aws

Panduan Pengguna

AWS CloudFormation Guard

AWS CloudFormation Guard: Panduan Pengguna

Table of Contents

Apa itu AWS CloudFormation Guard?

AWS CloudFormation Guard adalah alat evaluasi open-source, tujuan umum, dan evaluasi. policy-as-code Antarmuka baris perintah Guard (CLI) menyediakan bahasa khusus domain (DSL) simple-to-use dan deklaratif yang dapat Anda gunakan untuk mengekspresikan kebijakan sebagai kode. Selain itu, Anda dapat menggunakan CLI perintah untuk memvalidasi hierarkis terstruktur JSON atau YAML data terhadap aturan tersebut. Guard juga menyediakan kerangka pengujian unit bawaan untuk memverifikasi bahwa aturan Anda berfungsi sebagaimana dimaksud.

Guard tidak memvalidasi CloudFormation template untuk sintaks yang valid atau nilai properti yang diizinkan. Anda dapat menggunakan alat cfn-lint untuk melakukan pemeriksaan menyeluruh terhadap struktur template.

Penjaga tidak menyediakan penegakan sisi server. Anda dapat menggunakan CloudFormation Hooks untuk melakukan validasi dan penegakan sisi server, di mana Anda dapat memblokir atau memperingatkan operasi.

Untuk informasi rinci tentang AWS CloudFormation Guard pengembangan, lihat GitHub repositori Guard.

Topik

Apakah Anda pengguna Guard pertama kali?

Jika Anda adalah pengguna pertama kali Guard, kami sarankan Anda mulai dengan membaca bagian berikut:

- Menyiapkan Penjaga— Bagian ini menjelaskan cara menginstal Guard. Dengan Guard, Anda dapat menulis aturan kebijakan menggunakan Guard DSL dan memvalidasi data terstruktur Anda JSON - atau YAML -format terhadap aturan tersebut.

- Aturan Penjaga Menulis— Bagian ini memberikan panduan terperinci untuk menulis aturan kebijakan.
- Aturan Pengujian Guard— Bagian ini memberikan panduan terperinci untuk menguji aturan Anda guna memverifikasi bahwa aturan tersebut berfungsi sebagaimana dimaksud, dan memvalidasi data terstruktur Anda JSON - atau YAML -format terhadap aturan Anda.
- Memvalidasi data input terhadap aturan Guard— Bagian ini memberikan panduan terperinci untuk memvalidasi data terstruktur Anda JSON - atau YAML -format terhadap aturan Anda.
- CLIReferensi penjaga— Bagian ini menjelaskan perintah yang tersedia di GuardCLI.

Fitur penjaga

Dengan menggunakan Guard, Anda dapat menulis aturan kebijakan untuk memvalidasi data terstruktur apa pun JSON - atau YAML -format, termasuk namun tidak terbatas pada templat. AWS CloudFormation Guard mendukung seluruh spektrum end-to-end evaluasi pemeriksaan kebijakan. Aturan berguna dalam domain bisnis berikut:

- Tata kelola dan kepatuhan preventif (pengujian shift-left) — Validasi infrastruktur sebagai kode (IAc) atau komposisi infrastruktur dan layanan terhadap aturan kebijakan yang mewakili praktik terbaik organisasi Anda untuk keamanan dan kepatuhan. Misalnya, Anda dapat memvalidasi CloudFormation template, CloudFormation mengubah set, JSON berbasis file konfigurasi Terraform, atau konfigurasi Kubernetes.
- Tata kelola dan kepatuhan Detektif — Validasi kesesuaian sumber daya Configuration Management Database (CMDB) seperti AWS Config item konfigurasi berbasis (). CIs Misalnya, pengembang dapat menggunakan kebijakan Guard terhadap AWS Config CIs untuk terus memantau keadaan yang dikerahkan AWS dan AWS non-sumber daya, mendeteksi pelanggaran dari kebijakan, dan memulai remediasi.
- Keamanan penerapan — Pastikan bahwa perubahan aman sebelum penerapan. Misalnya, validasi set CloudFormation perubahan terhadap aturan kebijakan untuk mencegah perubahan yang mengakibatkan penggantian sumber daya, seperti mengganti nama tabel Amazon DynamoDB.

