aws

Guide de l’utilisateur

AWS CloudFormation Guard

AWS CloudFormation Guard: Guide de l’utilisateur

Table of Contents

Qu'est-ce que c'est AWS CloudFormation Guard ?

AWS CloudFormation Guard est un outil d'évaluation open source à usage général. policy-as-code L'interface de ligne de commande (CLI) Guard fournit un simple-to-use langage déclaratif spécifique au domaine (DSL) que vous pouvez utiliser pour exprimer la politique sous forme de code. En outre, vous pouvez utiliser les commandes CLI pour valider les données JSON ou YAML hiérarchiques structurées par rapport à ces règles. Guard fournit également un cadre de test unitaire intégré pour vérifier que vos règles fonctionnent comme prévu.

Guard ne valide pas les CloudFormation modèles pour une syntaxe valide ou des valeurs de propriétés autorisées. Vous pouvez utiliser l'outil cfn-lint pour effectuer une inspection approfondie de la structure du modèle.

Guard ne fournit pas d'application côté serveur. Vous pouvez utiliser les CloudFormation Hooks pour effectuer une validation et une application côté serveur, où vous pouvez bloquer ou avertir une opération.

Pour des informations détaillées sur AWS CloudFormation Guard le développement, consultez le GitHub référentiel Guard.

Rubriques

Utilisez-vous Guard pour la première fois ?

Si vous utilisez Guard pour la première fois, nous vous recommandons de commencer par lire les sections suivantes :

- Configuration de Guard— Cette section décrit comment installer Guard. Avec Guard, vous pouvez écrire des règles de politique à l'aide du Guard DSL et valider vos données structurées au format JSON ou YAML par rapport à ces règles.

- Règles de Writing Guard— Cette section fournit des instructions détaillées pour la rédaction des règles de politique.
- Règles de Testing Guard— Cette section fournit une procédure détaillée pour tester vos règles afin de vérifier qu'elles fonctionnent comme prévu, et pour valider vos données structurées au format JSON ou YAML par rapport à vos règles.
- Validation des données d'entrée par rapport aux règles Guard— Cette section fournit une procédure détaillée pour valider vos données structurées au format JSON ou YAML par rapport à vos règles.
- Référence GUARD CLI— Cette section décrit les commandes disponibles dans la GUARD CLI.

Caractéristiques de la protection

À l'aide de Guard, vous pouvez rédiger des règles de politique pour valider toutes les données structurées au format JSON ou YAML, y compris, mais sans s'y limiter, les modèles. CloudFormation Guard prend en charge l'ensemble du spectre de end-to-end l'évaluation des contrôles des politiques. Les règles sont utiles dans les domaines commerciaux suivants :

- Gouvernance préventive et conformité (tests shift-left) : validez l'infrastructure sous forme de code (IaC) ou les compositions d'infrastructure et de services par rapport aux règles de politique qui représentent les meilleures pratiques de votre organisation en matière de sécurité et de conformité. Par exemple, vous pouvez valider des CloudFormation modèles, des ensembles de CloudFormation modifications, des fichiers de configuration Terraform basés sur JSON ou des configurations Kubernetes.
- Detective Governance et conformité : validez la conformité des ressources de la base de données de gestion des configurations (CMDB), telles que les éléments de configuration AWS Config basés sur les éléments (CIs). Par exemple, les développeurs peuvent utiliser les politiques Guard AWS Config CIs pour surveiller en permanence l'état des AWS ressources déployées AWS et non déployées, détecter les violations des politiques et lancer des mesures correctives.
- Sécurité du déploiement : assurez-vous que les modifications sont sécurisées avant le déploiement. Par exemple, validez les ensembles de CloudFormation modifications par rapport aux règles de politique afin d'empêcher les modifications entraînant le remplacement de ressources, telles que le changement de nom d'une table Amazon DynamoDB.

Utiliser Guard with CloudFormation Hooks

Vous pouvez utiliser CloudFormation Guard pour créer un Hook in CloudFormation Hooks. CloudFormation Hooks vous permet d'appliquer de manière proactive vos règles Guard avant de CloudFormation créer, de mettre à jour ou de supprimer des opérations et de API de commande du Cloud AWS créer ou de mettre à jour des opérations. Les Hooks garantissent que les configurations de vos ressources sont conformes aux meilleures pratiques de votre organisation en matière de sécurité, d'exploitation et d'optimisation des coûts.

Pour plus de détails sur la façon d'utiliser Guard pour créer des crochets de CloudFormation garde, consultez la section Write Guard rules pour évaluer les ressources relatives à Guard Hooks dans le guide de l'utilisateur CloudFormation des Hooks.

Access Guard

Pour accéder au Guard DSL et aux commandes, vous devez installer le Guard CLI. Pour plus d'informations sur l'installation de la CLI Guard, consultezConfiguration de Guard.

Bonnes pratiques

Rédigez des règles simples et utilisez des règles nommées pour les référencer dans d'autres règles. Les règles complexes peuvent être difficiles à maintenir et à tester.

Con AWS CloudFormation Guard figuration

AWS CloudFormation Guard est une interface de ligne de commande (CLI) open source. Il vous fournit un langage simple et spécifique au domaine pour écrire des règles de politique et valider leurs données JSON et YAML hiérarchiques structurées par rapport à ces règles. Les règles peuvent représenter les directives de politique de l'entreprise relatives à la sécurité, à la conformité, etc. Les données hiérarchiques structurées peuvent représenter une infrastructure cloud décrite sous forme de code. Par exemple, vous pouvez créer des règles pour garantir qu'ils modélisent toujours des buckets Amazon Simple Storage Service (Amazon S3) chiffrés dans leurs modèles. CloudFormation

Les rubriques suivantes fournissent des informations sur l'installation de Guard à l'aide du système d'exploitation que vous avez choisi ou en tant que AWS Lambda fonction.

Rubriques

Installation de Guard pour Linux et macOS

Vous pouvez l'installer AWS CloudFormation Guard pour Linux et macOS à l'aide de la version binaire prédéfinie, Cargo, ou via Homebrew.

Installer Guard à partir d'une version binaire prédéfinie

Utilisez la procédure suivante pour installer Guard à partir d'un fichier binaire prédéfini.

1. Ouvrez un terminal et exécutez la commande suivante.

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

2. Exécutez la commande suivante pour définir votre PATH variable.

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

   Résultats : Vous avez correctement installé Guard et défini la PATH variable.

- (Facultatif) Pour confirmer l'installation de Guard, exécutez la commande suivante.

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

La commande renvoie le résultat suivant.

```
cfn-guard 3.1.2
```

Installez Guard from Cargo

Cargo est le gestionnaire de paquets de Rust. Procédez comme suit pour installer Rust, qui inclut Cargo. Ensuite, installez Guard from Cargo.

1. Exécutez la commande suivante depuis un terminal et suivez les instructions affichées à l'écran pour installer Rust.

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

- (Facultatif) Pour les environnements Ubuntu, exécutez la commande suivante.

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

2. Configurez votre variable d'PATHenvironnement et exécutez la commande suivante.

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

3. Une fois Cargo installé, exécutez la commande suivante pour installer Guard.

```
cargo install cfn-guard
```

Résultats : Vous avez correctement installé Guard.

- (Facultatif) Pour confirmer l'installation de Guard, exécutez la commande suivante.

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

La commande renvoie le résultat suivant.

```
cfn-guard 3.1.2
```

Installez Guard depuis Homebrew

Homebrew est un gestionnaire de paquets pour macOS et Linux. Procédez comme suit pour installer Homebrew. Ensuite, installez Guard depuis Homebrew.

1. Exécutez la commande suivante depuis un terminal et suivez les instructions affichées à l'écran pour installer Homebrew.

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

2. Une fois Homebrew installé, exécutez la commande suivante pour installer Guard.

   ```
   brew install cloudformation-guard
   ```

   Résultats : Vous avez correctement installé Guard.

   - (Facultatif) Pour confirmer l'installation de Guard, exécutez la commande suivante.

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

     La commande renvoie le résultat suivant.

     ```
     cfn-guard 3.1.2
     ```

Installation de Guard pour Windows

Vous pouvez l'installer AWS CloudFormation Guard pour Windows via Cargo ou Chocolatey.

Conditions préalables

Pour créer Guard à partir de l'interface de ligne de commande, vous devez installer les outils de génération pour Visual Studio 2019.

1. Téléchargez les outils de génération Microsoft Visual C++ depuis le site Web des outils de génération pour Visual Studio 2019.
2. Exécutez le programme d'installation et sélectionnez les valeurs par défaut.

Installez Guard from Cargo

Cargo est le gestionnaire de paquets de Rust. Procédez comme suit pour installer Rust, qui inclut Cargo. Ensuite, installez Guard from Cargo.

1. Téléchargez Rust, puis exécutez rustup-init.exe.
2. À l'invite de commande, choisissez 1, qui est l'option par défaut.

   La commande renvoie le résultat suivant.

   ```
   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. Pour terminer l'installation, appuyez sur la touche Entrée.
4. Une fois Cargo installé, exécutez la commande suivante pour installer Guard.

   ```
   cargo install cfn-guard
   ```

   Résultats : Vous avez correctement installé Guard.

   - (Facultatif) Pour confirmer l'installation de Guard, exécutez la commande suivante.

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

     La commande renvoie le résultat suivant.

     ```
     cfn-guard 3.1.2
     ```

Installez Guard de Chocolatey

Chocolatey est un gestionnaire de paquets pour Windows. Procédez comme suit pour installer Chocolatey. Ensuite, installez Guard de Chocolatey.

1. Suivez ce guide pour installer Chocolatey
2. Une fois Chocolatey installé, exécutez la commande suivante pour installer Guard.

   ```
   choco install cloudformation-guard
   ```

   Résultats : Vous avez correctement installé Guard.

   - (Facultatif) Pour confirmer l'installation de Guard, exécutez la commande suivante.

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

     La commande renvoie le résultat suivant.

     ```
     cfn-guard 3.1.2
     ```

Installation de Guard en tant que AWS Lambda fonction

Vous pouvez l'installer AWS CloudFormation Guard via Cargo, le gestionnaire de paquets de Rust. Guard as an AWS Lambda function (`cfn-guard-lambda`) est une enveloppe légère autour de Guard (`cfn-guard`) qui peut être utilisée comme fonction Lambda.

Conditions préalables

Avant de pouvoir installer Guard en tant que fonction Lambda, vous devez remplir les conditions préalables suivantes :

- AWS Command Line Interface (AWS CLI) configuré avec les autorisations nécessaires pour déployer et invoquer des fonctions Lambda. Pour plus d'informations, consultez Configuration de l'AWS CLI.
- Un rôle AWS Lambda d'exécution dans Gestion des identités et des accès AWS (IAM). Pour plus d'informations, consultez la section rôle AWS Lambda d'exécution.

- Dans CentOS/RHEL les environnements, ajoutez le référentiel de `musl-libc` packages à votre configuration yum. Pour plus d'informations, consultez ngompa/musl-libc.

Installez le gestionnaire de paquets Rust

Cargo est le gestionnaire de paquets de Rust. Procédez comme suit pour installer Rust, qui inclut Cargo.

1. Exécutez la commande suivante depuis un terminal, puis suivez les instructions à l'écran pour installer Rust.

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

   - (Facultatif) Pour les environnements Ubuntu, exécutez la commande suivante.

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

2. Configurez votre variable d'`PATH`environnement et exécutez la commande suivante.

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

Installez Guard en tant que fonction Lambda (Linux, macOS ou Unix)

Pour installer Guard en tant que fonction Lambda, procédez comme suit.

1. Depuis votre terminal de commande, exécutez la commande suivante.

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

   - (Facultatif) Pour confirmer l'installation de Guard en tant que fonction Lambda, exécutez la commande suivante.

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

     La commande renvoie le résultat suivant.

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

2. Pour installer `musl` le support, exécutez la commande suivante.

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

3. Compilez avec`musl`, puis exécutez la commande suivante dans votre terminal.

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

Pour un environnement d'exécution personnalisé, AWS Lambda nécessite un exécutable dont le nom figure `bootstrap` dans le fichier .zip du package de déploiement. Renommez le `cfn-lambda` fichier exécutable généré en, `bootstrap` puis ajoutez-le à l'archive .zip.

- Pour les environnements macOS, créez votre fichier de configuration de chargement à la racine du projet Rust ou dans`~/.cargo/config`.

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

4. Accédez au répertoire `cfn-guard-lambda` racine.

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

5. Exécutez la commande suivante dans votre terminal.

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

6. Exécutez la commande suivante pour l'envoyer `cfn-guard` en tant que fonction Lambda à votre compte.

