obtener más información, consulte Reglas de Writing Guard.

Descripción general de los archivos de pruebas de las unidades Guard

Los archivos de pruebas de las unidades de Guard son JSON archivos YAML con formato o formato que contienen múltiples entradas y los resultados esperados de las reglas escritas en un archivo de reglas de Guard. Puede haber varios ejemplos para evaluar diferentes expectativas. Le recomendamos que comience por comprobar si hay entradas vacías y, a continuación, vaya añadiendo información de forma progresiva para evaluar las distintas reglas y cláusulas.

Además, le recomendamos que nombre los archivos de pruebas unitarias con el sufijo `_test.json` o`_tests.yaml`. Por ejemplo, si tiene un nombre para un archivo de reglas`my_rules.guard`, asígnele un nombre al archivo `my_rules_tests.yaml` de pruebas unitarias.

Sintaxis

A continuación se muestra la sintaxis de un archivo de pruebas unitarias en YAML formato.

```
---
- name: <TEST NAME>
  input:
     <SAMPLE INPUT>
   expectations:
     rules:
```

```
        <RULE NAME>: [PASS|FAIL|SKIP]
```

Propiedades

A continuación se muestran las propiedades de un archivo de prueba de Guard.

`input`

Datos con los que poner a prueba tus reglas. Recomendamos que la primera prueba utilice una entrada vacía, como se muestra en el siguiente ejemplo.

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

Para las pruebas posteriores, añada los datos de entrada a la prueba.

Obligatorio: sí

`expectations`

El resultado esperado cuando se evalúan reglas específicas en función de los datos de entrada. Especifique una o varias reglas que desee probar además del resultado esperado para cada regla. El resultado esperado debe ser uno de los siguientes:

- `PASS`— Cuando se comparan con los datos de entrada, las reglas se evalúan como`true`.
- `FAIL`— Cuando se comparan con los datos de entrada, las reglas se evalúan como`false`.
- `SKIP`— Cuando se ejecuta con los datos de entrada, la regla no se activa.

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

Obligatorio: sí

Tutorial sobre cómo escribir un archivo de pruebas unitarias de Guard Rules

El siguiente es un archivo de reglas llamado`api_gateway_private.guard`. El objetivo de esta regla es comprobar si todos los tipos de recursos de Amazon API Gateway definidos en una

CloudFormation plantilla se implementan únicamente para acceso privado. También comprueba si al menos una declaración de política permite el acceso desde una nube privada virtual (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
            }
        }
    }
}
```

Este tutorial pone a prueba la intención de la primera regla: todos los `AWS::ApiGateway::RestApi` recursos desplegados deben ser privados.

1. Cree un archivo de pruebas unitarias denominado `api_gateway_private_tests.yaml` que contenga la siguiente prueba inicial. Con la prueba inicial, agrega una entrada vacía y espera que la regla `check_rest_api_is_private` se omita porque no hay `AWS::ApiGateway::RestApi` recursos como entradas.

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

2. Ejecuta la primera prueba en tu terminal con el `test` comando. Para el `--rules-file` parámetro, especifique su archivo de reglas. Para el `--test-data` parámetro, especifique el archivo de pruebas unitarias.

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

   El resultado de la primera prueba es`PASS`.

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

3. Agregue otra prueba a su archivo de pruebas unitarias. Ahora, amplíe las pruebas para incluir los recursos vacíos. El siguiente es el `api_gateway_private_tests.yaml` archivo actualizado.

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

4. Ejecute `test` con el archivo de pruebas unitarias actualizado.

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

El resultado de la segunda prueba es`PASS`.

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

5. Agregue dos pruebas más al archivo de pruebas unitarias. Amplíe las pruebas para incluir lo siguiente:

   - `AWS::ApiGateway::RestApi`Recurso sin propiedades especificadas.

     > **Note**
     >
     > No es una CloudFormation plantilla válida, pero es útil para comprobar si la regla funciona correctamente incluso con entradas con formato incorrecto.

Es de esperar que esta prueba falle porque la `EndpointConfiguration` propiedad no está especificada y, por lo tanto, no está configurada en. `PRIVATE`

- Un `AWS::ApiGateway::RestApi` recurso que cumple con la primera intención con la `EndpointConfiguration` propiedad establecida en`PRIVATE`, pero no cumple con la segunda porque no tiene ninguna declaración de política definida. Espere que esta prueba pase.

El siguiente es el archivo de pruebas unitarias actualizado.