Menggunakan Guard dengan CloudFormation Hooks

Anda dapat menggunakan CloudFormation Guard untuk membuat Hook in CloudFormation Hooks. CloudFormation Hooks memungkinkan Anda untuk secara proaktif menegakkan aturan Guard Anda sebelum CloudFormation membuat, memperbarui, atau menghapus operasi dan AWS Cloud Control

API membuat atau memperbarui operasi. Hooks memastikan konfigurasi sumber daya Anda sesuai dengan praktik terbaik keamanan, operasional, dan pengoptimalan biaya organisasi Anda.

Untuk detail tentang cara menggunakan Guard untuk membuat CloudFormation Guard Hooks, lihat Menulis aturan Guard untuk mengevaluasi sumber daya untuk Guard Hooks di Panduan Pengguna AWS CloudFormation Hooks.

Mengakses Guard

Untuk mengakses Guard DSL dan perintah, Anda harus menginstal GuardCLI. Untuk informasi tentang menginstal GuardCLI, lihatMenyiapkan Penjaga.

Praktik terbaik

Tulis aturan sederhana, dan gunakan aturan bernama untuk mereferensikannya dalam aturan lain. Aturan yang rumit bisa sulit untuk dipelihara dan diuji.

Menyiapkan AWS CloudFormation Guard

AWS CloudFormation Guard adalah antarmuka baris perintah sumber terbuka (CLI). Ini memberi Anda bahasa khusus domain yang sederhana untuk menulis aturan kebijakan dan memvalidasi hierarkis JSON dan YAML data terstruktur mereka terhadap aturan tersebut. Aturan dapat mewakili pedoman kebijakan perusahaan yang terkait dengan keamanan, kepatuhan, dan banyak lagi. Data hierarkis terstruktur dapat mewakili infrastruktur cloud yang digambarkan sebagai kode. Misalnya, Anda dapat membuat aturan untuk memastikan bahwa mereka selalu memodelkan bucket Amazon Simple Storage Service (Amazon S3) terenkripsi di template mereka. CloudFormation

Topik berikut memberikan informasi tentang cara menginstal Guard menggunakan sistem operasi yang Anda pilih atau sebagai AWS Lambda fungsi.

Topik

Menginstal Guard untuk Linux dan macOS

Anda dapat menginstal AWS CloudFormation Guard untuk Linux dan macOS dengan menggunakan biner rilis pra-bangun, Cargo, atau melalui Homebrew.

Instal Guard dari biner rilis yang dibuat sebelumnya

Gunakan prosedur berikut untuk menginstal Guard dari biner pra-bangun.

1. Buka terminal, dan jalankan perintah berikut.

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

2. Jalankan perintah berikut untuk mengatur PATH variabel Anda.

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

   Hasil: Anda telah berhasil menginstal Guard dan mengatur PATH variabel.

- (Opsional) Untuk mengkonfirmasi instalasi Guard, jalankan perintah berikut.

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

Perintah mengembalikan output berikut.

```
cfn-guard 3.0.0
```

Instal Guard dari Cargo

Cargo adalah manajer paket Rust. Selesaikan langkah-langkah berikut untuk menginstal Rust, yang mencakup Cargo. Kemudian, instal Guard from Cargo.

1. Jalankan perintah berikut dari terminal, dan ikuti petunjuk di layar untuk menginstal Rust.

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

- (Opsional) Untuk lingkungan Ubuntu, jalankan perintah berikut.

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

2. Konfigurasikan variabel PATH lingkungan Anda, dan jalankan perintah berikut.

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

3. Dengan Cargo diinstal, jalankan perintah berikut untuk menginstal Guard.

```
cargo install cfn-guard
```

Hasil: Anda telah berhasil menginstal Guard.

- (Opsional) Untuk mengkonfirmasi instalasi Guard, jalankan perintah berikut.

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

Perintah mengembalikan output berikut.

```
cfn-guard 3.0.0
```

Instal Guard dari Homebrew

Homebrew adalah manajer paket untuk macOS dan Linux. Selesaikan langkah-langkah berikut untuk menginstal Homebrew. Kemudian, instal Guard dari Homebrew.

1. Jalankan perintah berikut dari terminal, dan ikuti petunjuk di layar untuk menginstal Homebrew.

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

2. Dengan Homebrew diinstal, jalankan perintah berikut untuk menginstal Guard.

   ```
   brew install cloudformation-guard
   ```

   Hasil: Anda telah berhasil menginstal Guard.