```
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
```

Pour créer et exécuter Guard en tant que fonction Lambda

Pour appeler la fonction soumise `cfn-guard-lambda` en tant que fonction Lambda, exécutez la commande suivante.

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

Pour appeler la structure de demande de la fonction Lambda

Demandes visant à `cfn-guard-lambda` exiger les champs suivants :

- `data`— La version sous forme de chaîne du modèle YAML ou JSON
- `rules`— La version sous forme de chaîne du fichier d'ensemble de règles

Conditions préalables et présentation de l'utilisation des règles Guard

Cette section explique comment effectuer les tâches principales de Guard, à savoir l'écriture, le test et la validation de règles par rapport à des données au format JSON ou YAML. En outre, il contient des procédures pas à pas détaillées illustrant les règles d'écriture qui répondent à des cas d'utilisation spécifiques.

Rubriques

Conditions préalables

Avant de pouvoir écrire des règles de politique à l'aide du langage spécifique au domaine Guard (DSL), vous devez installer l'interface de ligne de commande (CLI) Guard. Pour de plus amples informations, veuillez consulter Configuration de Guard.

Présentation de l'utilisation des règles Guard

Lorsque vous utilisez Guard, vous effectuez généralement les étapes suivantes :

1. Écrivez des données au format JSON ou YAML pour les valider.
2. Règles de politique de Write Guard. Pour de plus amples informations, veuillez consulter Règles de Writing Guard.
3. Vérifiez que vos règles fonctionnent comme prévu à l'aide de la `test` commande Guard. Pour plus d'informations sur les tests unitaires, consultezRègles de Testing Guard.
4. Utilisez la `validate` commande Guard pour valider vos données au format JSON ou YAML par rapport à vos règles. Pour de plus amples informations, veuillez consulter Validation des données d'entrée par rapport aux règles Guard.

AWS CloudFormation Guard Règles d'écriture

Dans AWS CloudFormation Guard, les règles sont les policy-as-code règles. Vous rédigez des règles dans le langage spécifique au domaine Guard (DSL) par rapport auxquelles vous pouvez valider vos données au format JSON ou YAML. Les règles sont composées de clauses.

Vous pouvez enregistrer les règles écrites à l'aide du Guard DSL dans des fichiers en texte brut qui utilisent n'importe quelle extension de fichier.

Vous pouvez créer plusieurs fichiers de règles et les classer en tant qu'ensemble de règles. Les ensembles de règles vous permettent de valider vos données au format JSON ou YAML par rapport à plusieurs fichiers de règles en même temps.

Rubriques

Clauses

Les clauses sont des expressions booléennes dont la valeur est vraie (`PASS`) ou fausse (`FAIL`). Les clauses utilisent soit des opérateurs binaires pour comparer deux valeurs, soit des opérateurs unaires qui agissent sur une seule valeur.

Exemples de clauses unaires

La clause unaire suivante évalue si la collection `TcpBlockedPorts` est vide.

```
InputParameters.TcpBlockedPorts not empty
```

La clause unaire suivante détermine si la `ExecutionRoleArn` propriété est une chaîne.

```
Properties.ExecutionRoleArn is_string
```

Exemples de clauses binaires

La clause binaire suivante évalue si la `BucketName` propriété contient la chaîne`encrypted`, quel que soit le casier.

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

La clause binaire suivante évalue si la `ReadCapacityUnits` propriété est inférieure ou égale à 5 000.

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

Syntaxe pour écrire les clauses des règles Guard

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

Propriétés des clauses de la règle Guard

`query`

Expression séparée par des points (`.`) écrite pour parcourir des données hiérarchiques. Les expressions de requête peuvent inclure des expressions de filtre pour cibler un sous-ensemble de valeurs. Les requêtes peuvent être attribuées à des variables afin que vous puissiez les écrire une seule fois et les référencer ailleurs dans un ensemble de règles, ce qui vous permettra d'accéder aux résultats des requêtes.

Pour plus d'informations sur la rédaction de requêtes et le filtrage, consultezDéfinition des requêtes et filtrage.

Obligatoire : oui

`operator`

Opérateur unaire ou binaire qui permet de vérifier l'état de la requête. Le côté gauche (LHS) d'un opérateur binaire doit être une requête, et le côté droit (RHS) doit être une requête ou une valeur littérale.

Opérateurs binaires pris en charge : == `!`= (Égal) | > (Non égal) | >= (Supérieur à) | (Supérieur ou égal à) | < (Inférieur à) | <= (Inférieur ou égal à) | `IN` (Dans une liste au format [x, y, z]

Opérateurs unaires pris en charge : `exists` | `empty` | `is_string` | | `is_list` | `is_struct` `not(!)`

Obligatoire : oui

`query|value literal`

Une requête ou un littéral de valeur pris en charge tel que `string` ou`integer(64)`.

Litéraux de valeur pris en charge :

- Tous les types primitifs : `string`integer(64),`float(64)`,`bool`,`char`, `regex`
- Tous les types de plages spécialisés pour exprimer `integer(64)``float(64)`, ou les `char` plages exprimées comme suit :
  - `r[<lower_limit>, <upper_limit>]`, qui se traduit par toute valeur k répondant à l'expression suivante : `lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`, qui se traduit par toute valeur k répondant à l'expression suivante : `lower_limit <= k < upper_limit`
  - `r(<lower_limit>, <upper_limit>]`, qui se traduit par toute valeur k répondant à l'expression suivante : `lower_limit < k <= upper_limit`
  - `r(<lower_limit>, <upper_limit>)`,qui se traduit par toute valeur k répondant à l'expression suivante : `lower_limit < k < upper_limit`
- Tableaux associatifs (cartes) pour les données de structure clé-valeur imbriquées. Par exemple :

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

- Tableaux de types primitifs ou de types de tableaux associatifs

Obligatoire : conditionnel ; obligatoire lorsqu'un opérateur binaire est utilisé.

`custom message`

Chaîne fournissant des informations sur la clause. Le message s'affiche dans les sorties détaillées des `test` commandes `validate` et et peut être utile pour comprendre ou débugger l'évaluation des règles sur des données hiérarchiques.

Obligatoire : non

Utilisation de requêtes dans les clauses

Pour plus d'informations sur la rédaction de requêtes, reportez-vous Définition des requêtes et filtrage aux sections etAffectation et référencement de variables dans les règles Guard.

Utilisation d'opérateurs dans les clauses

Voici des exemples de CloudFormation modèles, `Template-1` et`Template-2`. Pour démontrer l'utilisation des opérateurs pris en charge, les exemples de requêtes et de clauses de cette section font référence à ces exemples de modèles.

Modèle-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
```

Modèle-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
```

Exemples de clauses utilisant des opérateurs unaires

- `empty`— Vérifie si une collection est vide. Vous pouvez également l'utiliser pour vérifier si une requête contient des valeurs dans des données hiérarchiques, car les requêtes aboutissent à une collection. Vous ne pouvez pas l'utiliser pour vérifier si une chaîne vide (`""`) est définie dans les requêtes contenant des valeurs de chaîne. Pour de plus amples informations, veuillez consulter Définition des requêtes et filtrage.

  La clause suivante vérifie si une ou plusieurs ressources sont définies dans le modèle. Il est évalué comme PASS étant donné qu'une ressource avec l'ID logique `S3Bucket` est définie dans`Template-1`.