```
---
- 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. Ejecute `test` con el archivo de pruebas unitarias actualizado.

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

El tercer resultado es `FAIL` y el cuarto resultado es`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. Comenta las pruebas 1 a 3 en tu archivo de pruebas unitarias. Acceda al contexto detallado solo para la cuarta prueba. El siguiente es el archivo de pruebas unitarias actualizado.

```
---
#- 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. Inspeccione los resultados de la evaluación ejecutando el `test` comando en su terminal, utilizando el `--verbose` indicador. El contexto detallado es útil para entender las evaluaciones. En este caso, proporciona información detallada sobre por qué la cuarta prueba tuvo éxito y obtuvo un PASS resultado.

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

Este es el resultado de esa ejecución.

```
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)
```

La observación clave del resultado es la línea`Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`, que indica que la comprobación ha sido aprobada. El ejemplo también mostró el caso en el que se `Types` esperaba que fuera una matriz, pero se proporcionó un único valor. En ese caso, Guard continuó la evaluación y proporcionó un resultado correcto.

9. Agregue un caso de prueba como el cuarto caso de prueba a su archivo de pruebas unitarias para un `AWS::ApiGateway::RestApi` recurso con la `EndpointConfiguration` propiedad especificada. El caso de prueba fallará en lugar de aprobarse. El siguiente es el archivo de pruebas unitarias actualizado.

```
---
#- 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. Ejecute el `test` comando con el archivo de pruebas unitarias actualizado utilizando el `--verbose` indicador.

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

El resultado es `FAIL` el esperado porque `REGIONAL` se ha especificado para`EndpointConfiguration`, pero no se espera.

```
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)
```

El resultado detallado del `test` comando sigue la estructura del archivo de reglas. Cada bloque del archivo de reglas es un bloque de la salida detallada. El bloque más alto es cada regla. Si hay `when` condiciones que no cumplan con la regla, aparecen en un bloque de condiciones similar. En el siguiente ejemplo, la condición `%api_gws !empty` se prueba y se aprueba.

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

Una vez que se cumple la condición, probamos las cláusulas de la regla.

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

`%api_gws`es una regla de bloqueo que corresponde al `BlockClause` nivel de la salida (línea:21). La cláusula de regla es un conjunto de cláusulas de conjunción (AND), donde cada cláusula de conjunción es un conjunto de disyunciones. `OR` La conjunción tiene una sola cláusula,. `Properties.EndpointConfiguration.Types[*] == "PRIVATE"` Por lo tanto, el resultado detallado muestra una sola cláusula. La ruta `/Resources/apiGw/Properties/EndpointConfiguration/Types/1` muestra qué valores de la entrada se comparan, que en este caso es el elemento `Types` indexado en 1.

En[Validar los datos de entrada según las reglas de Guard](), puede usar los ejemplos de esta sección para usar el `validate` comando para evaluar los datos de entrada según las reglas.

Validación de los datos de entrada según las reglas AWS CloudFormation Guard

Puede usar el AWS CloudFormation Guard `validate` comando para validar los datos según las reglas de Guard. Para obtener más información sobre el `validate` comando, incluidos sus parámetros y opciones, consulte validar.

Requisitos previos

- Escribe reglas de Guard para validar los datos de entrada con ellas. Para obtener más información, consulte Reglas de Writing Guard.

- Pon a prueba tus reglas para asegurarte de que funcionan según lo previsto. Para obtener más información, consulte Reglas de Testing Guard.

Uso del comando **validate**

Para validar los datos de entrada con arreglo a las reglas de Guard, como una AWS CloudFormation plantilla, ejecuta el `validate` comando Guard. Para el `--rules` parámetro, especifique el nombre de un archivo de reglas. Para el `--data` parámetro, especifique el nombre del archivo de datos de entrada.

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

Si Guard valida correctamente las plantillas, el `validate` comando devuelve un estado de salida de `0` (`$?`en bash). Si Guard identifica una infracción de las reglas, el `validate` comando devuelve un informe de estado de las reglas que han fallado. Utilice el indicador de resumen (`-s all`) para ver el árbol de evaluación detallado que muestra cómo Guard evaluó cada regla.

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

Validar varias reglas con varios archivos de datos

Para ayudar a mantener las reglas, puede escribirlas en varios archivos y organizarlas como desee. A continuación, puede validar varios archivos de reglas comparándolos con uno o varios archivos de datos. El `validate` comando puede tomar un directorio de archivos para las `--rules` opciones `--data` y. Por ejemplo, puede ejecutar el siguiente comando donde `/path/to/dataDirectory` contiene uno o más archivos de datos y `/path/to/ruleDirectory` uno o más archivos de reglas.

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

Puede escribir reglas para comprobar si los distintos recursos definidos en varias CloudFormation plantillas tienen las asignaciones de propiedades adecuadas para garantizar el cifrado en reposo. Para facilitar la búsqueda y el mantenimiento, puede disponer de reglas para comprobar el cifrado en reposo en cada recurso en archivos independientes`s3_bucket_encryption.guard`,

denominadosec2_volume_encryption.guard, y rds_dbinstance_encrytion.guard en un directorio con la ruta~/GuardRules/encryption_at_rest. Las CloudFormation plantillas que necesita validar se encuentran en un directorio con la ruta~/CloudFormation/templates. En este caso, ejecute el validate comando de la siguiente manera.

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

Solución de problemas AWS CloudFormation Guard

Si encuentra problemas al trabajar con él AWS CloudFormation Guard, consulte los temas de esta sección.

Temas

La cláusula falla cuando no hay recursos del tipo seleccionado

Cuando una consulta usa un filtro, por ejemplo`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`, si no hay `AWS::ApiGateway::RestApi` recursos en la entrada, la cláusula se evalúa como tal. `FAIL`

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

Para evitar este resultado, asigne filtros a las variables y utilice la comprobación de `when` condiciones.

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

Guard no evalúa la CloudFormation plantilla con referencias abreviadas Fn::GetAtt

Guard no admite las formas abreviadas de funciones intrínsecas. Por ejemplo, no se admite el uso YAML de`!Join`, `!Sub` en una AWS CloudFormation plantilla con formato. En su lugar, utilice las formas expandidas de las funciones CloudFormation intrínsecas. Por ejemplo`Fn::Join`, utilice CloudFormation plantillas con YAML formato «`Fn::Subin`» al evaluarlas según las reglas de Guard.

Para obtener más información sobre las funciones intrínsecas, consulte la referencia sobre las funciones intrínsecas en la Guía del AWS CloudFormation usuario.

Temas generales de solución de problemas

- Compruebe que `string` los literales no contengan cadenas de escape incrustadas. Actualmente, Guard no admite cadenas de escape integradas en los `string` literales.
- Compruebe que sus `!=` comparaciones comparen tipos de datos compatibles. Por ejemplo, a `string` y an no `int` son tipos de datos compatibles para la comparación. Al realizar la `!=` comparación, si los valores son incompatibles, se produce un error interno. Actualmente, el error se suprime y se convierte `false` para cumplir con la PartialEqcaracterística en Rust.

AWS CloudFormation Guard CLIparámetros y referencia de comandos

Los siguientes parámetros y comandos globales están disponibles a través de la interfaz de línea de AWS CloudFormation Guard comandos (CLI).

Temas

Proteja los parámetros CLI globales

Puede utilizar los siguientes parámetros con cualquier AWS CloudFormation Guard CLI comando.

`-h`, `--help`

Imprime información de ayuda.

`-V`, `--version`

Imprime la información de la versión.

árbol de análisis

Genera un árbol de análisis para las AWS CloudFormation Guard reglas definidas en un archivo de reglas.