  ```
  Resources !empty
  ```

  La clause suivante vérifie si une ou plusieurs balises sont définies pour la `S3Bucket` ressource. Il est évalué à PASS parce `S3Bucket` que deux balises sont définies pour la `Tags` propriété dans`Template-1`.

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

- `exists`— Vérifie si chaque occurrence de la requête possède une valeur et peut être utilisée à la place de`!= null`.

  La clause suivante vérifie si la `BucketEncryption` propriété est définie pour`S3Bucket`. Il est évalué à PASS parce qu'il `BucketEncryption` est défini pour `S3Bucket` dans`Template-1`.

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

> **Note**
>
> Les `not exists` contrôles `empty` et évaluent la présence `true` de clés de propriété manquantes lors de la traversée des données d'entrée. Par exemple, si la `Properties` section n'est pas définie dans le modèle pour le`S3Bucket`, la clause est `Resources.S3Bucket.Properties.Tag empty` évaluée à`true`. Les `empty` vérifications `exists` et n'affichent pas le chemin du pointeur JSON à l'intérieur du document dans les messages d'erreur. Ces deux clauses comportent souvent des erreurs de récupération qui ne tiennent pas compte de ces informations de traversée.

- `is_string`— Vérifie si chaque occurrence de la requête est de `string` type.

  La clause suivante vérifie si une valeur de chaîne est spécifiée pour la `BucketName` propriété de la `S3Bucket` ressource. Il est évalué à PASS parce que la valeur de chaîne `"MyServiceS3Bucket"` est spécifiée `BucketName` dans`Template-1`.

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

- `is_list`— Vérifie si chaque occurrence de la requête est de `list` type.

  La clause suivante vérifie si une liste est spécifiée pour la `Tags` propriété de la `S3Bucket` ressource. Il est évalué à PASS parce que deux paires clé-valeur sont spécifiées pour in. `Tags` `Template-1`

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

- `is_struct`— Vérifie si chaque occurrence de la requête est une donnée structurée.

  La clause suivante vérifie si des données structurées sont spécifiées pour la `BucketEncryption` propriété de la `S3Bucket` ressource. Il est évalué à PASS parce qu'il `BucketEncryption` est spécifié à l'aide du type de `ServerSideEncryptionConfiguration` propriété *`(object)`* in`Template-1`.

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

> **Note**
>
> Pour vérifier l'état inverse, vous pouvez utiliser l'opérateur (`not !`) avec les `is_struct` opérateurs `is_string`is_list, et.

Exemples de clauses utilisant des opérateurs binaires

La clause suivante vérifie si la valeur spécifiée pour la `BucketName` propriété de la `S3Bucket` ressource `Template-1` contient la chaîne`encrypt`, quel que soit le casier. Cela se traduit par le PASS fait que le nom du compartiment spécifié `"MyServiceS3Bucket"` ne contient pas la chaîne`encrypt`.

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

La clause suivante vérifie si la valeur spécifiée pour la `Size` propriété de la `NewVolume` ressource `Template-2` se situe dans une plage spécifique : 50 **<=** `Size` **<=** 200. Il est évalué à `PASS` parce que `100` c'est spécifié pour`Size`.

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

La clause suivante vérifie si la valeur spécifiée pour la `VolumeType` propriété de la `NewVolume` ressource dans `Template-2` est `io1``io2`, ou`gp3`. Il est évalué à `PASS` parce que `io1` c'est spécifié pour`NewVolume`.

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

> **Note**
>
> Les exemples de requêtes présentés dans cette section illustrent l'utilisation d'opérateurs utilisant les ressources avec un caractère logique IDs `S3Bucket` et`NewVolume`. Les noms de ressources sont souvent définis par l'utilisateur et peuvent être nommés arbitrairement dans un modèle d'infrastructure en tant que code (IaC). Pour écrire une règle générique qui s'applique à toutes les `AWS::S3::Bucket` ressources définies dans le modèle, la forme de requête la plus couramment utilisée est`Resources.*[ Type == 'AWS::S3::Bucket' ]`. Pour plus d'informations, consultez Définition des requêtes et

filtrage pour plus de détails sur l'utilisation et explorez le répertoire des exemples dans le `cloudformation-guard` GitHub référentiel.

Utilisation de messages personnalisés dans les clauses

Dans l'exemple suivant, les clauses pour `Template-2` inclure un message personnalisé.

```
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>>
```

Combinaison de clauses

Dans Guard, chaque clause écrite sur une nouvelle ligne est combinée implicitement à la clause suivante en utilisant une conjonction (logique booléenne`and`). Consultez l'exemple suivant.

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

Vous pouvez également utiliser la disjonction pour combiner une clause avec la clause suivante en spécifiant `or|OR` à la fin de la première clause.

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

Dans une clause Guard, les disjonctions sont évaluées en premier, suivies des conjonctions. Les règles de protection peuvent être définies comme une conjonction de disjonctions de clauses (une `and|AND` ou `or|OR` plusieurs) dont l'évaluation correspond à `true` (`PASS`) ou `false` (`FAIL`). Ceci est similaire à la forme normale conjonctive.

Les exemples suivants illustrent l'ordre des évaluations des clauses.

```
# (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
```

Toutes les clauses basées sur cet exemple `Template-1` peuvent être combinées à l'aide d'une conjonction. Consultez l'exemple suivant.

```
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
```

Utiliser des blocs avec les règles de garde

Les blocs sont des compositions qui éliminent la verbosité et la répétition d'un ensemble de clauses, de conditions ou de règles connexes. Il existe trois types de blocs :

- Blocs de requêtes
- `when`blocs
- Blocs de règles nommés

Blocs de requêtes

Les clauses suivantes sont basées sur cet exemple`Template-1`. La conjonction a été utilisée pour combiner les clauses.

```
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
```

Des parties de l'expression de requête de chaque clause sont répétées. Vous pouvez améliorer la composabilité et supprimer la verbosité et la répétition d'un ensemble de clauses associées ayant le même chemin de requête initial en utilisant un bloc de requête. Le même ensemble de clauses peut être écrit comme indiqué dans l'exemple suivant.

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

Dans un bloc de requête, la requête qui précède le bloc définit le contexte des clauses contenues dans le bloc.

Pour plus d'informations sur l'utilisation des blocs, consultezComposer des blocs de règles nommées.

whenblocs

Vous pouvez évaluer les blocs de manière conditionnelle en utilisant des `when` blocs, qui se présentent sous la forme suivante.

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

Le `when` mot clé indique le début du `when` bloc. `condition`est une règle de la Garde. Le bloc n'est évalué que si l'évaluation de la condition aboutit à `true` (PASS).

Voici un exemple de `when` bloc basé sur`Template-1`.

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

La clause contenue dans le `when` bloc n'est évaluée que si la valeur spécifiée pour `BucketName` est une chaîne. Si la valeur spécifiée pour `BucketName` est référencée dans la `Parameters` section du modèle, comme indiqué dans l'exemple suivant, la clause contenue dans le `when` bloc n'est pas évaluée.

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

Blocs de règles nommés

Vous pouvez attribuer un nom à un ensemble de règles (ensemble de règles), puis référencer ces blocs de validation modulaires, appelés blocs de règles nommées, dans d'autres règles. Les blocs de règles nommées se présentent sous la forme suivante.

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

Le `rule` mot clé indique le début du bloc de règles nommées.

`rule name`est une chaîne lisible par l'homme qui identifie de manière unique un bloc de règles nommé. Il s'agit d'une étiquette pour l'ensemble de règles Guard qu'elle encapsule. Dans cette utilisation, le terme règle de garde inclut les clauses, les blocs de requêtes, les `when` blocs et les blocs de règles nommées. Le nom de la règle peut être utilisé pour faire référence au résultat de l'évaluation de l'ensemble de règles qu'il encapsule, ce qui rend les blocs de règles nommés

réutilisables. Le nom de la règle fournit également un contexte sur les échecs des règles dans les sorties de `test` commande `validate` et. Le nom de la règle est affiché avec le statut d'évaluation du bloc (`PASS`,`FAIL`, ou`SKIP`) dans le résultat d'évaluation du fichier de règles. Consultez l'exemple suivant.

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

Vous pouvez également évaluer les blocs de règles nommées de manière conditionnelle en spécifiant le `when` mot clé suivi d'une condition après le nom de la règle.

Voici l'exemple de `when` bloc dont il a été question précédemment dans cette rubrique.

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

En utilisant des blocs de règles nommées, ce qui précède peut également être écrit comme suit.

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

Vous pouvez réutiliser et regrouper des blocs de règles nommés avec d'autres règles Guard. Voici quelques exemples.

```
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
    ...
}
```

Utilisation des fonctions intégrées

AWS CloudFormation Guard fournit des fonctions intégrées que vous pouvez utiliser dans vos règles pour effectuer des opérations telles que la manipulation de chaînes, l'analyse JSON et la conversion de types de données. Les fonctions ne sont prises en charge que par l'affectation à une variable.

Principales fonctions

`json_parse(json_string)`

Analyse les chaînes JSON en ligne à partir d'un modèle. Après l'analyse, vous pouvez évaluer les propriétés de l'objet obtenu.

`count(collection)`

Renvoie le nombre d'éléments auxquels une requête aboutit.

`regex_replace(base_string, regex_to_extract, regex_replacement)`

Remplace des parties d'une chaîne à l'aide d'expressions régulières.

Pour une liste complète des fonctions disponibles, notamment la manipulation de chaînes, les opérations de collecte et les fonctions de conversion de type de données, consultez la documentation des fonctions dans le GitHub référentiel Guard.

Définition des requêtes Guard et filtrage

Cette rubrique traite de la rédaction de requêtes et de l'utilisation du filtrage lors de la rédaction de clauses de règles Guard.

Conditions préalables

Le filtrage est un AWS CloudFormation Guard concept avancé. Nous vous recommandons de consulter les sujets fondamentaux suivants avant de vous familiariser avec le filtrage :

- Qu'est-ce que c'est AWS CloudFormation Guard ?
- Règles de rédaction, clauses

Définition des requêtes

Les expressions de requête sont de simples expressions séparées par des points (`.`) écrites pour parcourir des données hiérarchiques. Les expressions de requête peuvent inclure des expressions de filtre pour cibler un sous-ensemble de valeurs. Lorsque les requêtes sont évaluées, elles aboutissent à un ensemble de valeurs, similaire à un jeu de résultats renvoyé par une requête SQL.

L'exemple de requête suivant recherche des `AWS::IAM::Role` ressources dans un CloudFormation modèle.

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

Les requêtes suivent les principes de base suivants :

- Chaque point (`.`) de la requête traverse la hiérarchie lorsqu'un terme clé explicite est utilisé, par exemple `Resources` ou `Properties.Encrypted.` si une partie de la requête ne correspond pas à la donnée entrante, Guard génère une erreur de récupération.
- Un point (`.`) de la requête qui utilise un caractère générique `*` traverse toutes les valeurs de la structure à ce niveau.
- Une partie point (`.`) de la requête qui utilise un caractère générique de tableau `[*]` parcourt tous les indices de ce tableau.

- Toutes les collections peuvent être filtrées en spécifiant des filtres entre crochets`[]`. Les collections peuvent être rencontrées des manières suivantes :
  - Les tableaux présents naturellement dans le datum sont des collections. Voici quelques exemple de commandes :

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

    Balises : `[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - Lorsque vous parcourez toutes les valeurs d'une structure telle que `Resources.*`
  - Tout résultat de requête est lui-même une collection à partir de laquelle les valeurs peuvent être filtrées davantage. Consultez l'exemple suivant.

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

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

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

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

Voici un exemple d'extrait CloudFormation de modèle.

```
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
    ...
```

Sur la base de ce modèle, le chemin parcouru est `SampleRole` et la valeur finale sélectionnée est`Type: AWS::IAM::Role`.

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

La valeur résultante de la requête `Resources.*[ Type == 'AWS::IAM::Role' ]` au format YAML est illustrée dans l'exemple suivant.

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

Voici certaines des manières dont vous pouvez utiliser les requêtes :

- Affectez une requête à des variables afin que les résultats de la requête soient accessibles en référençant ces variables.
- Suivez la requête avec un bloc qui teste chacune des valeurs sélectionnées.
- Comparez une requête directement à une clause de base.

Affectation de requêtes à des variables

Guard prend en charge les assignations de variables ponctuelles dans un périmètre donné. Pour plus d'informations sur les variables dans les règles Guard, consultezAffectation et référencement de variables dans les règles Guard.

Vous pouvez attribuer des requêtes à des variables afin de pouvoir écrire des requêtes une seule fois, puis les référencer ailleurs dans vos règles Guard. Consultez les exemples d'attribution de variables suivants qui illustrent les principes de requête décrits plus loin dans cette section.