Sintaxis

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

Parámetros

`-h, --help`

Imprime información de ayuda.

`-j, --print-json`

Imprime el resultado en JSON formato.

`-y, --print-yaml`

Imprime la salida en YAML formato.

`-V, --version`

Imprime la información de la versión.

Opciones

`-o, --output`

Escribe el árbol generado en un archivo de salida.

`-r, --rules`

Proporciona un archivo de reglas.

Ejemplos

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

regla

Toma un archivo de AWS CloudFormation plantilla YAML con formato JSON - o - y genera automáticamente un conjunto de AWS CloudFormation Guard reglas que coinciden con las propiedades de los recursos de la plantilla. Este comando es una forma útil de empezar a escribir reglas o de crear ready-to-use reglas a partir de plantillas de funcionalidad conocida.

Sintaxis

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

Parámetros

`-h`, `--help`

Imprime información de ayuda.

`-V`, `--version`

Imprime la información de la versión.

Opciones

`-o`, `--output`

Escribe las reglas generadas en un archivo de salida. Dada la posibilidad de que surjan cientos o incluso miles de reglas, recomendamos usar esta opción.

`-t`, `--template`

Proporciona la ruta a un archivo CloudFormation de plantilla en JSON nuestro YAML formato.

Ejemplos

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

prueba

Valida un archivo de AWS CloudFormation Guard reglas comparándolo con un archivo de pruebas de una unidad de guardia en JSON YAML nuestro formato para determinar el éxito de las reglas individuales.

Sintaxis

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

Parámetros

`-h`, `--help`

Imprime información de ayuda.

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

Ordena por la hora de la última modificación dentro de un directorio

`-V`, `--version`

Imprime la información de la versión.

`-v`, `--verbose`

Aumenta la verbosidad de la salida. Se puede especificar varias veces.

El resultado detallado sigue la estructura del archivo de reglas de Guard. Cada bloque del archivo de reglas es un bloque de la salida detallada. El bloque más alto es cada regla. Si hay `when` condiciones que no cumplan con la regla, aparecen como un bloque de condiciones similar.

Opciones

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

Proporciona el nombre de un archivo de reglas.

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

Proporciona el nombre de un archivo o directorio para los archivos de datos en cualquier JSON YAML formato.

args

<alphabetical>

Ordena alfabéticamente dentro de un directorio.

Ejemplos

```
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
```

Véase también

Reglas de Testing Guard

validar

Valida los datos según AWS CloudFormation Guard las reglas para determinar el éxito o el fracaso.

Sintaxis

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

Parámetros

`-a, --alphabetical`

Valida los archivos de un directorio ordenado alfabéticamente.

`-h, --help`

Imprime información de ayuda.

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

Valida los archivos de un directorio que está ordenado por la hora de la última modificación.

`-P`, `--payload`

Permite proporcionar reglas y datos en el siguiente JSON formato mediante: `stdin`

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

Por ejemplo:

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

En «reglas», especifique una lista de cadenas de archivos de reglas. Para «datos», especifique una lista de cadenas de archivos de datos.

Si especifica la `--payload` marca, no especifique las `--data` opciones `--rules` u.

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

Imprime la salida en JSON formato.

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

Muestra el error de la cláusula, incluido un resumen.

`-V`, `--version`

Imprime la información de la versión.

`-v`, `--verbose`

Aumenta la verbosidad de la salida. Se puede especificar varias veces.

Opciones

`-d`, `--data` (cadena)

Proporciona el nombre de un archivo o directorio para los archivos de datos en cualquier JSON YAML formato. Si proporciona un directorio, Guard evalúa las reglas especificadas comparándolas con todos los archivos de datos del directorio. El directorio debe contener solo archivos de datos; no puede contener archivos de datos y de reglas.

Si especifica la `--payload` marca, no especifique la `--data` opción.

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

Escribe en un archivo de salida.

Valor predeterminado: `single-line-summary`

Valores permitidos: `json` | `yaml` | `single-line-summary`

`-r`, `--rules` (cadena)

Proporciona el nombre de un archivo de reglas o un directorio de archivos de reglas. Si proporciona un directorio, Guard evalúa todas las reglas del directorio comparándolas con los datos especificados. El directorio debe contener solo archivos de reglas; no puede contener archivos de datos y de reglas.