```
#
```

```
# 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'
    }
]
```

Parcours direct des valeurs d'une variable affectée à une requête

Guard prend en charge l'exécution directe par rapport aux résultats d'une requête. Dans l'exemple suivant, le `when` bloc est testé par rapport à la `AvailabilityZone` propriété `EncryptedVolumeType`, et pour chaque `AWS::EC2::Volume` ressource trouvée dans un CloudFormation modèle.

```
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']
        }
    }
}
```

Comparaisons directes au niveau des clauses

Guard prend également en charge les requêtes dans le cadre de comparaisons directes. Par exemple, consultez ce qui suit.

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

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

Dans l'exemple précédent, les deux clauses (en commençant par le `some` mot clé) exprimées sous la forme illustrée sont considérées comme des clauses indépendantes et sont évaluées séparément.

Formulaire de clause unique et de clause de bloc

Pris ensemble, les deux exemples de clauses présentés dans la section précédente ne sont pas équivalents au bloc suivant.

```
let resources = Resources.*

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

Ce bloc interroge chaque `Tag` valeur de la collection et compare les valeurs de ses propriétés aux valeurs de propriété attendues. La forme combinée des clauses de la section précédente évalue les deux clauses indépendamment. Tenez compte de l'entrée suivante.

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

Les clauses du premier formulaire sont évaluées à`PASS`. Lors de la validation de la première clause sous sa forme initiale, le chemin suivant traverse`Resources`, `PropertiesTags`, et `Key` correspond à la valeur `NotPRODEnd` et ne correspond pas à la valeur `PROD` attendue.

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

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

Il en va de même pour la deuxième clause du premier formulaire. Le chemin traverse`Resources`, `PropertiesTags`, et `Value` correspond à la valeur`AppStart`. Par conséquent, la deuxième clause est indépendante.

Le résultat global est un`PASS`.

Cependant, le formulaire de bloc est évalué comme suit. Pour chaque `Tags` valeur, il compare si `Key` et `Value` si les valeurs correspondent ; `NotAppStart` et `NotPRODEnd` les valeurs ne correspondent pas dans l'exemple suivant.

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

Parce que les évaluations vérifient les deux `Key == /PROD$/Value == /^App/`, la correspondance n'est pas complète. Par conséquent, le résultat est`FAIL`.

> **Note**
>
> Lorsque vous travaillez avec des collections, nous vous recommandons d'utiliser le formulaire de clause de blocage lorsque vous souhaitez comparer plusieurs valeurs pour chaque élément de la collection. Utilisez le formulaire à clause unique lorsque la collection est un ensemble de valeurs scalaires ou lorsque vous souhaitez uniquement comparer un seul attribut.

Résultats de la requête et clauses associées

Toutes les requêtes renvoient une liste de valeurs. Toute partie d'une traversée, telle qu'une clé manquante, des valeurs vides pour un tableau (`Tags: []`) lors de l'accès à tous les index, ou des valeurs manquantes pour une carte lorsque vous rencontrez une carte vide (`Resources: {}`), peut entraîner des erreurs de récupération.

Toutes les erreurs de récupération sont considérées comme des échecs lors de l'évaluation des clauses par rapport à de telles requêtes. La seule exception est lorsque des filtres explicites sont utilisés dans la requête. Lorsque des filtres sont utilisés, les clauses associées sont ignorées.

Les échecs de blocage suivants sont associés à l'exécution de requêtes.

- Si un modèle ne contient pas de ressources, la requête est évaluée à`FAIL`, et les clauses de niveau de bloc associées sont également évaluées à`FAIL`.
- Lorsqu'un modèle contient un bloc de ressources vide comme`{ "Resources": {} }`, la requête est évaluée à`FAIL`, et les clauses de niveau de bloc associées sont également évaluées à`FAIL`.
- Si un modèle contient des ressources mais qu'aucune ne correspond à la requête, la requête renvoie des résultats vides et les clauses de niveau de bloc sont ignorées.

Utilisation de filtres dans les requêtes

Les filtres dans les requêtes sont en fait des clauses Guard utilisées comme critères de sélection. Voici la structure d'une clause.

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

Gardez à l'esprit les points essentiels suivants AWS CloudFormation Guard Règles d'écriture lorsque vous travaillez avec des filtres :

- Combinez des clauses à l'aide de la forme normale conjonctive (CNF).
- Spécifiez chaque clause de conjonction (`and`) sur une nouvelle ligne.
- Spécifiez les disjonctions (`or`) en utilisant le `or` mot clé entre deux clauses.

L'exemple suivant illustre les clauses conjonctives et disjonctives.

```
resourceType == 'AWS::EC2::SecurityGroup'
```

```
InputParameters.TcpBlockedPorts not empty

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

Utilisation de clauses pour les critères de sélection

Vous pouvez appliquer un filtrage à n'importe quelle collection. Le filtrage peut être appliqué directement sur les attributs de l'entrée qui ressemblent déjà à une collection`securityGroups: [....]`. Vous pouvez également appliquer un filtrage à une requête, qui est toujours une collection de valeurs. Vous pouvez utiliser toutes les fonctionnalités des clauses, y compris la forme normale conjonctive, pour le filtrage.

La requête courante suivante est souvent utilisée lors de la sélection de ressources par type à partir d'un CloudFormation modèle.

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

La requête `Resources.*` renvoie toutes les valeurs présentes dans la `Resources` section de l'entrée. Pour l'exemple de modèle saisi dansDéfinition des requêtes, la requête renvoie ce qui suit.

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

Maintenant, appliquez le filtre à cette collection. Le critère à respecter est`Type == AWS::IAM::Role`. Voici le résultat de la requête après l'application du filtre.

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

Ensuite, vérifiez les différentes clauses relatives aux `AWS::IAM::Role` ressources.

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

Voici un exemple de requête de filtrage qui sélectionne toutes les `AWS::IAM::ManagedPolicy` ressources `AWS::IAM::Policy` et toutes les ressources.

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

L'exemple suivant vérifie si ces ressources de politique ont une valeur `PolicyDocument` spécifiée.

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

Définition de besoins de filtrage plus complexes

Prenons l'exemple suivant d'élément de AWS Config configuration pour les informations relatives aux groupes de sécurité d'entrée et de sortie.

```
---
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
```

Notez ce qui suit :

- `ipPermissions`(règles d'entrée) est un ensemble de règles à l'intérieur d'un bloc de configuration.
- Chaque structure de règles contient des attributs tels que `ipv4Ranges` et `ipv6Ranges` pour spécifier une collection de blocs CIDR.

Écrivons une règle qui sélectionne toutes les règles d'entrée qui autorisent les connexions depuis n'importe quelle adresse IP et vérifie que les règles n'autorisent pas l'exposition des ports TCP bloqués.

Commencez par la partie de requête qui couvre IPv4, comme indiqué dans l'exemple suivant.

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

Le `some` mot clé est utile dans ce contexte. Toutes les requêtes renvoient un ensemble de valeurs correspondant à la requête. Par défaut, Guard évalue que toutes les valeurs renvoyées à la suite de la requête sont comparées aux vérifications. Toutefois, il se peut que ce comportement ne soit pas toujours celui dont vous avez besoin pour les vérifications. Tenez compte de la partie suivante de l'entrée provenant de l'élément de configuration.

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

Deux valeurs sont présentes pour`ipv4Ranges`. Toutes les `ipv4Ranges` valeurs ne correspondent pas à une adresse IP désignée par`0.0.0.0/0`. Vous voulez voir si au moins une valeur correspond`0.0.0.0/0`. Vous indiquez à Guard qu'il n'est pas nécessaire que tous les résultats renvoyés par une requête correspondent, mais qu'au moins un résultat doit correspondre. Le `some` mot clé indique à Guard de s'assurer qu'une ou plusieurs valeurs de la requête résultante correspondent à la vérification. Si aucune valeur de résultat de requête ne correspond, Guard génère une erreur.

Ajoutez ensuite IPv6, comme indiqué dans l'exemple suivant.

```
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'
]
```

Enfin, dans l'exemple suivant, confirmez que le protocole ne l'est pas`udp`.

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

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

Voici la règle complète.

```
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
                    >>
                }
            }

        }
    }
}
```

Séparer les collections en fonction de leurs types contenus

Lorsque vous utilisez des modèles de configuration d'infrastructure en tant que code (IaC), vous pouvez rencontrer une collection contenant des références à d'autres entités dans le modèle de configuration. Voici un exemple de CloudFormation modèle qui décrit les tâches Amazon Elastic Container Service (Amazon ECS) avec une référence locale, une référence `TaskRoleArn` à et une référence `TaskArn` de chaîne directe.

```
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'
```

Considérons la requête suivante :

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

Cette requête renvoie une collection de valeurs contenant les trois `AWS::ECS::TaskDefinition` ressources présentées dans l'exemple de modèle. Séparez `ecs_tasks` les références `TaskRoleArn` locales des autres, comme indiqué dans l'exemple suivant.

```
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
    >>
}
```

Affectation et référencement de variables dans les règles Guard

Vous pouvez attribuer des variables dans vos fichiers de AWS CloudFormation Guard règles pour stocker les informations auxquelles vous souhaitez faire référence dans vos règles Guard. Guard prend en charge l'attribution de variables en un seul coup. Les variables sont évaluées paresseusement, ce qui signifie que Guard n'évalue les variables que lorsque les règles sont exécutées.

Rubriques

Affectation de variables

Utilisez le `let` mot clé pour initialiser et attribuer une variable. Il est recommandé d'utiliser Snake Case pour les noms de variables. Les variables peuvent stocker des littéraux statiques ou des propriétés dynamiques résultant de requêtes. Dans l'exemple suivant, la variable `ecs_task_definition_task_role_arn` stocke la valeur de chaîne statique`arn:aws:iam:123456789012:role/my-role-name`.

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

Dans l'exemple suivant, la variable `ecs_tasks` stocke les résultats d'une requête qui recherche toutes les `AWS::ECS::TaskDefinition` ressources d'un CloudFormation modèle. Vous pouvez faire référence `ecs_tasks` aux informations d'accès relatives à ces ressources lorsque vous rédigez des règles.

```
let ecs_tasks = Resources.*[
```

```
    Type == 'AWS::ECS::TaskDefinition'
]
```

Référencement de variables

Utilisez le % préfixe pour référencer une variable.

Sur la base de l'exemple de `ecs_task_definition_task_role_arn` variable dansAffectation de variables, vous pouvez faire référence `ecs_task_definition_task_role_arn` dans la `query|value literal` section d'une clause de règle de garde. L'utilisation de cette référence garantit que la valeur spécifiée pour la `TaskDefinitionArn` propriété de toute `AWS::ECS::TaskDefinition` ressource dans un CloudFormation modèle est la valeur de chaîne statique`earn:aws:iam:123456789012:role/my-role-name`.

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

Sur la base de la `ecs_tasks` variable exemple inAffectation de variables, vous pouvez faire référence `ecs_tasks` à une requête (par exemple, %ECS_Tasks.properties). Guard évalue d'abord la variable, `ecs_tasks` puis utilise les valeurs renvoyées pour parcourir la hiérarchie. Si la variable `ecs_tasks` prend des valeurs autres que des chaînes, Guard génère une erreur.

> **Note**
>
> À l'heure actuelle, Guard ne prend pas en charge le référencement de variables dans les messages d'erreur personnalisés.

Champ d'application variable

La portée fait référence à la visibilité des variables définies dans un fichier de règles. Un nom de variable ne peut être utilisé qu'une seule fois dans une portée. Il existe trois niveaux où une variable peut être déclarée, ou trois portées de variables possibles :

- Niveau fichier — Généralement déclarées en haut du fichier de règles, vous pouvez utiliser des variables de niveau fichier dans toutes les règles du fichier de règles. Ils sont visibles dans l'ensemble du fichier.

Dans l'exemple de fichier de règles suivant, les variables `ecs_task_definition_task_role_arn` et B `ecs_task_definition_execution_role_arn` sont initialisées au niveau du fichier.

```
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
}
```

- Niveau règle — Déclarées dans une règle, les variables au niveau de la règle ne sont visibles que pour cette règle spécifique. Toute référence en dehors de la règle entraîne une erreur.

  Dans l'exemple de fichier de règles suivant, les variables `ecs_task_definition_task_role_arn` et B `ecs_task_definition_execution_role_arn` sont initialisées au niveau des règles. Ils ne `ecs_task_definition_task_role_arn` peuvent être référencés que dans la règle `check_ecs_task_definition_task_role_arn` nommée. Vous ne pouvez référencer la `ecs_task_definition_execution_role_arn` variable que dans la règle `check_ecs_task_definition_execution_role_arn` nommée.

```
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
}
```

- Niveau bloc — Déclarées dans un bloc, tel qu'une when clause, les variables au niveau du bloc ne sont visibles que pour ce bloc spécifique. Toute référence en dehors du bloc entraîne une erreur.

  Dans l'exemple de fichier de règles suivant, les variables `ecs_task_definition_task_role_arn` et B `ecs_task_definition_execution_role_arn` sont initialisées au niveau du bloc au sein du `AWS::ECS::TaskDefinition` bloc de type. Vous ne pouvez référencer les `ecs_task_definition_execution_role_arn` variables `ecs_task_definition_task_role_arn` et dans les blocs `AWS::ECS::TaskDefinition` de type que pour leurs règles respectives.

```
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
    }
}
```

Exemples de variables dans les fichiers de règles Guard

Les sections suivantes fournissent des exemples d'attribution statique et dynamique de variables.

Affectation statique

Voici un exemple de CloudFormation modèle.

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

Sur la base de ce modèle, vous pouvez écrire une règle appelée `check_ecs_task_definition_task_role_arn` qui garantit que la `TaskRoleArn` propriété de toutes les ressources du `AWS::ECS::TaskDefinition` modèle est`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
}
```

Dans le cadre de la règle, vous pouvez initialiser une variable appelée `ecs_task_definition_task_role_arn` et lui attribuer la valeur `'arn:aws:iam::123456789012:role/my-role-name'` de chaîne statique. La clause de règle vérifie si la valeur spécifiée pour la `TaskRoleArn` propriété de la `EcsTask` ressource est `arn:aws:iam::123456789012:role/my-role-name` en faisant référence à la `ecs_task_definition_task_role_arn` variable dans la `query|value literal` section.

Affectation dynamique

Voici un exemple de CloudFormation modèle.

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

Sur la base de ce modèle, vous pouvez initialiser une variable appelée `ecs_tasks` dans le cadre du fichier et lui attribuer la requête`Resources.*[ Type == 'AWS::ECS::TaskDefinition'`.

Guard interroge toutes les ressources du modèle de saisie et y stocke les informations les concernant`ecs_tasks`. Vous pouvez également écrire une règle appelée `check_ecs_task_definition_task_role_arn` qui garantit que la `TaskRoleArn` propriété de toutes les ressources du `AWS::ECS::TaskDefinition` modèle est `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 clause de règle vérifie si la valeur spécifiée pour la `TaskRoleArn` propriété de la `EcsTask` ressource est `arn:aws:iam::123456789012:role/my-role-name` en faisant référence à la `ecs_task_definition_task_role_arn` variable dans la `query` section.

Application de la configuration des CloudFormation modèles

Passons en revue un exemple plus complexe de cas d'utilisation en production. Dans cet exemple, nous écrivons des règles Guard pour garantir des contrôles plus stricts sur la façon dont les tâches Amazon ECS sont définies.

Voici un exemple de CloudFormation modèle.

```
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'
```

Sur la base de ce modèle, nous écrivons les règles suivantes pour garantir le respect de ces exigences :

- Chaque `AWS::ECS::TaskDefinition` ressource du modèle est associée à la fois à un rôle de tâche et à un rôle d'exécution.
- Les rôles de tâches et les rôles d'exécution sont des rôles Gestion des identités et des accès AWS (IAM).
- Les rôles sont définis dans le modèle.
- La `PermissionsBoundary` propriété est spécifiée pour chaque rôle.

```
# 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
    }
}
```

Composer des blocs de règles nommées dans AWS CloudFormation Guard

Lorsque vous écrivez des blocs de règles nommées à l'aide de AWS CloudFormation Guard, vous pouvez utiliser les deux styles de composition suivants :

- Dépendance conditionnelle
- Dépendance corrélationnelle

L'utilisation de l'un ou l'autre de ces styles de composition des dépendances contribue à promouvoir la réutilisabilité et à réduire la verbosité et la répétition dans les blocs de règles nommés.

Rubriques

- Conditions préalables
- Composition des dépendances conditionnelles
- Composition des dépendances corrélationnelles

Conditions préalables

Pour en savoir plus sur les blocs de règles nommés, consultez Writing rules.

Composition des dépendances conditionnelles

Dans ce style de composition, l'évaluation d'un `when` bloc ou d'un bloc de règles nommées dépend conditionnellement du résultat de l'évaluation d'un ou de plusieurs autres blocs ou clauses de règles nommées. L'exemple de fichier de règles de garde suivant contient des blocs de règles nommées qui illustrent les dépendances conditionnelles.

```
# 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
    ...
}
```

Dans l'exemple de fichier de règles précédent, `Example-1` les résultats possibles sont les suivants :

- Si `rule_name_A` la valeur est égale à`PASS`, les règles Guard encapsulées par `rule_name_B` sont évaluées.
- Si `rule_name_A` la valeur est égale à`FAIL`, les règles Guard encapsulées par ne `rule_name_B` sont pas évaluées. `rule_name_B`évalue à. `SKIP`
- Si `rule_name_A` la valeur est égale à`SKIP`, les règles Guard encapsulées par ne `rule_name_B` sont pas évaluées. `rule_name_B`évalue à. `SKIP`

> **Note**
>
> Ce cas se produit s'il dépend `rule_name_A` conditionnellement d'une règle qui évalue jusqu'à `FAIL` et aboutit à une `rule_name_A` évaluation vers. `SKIP`

Voici un exemple d'élément de configuration d'une base de données de gestion de configuration (CMDB) provenant d'un AWS Config élément contenant des informations sur les groupes de sécurité d'entrée et de sortie. Cet exemple illustre la composition des dépendances conditionnelles.

```
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
                }
            }
        }
    }
}
```

Dans l'exemple précédent, `check_parameter_validity` dépend conditionnellement de `check_resource_type_and_parameter` et `check_ip_procotol_and_port_range_validity` dépend conditionnellement de. `check_parameter_validity` Voici un élément de configuration de la base de données de gestion de la configuration (CMDB) conforme aux règles précédentes.

```
---
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
```

Composition des dépendances corrélationnelles

Dans ce style de composition, l'évaluation d'un `when` bloc ou d'un bloc à règles nommées dépend corrélationnellement du résultat de l'évaluation d'une ou de plusieurs autres règles Guard. La dépendance corrélationnelle peut être obtenue comme suit.

```
# 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
    ...
}
```

Pour vous aider à comprendre la composition des dépendances corrélationnelles, consultez l'exemple suivant de fichier de règles 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'
}
```

Dans le fichier de règles précédent, `ensure_elbs_are_internal_and_secure` possède une dépendance corrélationnelle sur. `ensure_all_elbs_are_secure` Voici un exemple de CloudFormation modèle conforme aux règles précédentes.

```
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...'
```

Rédaction de clauses pour effectuer des évaluations contextuelles

AWS CloudFormation Guard les clauses sont évaluées par rapport à des données hiérarchiques. Le moteur d'évaluation Guard résout les requêtes relatives aux données entrantes en suivant les données hiérarchiques telles que spécifiées, à l'aide d'une simple notation en pointillés. Plusieurs clauses sont souvent nécessaires pour effectuer une évaluation par rapport à une carte de données

ou à une collection. Guard fournit une syntaxe pratique pour écrire de telles clauses. Le moteur est conscient du contexte et utilise les données correspondantes associées pour les évaluations.

Voici un exemple de configuration de Kubernetes Pod avec des conteneurs, à laquelle vous pouvez appliquer des évaluations contextuelles.

```
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
```

Vous pouvez créer des clauses Guard pour évaluer ces données. Lors de l'évaluation d'un fichier de règles, le contexte est l'intégralité du document d'entrée. Vous trouverez ci-dessous des exemples de clauses qui valident l'application des limites pour les conteneurs spécifiés dans 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
            >>
        }
    }
}
```

Compréhension **`context`** lors des évaluations

Au niveau du bloc de règles, le contexte entrant est le document complet. Les évaluations de la `when` condition sont effectuées par rapport à ce contexte racine entrant dans lequel se trouvent les `kind` attributs `apiVersion` et. Dans l'exemple précédent, ces conditions sont évaluées à`true`.

Maintenant, parcourez la hiérarchie `spec.containers[*]` comme indiqué dans l'exemple précédent. Pour chaque traversée de la hiérarchie, la valeur de contexte change en conséquence. Une fois la traversée du `spec` bloc terminée, le contexte change, comme indiqué dans l'exemple suivant.

```
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
```

Après avoir parcouru l'`containers`attribut, le contexte est illustré dans l'exemple suivant.

```
- 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
```

Comprendre les boucles

Vous pouvez utiliser l'expression `[*]` pour définir une boucle pour toutes les valeurs contenues dans le tableau de l'`containers`attribut. Le bloc est évalué pour chaque élément qu'il contient`containers`. Dans l'exemple d'extrait de règle précédent, les clauses contenues dans le

bloc définissent les contrôles à valider par rapport à une définition de conteneur. Le bloc de clauses qu'il contient est évalué deux fois, une fois pour chaque définition de conteneur.