Si especifica la `--payload` marca, no especifique la `--rules` opción.

`--show-summary` (cadena)

Especifica la verbosidad del resumen de evaluación de la regla de Guardia. Si lo especifica`all`, Guard mostrará el resumen completo. Si lo especifica`pass,fail`, Guard solo muestra información resumida de las reglas aprobadas o no. Si lo especifica`none`, Guard no mostrará información resumida. De forma predeterminada, se especifica `all`.

Valores permitidos: `all` | `pass,fail` | `none`

`-t`, `--type` (cadena)

Proporciona el formato de los datos de entrada. Al especificar el tipo de datos de entrada, Guard muestra los nombres lógicos de los recursos de la CloudFormation plantilla en la salida. De forma

predeterminada, Guard muestra las rutas y los valores de las propiedades, como`Property [/Resources/vol2/Properties/Encrypted`.

Valores permitidos: `CFNTemplate`

Ejemplos

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

Output

Si Guard valida correctamente las plantillas, el `validate` comando devuelve un estado de salida de `0` (`$?`en bash). Si Guard identifica una infracción de las reglas, el `validate` comando devuelve un informe de estado de las reglas que han fallado. Usa el indicador detallado (`-v`) para ver el árbol de evaluación detallado que muestra cómo Guard evaluó cada regla.

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

Véase también

Validar los datos de entrada según las reglas de Guard

Seguridad en AWS CloudFormation Guard

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de cumplimiento que se aplican a Guard, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad y .
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables

La siguiente documentación le ayuda a entender cómo aplicar el modelo de responsabilidad compartida al instalar Guard como una AWS Lambda función (cfn-guard-lambda):

- La seguridad en la guía AWS Command Line Interface del usuario
- La seguridad en la guía para AWS Lambda desarrolladores
- La seguridad en la guía AWS Identity and Access Management del usuario

AWS CloudFormation Guard Historial de documentos

En la siguiente tabla se describen las versiones de la documentación de AWS CloudFormation Guard.

- Actualización de la documentación más reciente: 30 de junio de 2023
- Última versión: 3.0.0

Cambio	Descripción	Fecha
Lanzamiento de la versión 3.0.0	La versión 3.0.0 presenta las siguientes mejoras: • Los temas de introducción e instalación se actualizaron para la versión 3.0.0 de Guard. • Se agregaron instrucciones de instalación para Homebrew y Chocolatey. • La información relacionada con la migración de las reglas de Guard se actualizó para reflejar los cambios en la versión 3.0.0 de Guard. • Se agregó un enlace destacado al AWS CloudFormation Guard GitHub repositorio.	30 de junio de 2023
Lanzamiento de la versión 2.1.3	La versión 2.1.3 presenta las siguientes mejoras: Se ha agregado información sobre las mejoras de Guard 2.1.3. Las referencias a Guard	9 de junio de 2023

	2.0 se han actualizado a Guard 2.1.3.	
Lanzamiento de la versión 2.0.4	La versión 2.0.4 presenta las siguientes mejoras: Se agregó la `--payload` bandera al `validate` comando. Para obtener más información, consulte validar en la CLI referencia de Guard.	19 de octubre de 2021

Lanzamiento de la versión 2.0.3	La versión 2.0.3 presenta las siguientes mejoras:
	
	• Puede proporcionar nombres de prueba para cada prueba en su archivo de pruebas unitarias. Para obtener más información, consulte Reglas de Testing Guard.
	• Se agregaron las siguientes opciones al `validate` comando:
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	
	  Para obtener más información, consulte validar en la CLI referencia de Guard.
		27 de julio de 2021

Versión inicial	Versión inicial de la Guía AWS CloudFormation Guard del usuario.	15 de julio de 2021

AWS Glosario

Para obtener la AWS terminología más reciente, consulte el AWS glosario de la Glosario de AWS Referencia.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.