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

Pour chaque itération, la valeur de contexte est la valeur correspondant à l'index correspondant.

> **Note**
>
> Le seul format d'accès à l'index pris en charge est `[<integer>]` ou`[*]`. Actuellement, Guard ne prend pas en charge les plages de ce type`[2..4]`.

Arrays (tableaux)

Souvent, dans les endroits où un tableau est accepté, les valeurs uniques sont également acceptées. Par exemple, s'il n'y a qu'un seul conteneur, le tableau peut être supprimé et l'entrée suivante est acceptée.

```
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 attribut peut accepter un tableau, assurez-vous que votre règle utilise la forme matricielle. Dans l'exemple précédent, vous utilisez `containers[*]` et non`containers`. Guard évalue correctement lorsqu'il parcourt les données lorsqu'elles ne rencontrent que le formulaire à valeur unique.

> **Note**
>
> Utilisez toujours la forme de tableau lorsque vous exprimez l'accès à une clause de règle lorsqu'un attribut accepte un tableau. Guard évalue correctement même dans le cas où une seule valeur est utilisée.

En utilisant le formulaire **`spec.containers[*]`** au lieu de **`spec.containers`**

Les requêtes Guard renvoient un ensemble de valeurs résolues. Lorsque vous utilisez le formulaire`spec.containers`, les valeurs résolues pour la requête contiennent le tableau référencé par`containers`, et non les éléments qu'il contient. Lorsque vous utilisez le formulaire`spec.containers[*]`, vous faites référence à chaque élément individuel qu'il contient. N'oubliez pas d'utiliser le `[*]` formulaire chaque fois que vous avez l'intention d'évaluer chaque élément contenu dans le tableau.

Utilisation **`this`** pour référencer la valeur de contexte actuelle

Lorsque vous créez une règle de garde, vous pouvez référencer la valeur de contexte en utilisant`this`. Souvent, elle `this` est implicite car elle est liée à la valeur du contexte. Par exemple, `this.apiVersion`this.`kind`, et `this.spec` sont liés à la racine ou au document. En revanche, `this.resources` est lié à chaque valeur pour`containers`, telle que `/spec/containers/0/` et`/spec/containers/1`. De même, `this.cpu` et `this.memory` cartographiez les limites, en particulier `/spec/containers/0/resources/limits` et`/spec/containers/1/resources/limits`.

Dans l'exemple suivant, la règle précédente pour la configuration de Kubernetes Pod est réécrite pour être utilisée explicitement. `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
            >>
        }
    }
}
```

Vous n'avez pas besoin d'utiliser `this` explicitement. Cependant, la `this` référence peut être utile lorsque vous travaillez avec des scalaires, comme le montre l'exemple suivant.

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

Dans l'exemple précédent, `this` est utilisé pour faire référence à chaque numéro de port.

Erreurs potentielles liées à l'utilisation de l'implicite **`this`**

Lors de la création de règles et de clauses, des erreurs fréquentes se produisent lors du référencement d'éléments à partir de la valeur de `this` contexte implicite. Par exemple, considérez la donnée d'entrée suivante à évaluer (elle doit être acceptée).

```
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
```

Lorsqu'elle est testée par rapport au modèle précédent, la règle suivante génère une erreur car elle suppose à tort qu'elle tire parti de l'implicite`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
                >>
            }
        }
    }
}
```

Pour suivre cet exemple, enregistrez le fichier de règles précédent avec le nom `any_ip_ingress_check.guard` et les données avec le nom du fichier`ip_ingress.yaml`. Exécutez ensuite la `validate` commande suivante avec ces fichiers.

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

Dans le résultat suivant, le moteur indique que sa tentative de récupération d'une propriété `InputParameters.TcpBlockedPorts[*]` sur la valeur `/configuration/ipPermissions/0` a `/configuration/ipPermissions/1` échoué.

```
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[*]
```

Pour mieux comprendre ce résultat, réécrivez la règle en utilisant des références `this` explicites.

```
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`fait référence à chaque valeur contenue dans la variable`any_ip_permissions`. La requête affectée à la variable sélectionne `configuration.ipPermissions` les valeurs correspondantes. L'erreur indique une tentative de récupération `InputParamaters` dans ce contexte, mais elle `InputParameters` s'est produite dans le contexte racine.

Le bloc interne fait également référence à des variables hors de portée, comme indiqué dans l'exemple suivant.

```
{
    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`fait référence à chaque valeur de port dans`[21, 22, 110]`, mais il fait également référence à `fromPort` et`toPort`. Ils appartiennent tous deux à la portée du bloc extérieur.

Résoudre les erreurs à l'aide de l'utilisation implicite de **`this`**

Utilisez des variables pour attribuer et référencer des valeurs de manière explicite. Tout d'abord, cela `InputParameter.TcpBlockedPorts` fait partie du contexte d'entrée (racine). `InputParameter.TcpBlockedPorts`Sortez du bloc interne et attribuez-le explicitement, comme indiqué dans l'exemple suivant.

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

Ensuite, faites référence à cette variable de manière explicite.

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

Procédez de même pour les `this` références internes à l'intérieur`%ports`.

Cependant, toutes les erreurs ne sont pas encore corrigées car la boucle à l'intérieur contient `ports` toujours une référence incorrecte. L'exemple suivant montre la suppression de la référence incorrecte.

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

Ensuite, exécutez à nouveau la `validate` commande. Cette fois, ça passe.

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

Le résultat de la `validate` commande est le suivant.

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

Pour tester cette approche en cas d'échec, l'exemple suivant utilise un changement de charge utile.

```
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 se situe dans la plage comprise entre 89 et 109 pour lesquelles n'importe quelle IPv6 adresse est autorisée. Voici le résultat de la `validate` commande après l'avoir exécutée à nouveau.

```
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 Règles de test

Vous pouvez utiliser le cadre de test unitaire AWS CloudFormation Guard intégré pour vérifier que vos règles Guard fonctionnent comme prévu. Cette section explique comment écrire un fichier de test unitaire et comment l'utiliser pour tester votre fichier de règles à l'aide de la `test` commande.

Votre fichier de test unitaire doit avoir l'une des extensions suivantes : `.json`.`JSON`,`.jsn`,`.yaml`,`.YAML`, ou `.yml`.

Rubriques

Conditions préalables

Rédigez des règles de garde pour évaluer vos données d'entrée. Pour de plus amples informations, veuillez consulter Règles de Writing Guard.

Vue d'ensemble des fichiers de tests unitaires Guard

Les fichiers de test unitaire Guard sont des fichiers au format JSON ou YAML qui contiennent plusieurs entrées et les résultats attendus pour les règles écrites dans un fichier de règles Guard. Il peut y avoir plusieurs échantillons pour évaluer les différentes attentes. Nous vous recommandons de commencer par tester les entrées vides, puis d'ajouter progressivement des informations pour évaluer les différentes règles et clauses.

Nous vous recommandons également de nommer les fichiers de tests unitaires en utilisant le suffixe `_test.json` ou`_tests.yaml`. Par exemple, si vous avez un fichier de règles nommé`my_rules.guard`, nommez votre fichier de tests unitaires`my_rules_tests.yaml`.

Syntaxe

Ce qui suit montre la syntaxe d'un fichier de test unitaire au format YAML.

```
---
- name: <TEST NAME>
```

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

Propriétés

Voici les propriétés d'un fichier de test Guard.

`input`

Des données pour tester vos règles. Nous recommandons que votre premier test utilise une entrée vide, comme illustré dans l'exemple suivant.

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

Pour les tests suivants, ajoutez les données d'entrée à tester.

Obligatoire : oui

`expectations`

Le résultat attendu lorsque des règles spécifiques sont évaluées par rapport à vos données d'entrée. Spécifiez une ou plusieurs règles que vous souhaitez tester en plus du résultat attendu pour chaque règle. Le résultat attendu doit être l'un des suivants :

- `PASS`— Lorsqu'elles sont exécutées par rapport à vos données d'entrée, les règles sont évaluées à`true`.
- `FAIL`— Lorsqu'elles sont exécutées par rapport à vos données d'entrée, les règles sont évaluées à`false`.
- `SKIP`— Lorsqu'elle est exécutée sur vos données d'entrée, la règle n'est pas déclenchée.

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

Obligatoire : oui

Procédure pas à pas de rédaction d'un fichier de test unitaire des règles Guard

Voici un fichier de règles nomméapi_gateway_private.guard. Le but de cette règle est de vérifier si tous les types de ressources Amazon API Gateway définis dans un CloudFormation modèle sont déployés pour un accès privé uniquement. Il vérifie également si au moins une déclaration de politique autorise l'accès depuis un cloud privé virtuel (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 Gestion des
 identités et des accès AWS (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
            }
        }
    }
}
```

Cette procédure pas à pas teste l'intention de la première règle : toutes les `AWS::ApiGateway::RestApi` ressources déployées doivent être privées.

1. Créez un fichier de test unitaire appelé `api_gateway_private_tests.yaml` qui contient le test initial suivant. Lors du test initial, ajoutez une entrée vide et attendez-vous à ce que la règle `check_rest_api_is_private` soit ignorée car il n'y a aucune `AWS::ApiGateway::RestApi` ressource en entrée.

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

2. Exécutez le premier test dans votre terminal à l'aide de la `test` commande. Pour le `--rules-file` paramètre, spécifiez votre fichier de règles. Pour le `--test-data` paramètre, spécifiez votre fichier de test unitaire.

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

   Le résultat du premier test est`PASS`.

   ```
   Test Case #1
   Name: "MyTest1"
     PASS Rules:
   ```

```
    check_rest_api_is_private: Expected = SKIP, Evaluated = SKIP
```

3. Ajoutez un autre test à votre fichier de tests unitaires. Maintenant, étendez les tests pour inclure les ressources vides. Le `api_gateway_private_tests.yaml` fichier mis à jour est le suivant.

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

4. Exécutez `test` avec le fichier de test unitaire mis à jour.

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

Le résultat du deuxième test est`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. Ajoutez deux autres tests à votre fichier de tests unitaires. Étendez les tests pour inclure les éléments suivants :

   - `AWS::ApiGateway::RestApi`Ressource dont aucune propriété n'est spécifiée.

> **Note**
>
> Ce CloudFormation modèle n'est pas valide, mais il est utile pour vérifier si la règle fonctionne correctement, même pour des entrées mal formées.

Attendez-vous à ce que ce test échoue car la `EndpointConfiguration` propriété n'est pas spécifiée et n'est donc pas définie sur`PRIVATE`.

- `AWS::ApiGateway::RestApi`Ressource qui répond à la première intention avec la `EndpointConfiguration` propriété définie sur, `PRIVATE` mais ne satisfait pas à la seconde, car aucune déclaration de politique n'est définie pour elle. Attendez-vous à ce que ce test soit réussi.

Le fichier de test unitaire mis à jour est le suivant.

```
---
- 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. Exécutez `test` avec le fichier de test unitaire mis à jour.

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

Le troisième résultat est`FAIL`, et le quatrième résultat l'est`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. Commentez les tests 1 à 3 dans votre fichier de tests unitaires. Accédez au contexte détaillé pour le quatrième test uniquement. Le fichier de test unitaire mis à jour est le suivant.

```
---
#- 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. Vérifiez les résultats de l'évaluation en exécutant la `test` commande dans votre terminal à l'aide de l'`--verbose`indicateur. Le contexte verbeux est utile pour comprendre les évaluations. Dans ce cas, il fournit des informations détaillées sur les raisons pour lesquelles le quatrième test a réussi et a donné un PASS résultat.

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

Voici le résultat de cette exécution.

```
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 principale observation de la sortie est la ligne`Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`, qui indique que le contrôle a été réussi. L'exemple a également montré le cas où l'on s'`Type`sattendait à ce qu'il s'agisse d'un tableau, mais où une seule valeur a été donnée. Dans ce cas, Guard a poursuivi son évaluation et a fourni un résultat correct.

9. Ajoutez un scénario de test tel que le quatrième scénario de test à votre fichier de test unitaire pour une `AWS::ApiGateway::RestApi` ressource dont la `EndpointConfiguration` propriété est spécifiée. Le scénario de test échouera au lieu de réussir. Le fichier de test unitaire mis à jour est le suivant.

```
---
#- 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. Exécutez la `test` commande avec le fichier de test unitaire mis à jour à l'aide de l'`--verbose`indicateur.

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

Le résultat est `FAIL` conforme aux attentes car `REGIONAL` il est spécifié `EndpointConfiguration` mais n'est pas attendu.

```
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)
```

La sortie détaillée de la `test` commande suit la structure du fichier de règles. Chaque bloc du fichier de règles est un bloc de la sortie détaillée. Le bloc le plus élevé correspond à chaque règle. S'il existe `when` des conditions contraires à la règle, elles apparaissent dans un bloc de conditions frère. Dans l'exemple suivant, la condition `%api_gws !empty` est testée et elle passe.

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

Une fois la condition remplie, nous testons les clauses des règles.

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

`%api_gws`est une règle de blocage qui correspond au `BlockClause` niveau de la sortie (ligne : 21). La clause de règle est un ensemble de clauses de conjonction (AND), chaque clause de conjonction étant un ensemble de disjonctions. 0R La conjonction comporte une seule clause,`Properties.EndpointConfiguration.Types[*] == "PRIVATE"`. Par conséquent, la sortie détaillée affiche une seule clause. Le chemin `/Resources/apiGw/Properties/EndpointConfiguration/Types/1` indique quelles valeurs de l'entrée sont comparées. Dans ce cas, il s'agit de l'élément à `Types` indexer à 1.

DansValidation des données d'entrée par rapport aux règles Guard, vous pouvez utiliser les exemples de cette section pour utiliser la `validate` commande afin d'évaluer les données d'entrée par rapport aux règles.

Utilisation de paramètres d'entrée avec des AWS CloudFormation Guard règles

AWS CloudFormation Guard vous permet d'utiliser des paramètres d'entrée pour des recherches de données dynamiques lors de la validation. Cette fonctionnalité est particulièrement utile lorsque vous

devez référencer des données externes dans vos règles. Toutefois, lors de la spécification des clés de paramètres d'entrée, Guard exige qu'il n'y ait aucun chemin conflictuel.

Comment utiliser

1. Utilisez l'`-i`indicateur `--input-parameters` ou pour spécifier les fichiers contenant les paramètres d'entrée. Plusieurs fichiers de paramètres d'entrée peuvent être spécifiés et seront combinés pour former un contexte commun. Les clés de paramètres d'entrée ne peuvent pas avoir de chemins conflictuels.
2. Utilisez l'`-d`indicateur `--data` ou pour spécifier le fichier modèle à valider.

Exemple d'utilisation

1. Créez un fichier de paramètres d'entrée (par exemple,`network.yaml`) :

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

2. Référencez ces paramètres dans votre fichier de règles de garde (par exemple,`security_groups.guard`) :

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

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

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

3. Créez un modèle de données défaillant (par exemple,`security_groups_fail.yaml`) :

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

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

4. Exécutez la commande de validation :

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

Dans cette commande :

- `-r`spécifie le fichier de règles.
- `-i`spécifie le fichier de paramètres d'entrée.
- `-d`indique le fichier de données (modèle) à valider.

Paramètres d'entrée multiples

Vous pouvez spécifier plusieurs fichiers de paramètres d'entrée :

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

Tous les fichiers spécifiés avec `-i` seront combinés pour former un contexte unique pour la recherche de paramètres.

Validation des données d'entrée par rapport aux règles AWS CloudFormation Guard

Vous pouvez utiliser la AWS CloudFormation Guard `validate` commande pour valider les données par rapport aux règles Guard. Pour plus d'informations sur la `validate` commande, notamment ses paramètres et options, consultez la section valider.

Conditions préalables

- Règles Write Guard pour valider vos données d'entrée. Pour de plus amples informations, veuillez consulter Règles de Writing Guard.
- Testez vos règles pour vous assurer qu'elles fonctionnent comme prévu. Pour de plus amples informations, veuillez consulter Règles de Testing Guard.

Utilisation de la commande **`validate`**

Pour valider vos données d'entrée par rapport à vos règles Guard, telles qu'un AWS CloudFormation modèle, exécutez la `validate` commande Guard. Pour le `--rules` paramètre, spécifiez le nom d'un fichier de règles. Pour le `--data` paramètre, spécifiez le nom du fichier de données d'entrée.

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

Si Guard valide les modèles avec succès, la `validate` commande renvoie un statut de sortie de `0` (`$?`en bash). Si Guard identifie une violation des règles, la `validate` commande renvoie un rapport d'état des règles qui ont échoué. Utilisez l'indicateur récapitulatif (`-s all`) pour voir l'arbre d'évaluation détaillé qui montre comment Guard a évalué chaque règle.

```
template.json Status = FAIL
SKIP rules
rules.guard/aws_apigateway_deployment_checks    SKIP
rules.guard/aws_apigateway_stage_checks         SKIP
rules.guard/aws_dynamodb_table_checks           SKIP
PASS rules
rules.guard/aws_events_rule_checks              PASS
rules.guard/aws_iam_role_checks                 PASS
FAILED rules
rules.guard/aws_ec2_volume_checks               FAIL
rules.guard/mixed_types_checks                  FAIL
---
Evaluation of rules rules.guard against data template.json
--
Property [/Resources/vol2/Properties/Encrypted] in data [template.json] is not
 compliant with [rules.guard/aws_ec2_volume_checks] because provided value [false] did
 not match expected value [true]. Error Message []
Property traversed until [/Resources/vol2/Properties] in data [template.json] is not
 compliant with [rules.guard/aws_ec2_volume_checks] due to retrieval error. Error
```

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

Validation de plusieurs règles par rapport à plusieurs fichiers de données

Pour faciliter le maintien des règles, vous pouvez les écrire dans plusieurs fichiers et les organiser comme vous le souhaitez. Vous pouvez ensuite valider plusieurs fichiers de règles par rapport à un ou plusieurs fichiers de données. La `validate` commande peut utiliser un répertoire de fichiers pour les `--rules` options `--data` et. Par exemple, vous pouvez exécuter la commande suivante `/path/to/dataDirectory` contenant un ou plusieurs fichiers de données et `/path/to/ruleDirectory` un ou plusieurs fichiers de règles.

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

Vous pouvez écrire des règles pour vérifier si les différentes ressources définies dans plusieurs CloudFormation modèles possèdent les propriétés appropriées pour garantir le chiffrement au repos. Pour faciliter la recherche et la maintenance, vous pouvez définir des règles pour vérifier le chiffrement au repos dans chaque ressource dans des fichiers distincts`s3_bucket_encryption.guard`, appelé`ec2_volume_encryption.guard`, et `rds_dbinstance_encrytion.guard` dans un répertoire avec le chemin`~/GuardRules/encryption_at_rest`. Les CloudFormation modèles que vous devez valider se trouvent dans un répertoire avec le chemin`~/CloudFormation/templates`. Dans ce cas, exécutez la `validate` commande comme suit.

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

Résolution des problèmes AWS CloudFormation Guard

Si vous rencontrez des problèmes lors de l' AWS CloudFormation Guard utilisation, consultez les rubriques de cette section.

Rubriques

La clause échoue lorsqu'aucune ressource du type sélectionné n'est présente

Lorsqu'une requête utilise un filtre tel que`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`, s'il n'y a aucune `AWS::ApiGateway::RestApi` ressource dans l'entrée, la clause est évaluée à`FAIL`.

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

Pour éviter ce résultat, attribuez des filtres aux variables et utilisez le contrôle des `when` conditions.

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

Guard n'évalue pas le CloudFormation modèle contenant des références abrégées Fn::GetAtt

Guard ne prend pas en charge les formes abrégées des fonctions intrinsèques. Par exemple, l'utilisation de`!Join`, `!Sub` dans un CloudFormation modèle au format YAML n'est pas prise en charge. Utilisez plutôt les formes étendues des fonctions CloudFormation intrinsèques. Par exemple`Fn::Join`, utilisez-le `Fn::Sub` dans les CloudFormation modèles au format YAML lorsque vous les évaluez par rapport aux règles Guard.

Pour plus d'informations sur les fonctions intrinsèques, consultez la référence aux fonctions intrinsèques dans le guide de AWS CloudFormation l'utilisateur.

Rubriques générales de résolution des problèmes

- Vérifiez que `string` les littéraux ne contiennent pas de chaînes échappées intégrées. Guard ne prend pas en charge les chaînes d'échappement intégrées dans `string` les littéraux. Si votre intention est d'analyser des chaînes JSON intégrées, utilisez la `json_parse()` fonction disponible dans Guard 3.0.0 et versions ultérieures. Pour de plus amples informations, veuillez consulter Utilisation des fonctions intégrées.
- Vérifiez que vos `!=` comparaisons comparent les types de données compatibles. Par exemple, a `string` et an ne `int` sont pas des types de données compatibles à des fins de comparaison. Lorsque vous effectuez une `!=` comparaison, si les valeurs sont incompatibles, une erreur se produit en interne. Actuellement, l'erreur est supprimée et convertie `false` pour satisfaire le PartialEqtrait dans Rust.

AWS CloudFormation Guard Paramètres de la CLI et référence de commande

Les paramètres et commandes globaux suivants sont disponibles via l'interface de ligne de AWS CloudFormation Guard commande (CLI).

Rubriques

Paramètres globaux de la CLI Guard

Vous pouvez utiliser les paramètres suivants avec n'importe quelle commande AWS CloudFormation Guard CLI.

`-h`, `--help`

Imprime les informations d'aide.

`-V`, `--version`

Imprime les informations de version.

arbre d'analyse

Génère un arbre d'analyse pour les AWS CloudFormation Guard règles définies dans un fichier de règles.

Syntaxe

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

```
--rules <value>
```

Parameters

`-h, --help`

Imprime les informations d'aide.

`-p, --print-json`

Imprime la sortie au format JSON.

`-y, --print-yaml`

Imprime la sortie au format YAML.

`-V, --version`

Imprime les informations de version.

Options

`-o, --output`

Ecrit l'arbre généré dans un fichier de sortie.

`-r, --rules`

Fournit un fichier de règles.

Exemples

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

rulegen

Prend un fichier AWS CloudFormation modèle au format JSON ou YAML et génère automatiquement un ensemble de AWS CloudFormation Guard règles correspondant aux propriétés des ressources du modèle. Cette commande est utile pour commencer à écrire des règles ou pour créer des ready-to-use règles à partir de modèles éprouvés.

Syntaxe

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

Parameters

`-h`, `--help`

Imprime les informations d'aide.

`-V`, `--version`

Imprime les informations de version.

Options

`-o`, `--output`

Ecrit les règles générées dans un fichier de sortie. Étant donné le risque que des centaines, voire des milliers de règles apparaissent, nous vous recommandons d'utiliser cette option.

`-t`, `--template`

Fournit le chemin d'accès à un fichier CloudFormation modèle au format JSON ou YAML.

Exemples

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

test

Valide un fichier de AWS CloudFormation Guard règles par rapport à un fichier de test unitaire Guard au format JSON ou YAML afin de déterminer le succès de chaque règle.

Syntaxe

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

```
--test-data <value>
```

Parameters

`-a`, `--alphabetical`

Triez par ordre alphabétique dans un répertoire.

`-h`, `--help`

Imprime les informations d'aide.

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

Trie par date de dernière modification dans un répertoire

`-V`, `--version`

Imprime les informations de version.

`-v`, `--verbose`

Augmente la verbosité de sortie. Peut être spécifié plusieurs fois.

La sortie détaillée suit la structure du fichier de règles Guard. Chaque bloc du fichier de règles est un bloc de la sortie détaillée. Le bloc le plus élevé correspond à chaque règle. S'il existe des `when` conditions contraires à la règle, elles apparaissent sous la forme d'un bloc de conditions frère.

Options

`-d`, `--dir`

Indiquez le répertoire racine pour les règles.

`-o`, `--output-format`

Spécifiez le format dans lequel la sortie doit être affichée.

Par défaut : `single-line-summary`

Valeurs autorisées : `json` | `yaml` | `single-line-summary` | `junit`

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

Fournit le nom d'un fichier de règles.

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

Fournit le nom d'un fichier ou d'un répertoire pour les fichiers de données au format JSON ou YAML.

Exemples

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

Output

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

Consultez aussi

Règles de Testing Guard

valider

Valide les données par rapport aux AWS CloudFormation Guard règles afin de déterminer le succès ou l'échec.

Syntaxe

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

Parameters

`-a`, `--alphabetical`

Valide les fichiers d'un répertoire classé par ordre alphabétique.

`-h, --help`

Imprime les informations d'aide.

`-m, --last-modified`

Valide les fichiers d'un répertoire trié par date de dernière modification.

`-P, --payload`

Fournissez des règles et des données au format JSON suivant via `stdin` :

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

Par exemple :

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

Pour « règles », spécifiez une liste des versions sous forme de chaîne des fichiers de règles. Pour « données », spécifiez une liste des versions sous forme de chaîne des fichiers de données.

Quand `--payload` est spécifié `--rules` et `--data` ne peut pas être précisé.

`-p, --print-json`

Imprime la sortie au format JSON.

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

Affiche l'échec de la clause, y compris un résumé.

`-V, --version`

Imprime les informations de version.

`-v`, `--verbose`

Augmente la verbosité de sortie. Peut être spécifié plusieurs fois.

`-z`, `--structured`

Imprime une liste de formats de sortie structurés et valides JSON/YAML. This argument conflicts with the following arguments: verbose, print-json, show-summary: all/fail/pass/skip : single-line-summary

Options

`-d`, `--data` (chaîne)

Fournit un fichier de données ou un répertoire de fichiers de données au format JSON ou YAML. Permet de transmettre plusieurs valeurs en utilisant cette option à plusieurs reprises.

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

Pour les arguments de répertoire tels que `data-dir1` ci-dessus, l'analyse n'est prise en charge que pour les fichiers portant les extensions suivantes : .yaml, .yml, .json, .jsn, .template

Si vous spécifiez le `--payload` drapeau, ne spécifiez pas l'`--data`option.

`-i`, `--input-parameters` (chaîne)

Fournit un fichier de paramètres ou un répertoire de fichiers de paramètres au format JSON ou YAML qui spécifie les paramètres supplémentaires à utiliser ainsi que les fichiers de données à utiliser comme contexte combiné. Tous les fichiers de paramètres transmis en entrée sont fusionnés et ce contexte combiné est à nouveau fusionné avec chaque fichier passé en argument pour`data`. De ce fait, chaque fichier est censé contenir des propriétés qui s'excluent mutuellement, sans aucun chevauchement. Permet de transmettre plusieurs valeurs en utilisant cette option à plusieurs reprises.

Pour les arguments de répertoire, l'analyse n'est prise en charge que pour les fichiers portant les extensions suivantes : .yaml, .yml, .json, .jsn, .template

`-o`, `--output-format` (chaîne)

Spécifie le format de sortie.

Par défaut : `single-line-summary`

Valeurs autorisées : `json` | `yaml` | `single-line-summary` | `junit` | `sarif`

`-r`, `--rules` (chaîne)

Fournit un fichier de règles ou un répertoire de fichiers de règles. Permet de transmettre plusieurs valeurs en utilisant cette option à plusieurs reprises.

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

Pour les arguments de répertoire tels que `rules-dir1` ci-dessus, l'analyse n'est prise en charge que pour les fichiers portant les extensions suivantes : .guard, .ruleset

Si vous spécifiez le `--payload` drapeau, ne spécifiez pas l'`--rules`option.

`--show-summary` (chaîne)

Contrôle si le tableau récapitulatif doit être affiché. `--show-summary fail`(par défaut) ou `--show-summary pass,fail` (afficher uniquement les règles qui ont réussi ou échoué) ou `--show-summary none` (pour le désactiver) ou `--show-summary all` (pour afficher toutes les règles qui réussissent, échouent ou sont ignorées).

Par défaut : `fail`

Valeurs autorisées : `none` | `all` | `pass` | `fail` | `skip`

`-t`, `--type` (chaîne)

Indique le format de vos données d'entrée. Lorsque vous spécifiez le type de données d'entrée, Guard affiche les noms logiques des ressources du CloudFormation modèle dans la sortie. Par défaut, Guard affiche les chemins et les valeurs des propriétés, tels que`Property [/Resources/vol2/Properties/Encrypted`.

Valeurs autorisées : `CFNTemplate`

Exemple

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

Output

Si Guard valide les modèles avec succès, la `validate` commande renvoie un statut de sortie de `0` ($?en bash). Si Guard identifie une violation des règles, la `validate` commande renvoie un rapport d'état des règles qui ont échoué.

```
example.json Status = FAIL
FAILED rules
rules.guard/policy_effect_is_deny    FAIL
---
Evaluation of rules rules.guard against data example.json
--
Property [/path/to/Effect] in data [example.json] is not compliant with
 [policy_effect_is_deny] because provided value ["Allow"] did not match expected value
 ["Deny"]. Error Message [ Policy statement "Effect" must be "Deny".]
```

Consultez aussi

- Validation des données d'entrée par rapport aux règles Guard
- Utilisation des paramètres d'entrée avec les règles Guard

Sécurité dans AWS CloudFormation Guard

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit ceci comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de AWS conformité Programmes de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à Guard, voir AWS Services concernés par programme de conformitéAWS .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre entreprise et la législation et la réglementation applicables.

La documentation suivante vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'installation de Guard en tant que AWS Lambda fonction (cfn-guard-lambda) :

- La sécurité dans le guide de AWS Command Line Interface l'utilisateur
- La sécurité dans le guide AWS Lambda du développeur
- La sécurité dans le guide de Gestion des identités et des accès AWS l'utilisateur

AWS CloudFormation Guard Historique du document

Le tableau suivant décrit les versions de documentation pour AWS CloudFormation Guard.

- Dernière mise à jour de la documentation : 30 juillet 2025
- Dernière version : 3.1.2

Modification	Description	Date
Mise à jour de la documentation	Documentation de référence des commandes Guard CLI mise à jour pour l'aligner sur la mise en œuvre actuelle. Références de version mises à jour vers Guard 3.1.2.	30 juillet 2025
Sortie de la version 3.0.0	La version 3.0.0 introduit les améliorations suivantes : • Rubriques d'introduction et d'installation mises à jour pour la sortie de Guard 3.0.0. • Ajout d'instructions d'installation pour Homebrew etChocolatey. • Informations relatives à la migration des règles Guard mises à jour pour refléter les modifications apportées à Guard version 3.0.0. • Ajout d'un lien visible vers le AWS CloudFormation Guard GitHub dépôt.	30 juin 2023

Sortie de la version 2.1.3	La version 2.1.3 introduit les améliorations suivantes : Des informations sur les améliorations apportées à Guard 2.1.3 ont été ajoutées. Les références à Guard 2.0 ont été mises à jour vers Guard 2.1.3.	9 juin 2023
Sortie de la version 2.0.4	La version 2.0.4 introduit les améliorations suivantes : Le `--payload` drapeau a été ajouté à la `validate` commande. Pour plus d'informations, consultez la section validation dans la référence de la CLI Guard.	19 octobre 2021

Sortie de la version 2.0.3	La version 2.0.3 introduit les améliorations suivantes :
	• Vous pouvez fournir des noms de test pour chaque test dans votre fichier de test unitaire. Pour de plus amples informations, veuillez consulter Règles de Testing Guard.
	• Les options suivantes ont été ajoutées à la `validate` commande :
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	  Pour plus d'informations, consultez la section validation dans la référence de la CLI Guard.
		27 Juillet 2021

Première version	Publication initiale du guide de AWS CloudFormation Guard l'utilisateur.	15 juillet 2021

AWS Glossaire

Pour la AWS terminologie la plus récente, consultez le AWS glossaire dans la Glossaire AWS référence.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.