   - (Opsional) Untuk mengkonfirmasi instalasi Guard, jalankan perintah berikut.

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

     Perintah mengembalikan output berikut.

     ```
     cfn-guard 3.0.0
     ```

Menginstal Guard untuk Windows

Anda dapat menginstal AWS CloudFormation Guard untuk Windows melalui Cargo atau melalui Chocolatey.

Prasyarat

Untuk membangun Guard dari antarmuka baris perintah, Anda harus menginstal Build Tools for Visual Studio 2019.

1. Unduh alat build Microsoft Visual C ++ dari situs web Build Tools for Visual Studio 2019.

2. Jalankan installer, dan pilih default.

Instal Guard dari Cargo

Cargo adalah manajer paket Rust. Selesaikan langkah-langkah berikut untuk menginstal Rust, yang mencakup Cargo. Kemudian, instal Guard from Cargo.

1. Unduh Rust lalu jalankan rustup-init.exe.
2. Dari command prompt, pilih 1, yang merupakan opsi default.

   Perintah mengembalikan output berikut.

   ```
   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. Untuk menyelesaikan instalasi, tekan tombol Enter.
4. Dengan Cargo diinstal, jalankan perintah berikut untuk menginstal Guard.

   ```
   cargo install cfn-guard
   ```

   Hasil: Anda telah berhasil menginstal Guard.

   - (Opsional) Untuk mengkonfirmasi instalasi Guard, jalankan perintah berikut.

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

     Perintah mengembalikan output berikut.

     ```
     cfn-guard 3.0.0
     ```

Instal Guard dari Chocolatey

Chocolatey adalah manajer paket untuk Windows. Selesaikan langkah-langkah berikut untuk menginstal Chocolatey. Kemudian, instal Guard dari Chocolatey.

1. Ikuti panduan ini untuk menginstal Chocolatey
2. Dengan Chocolatey diinstal, jalankan perintah berikut untuk menginstal Guard.

   ```
   choco install cloudformation-guard
   ```

   Hasil: Anda telah berhasil menginstal Guard.

   - (Opsional) Untuk mengkonfirmasi instalasi Guard, jalankan perintah berikut.

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

     Perintah mengembalikan output berikut.

     ```
     cfn-guard 3.0.0
     ```

Memasang Guard sebagai AWS Lambda fungsi

Anda dapat menginstal AWS CloudFormation Guard melalui Cargo, manajer paket Rust. Guard as an AWS Lambda function (`cfn-guard-lambda`) adalah pembungkus ringan di sekitar Guard (`cfn-guard`) yang dapat digunakan sebagai fungsi Lambda.

Prasyarat

Sebelum Anda dapat menginstal Guard sebagai fungsi Lambda, Anda harus memenuhi prasyarat berikut:

- AWS Command Line Interface (AWS CLI) dikonfigurasi dengan izin untuk menyebarkan dan memanggil fungsi Lambda. Untuk informasi selengkapnya, lihat Mengonfigurasi AWS CLI.
- Peran AWS Lambda eksekusi di AWS Identity and Access Management (IAM). Untuk informasi selengkapnya, lihat peran AWS Lambda eksekusi.
- Di RHEL lingkungan CentOS/, tambahkan repositori `musl-libc` paket ke konfigurasi yum Anda. Untuk informasi lebih lanjut, lihat ngompa/musl-libc.

Instal manajer paket Rust

Cargo adalah manajer paket Rust. Selesaikan langkah-langkah berikut untuk menginstal Rust, yang mencakup Cargo.

1. Jalankan perintah berikut dari terminal, lalu ikuti petunjuk di layar untuk menginstal Rust.

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

   - (Opsional) Untuk lingkungan Ubuntu, jalankan perintah berikut.

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

2. Konfigurasikan variabel `PATH` lingkungan Anda, dan jalankan perintah berikut.

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

Instal Guard sebagai fungsi Lambda (Linux, macOS, atau Unix)

Untuk menginstal Guard sebagai fungsi Lambda, selesaikan langkah-langkah berikut.

1. Dari terminal perintah Anda, jalankan perintah berikut.

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

   - (Opsional) Untuk mengkonfirmasi instalasi Guard sebagai fungsi Lambda, jalankan perintah berikut.

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

     Perintah mengembalikan output berikut.

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

2. Untuk menginstal `musl` dukungan, jalankan perintah berikut.

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

3. Bangun dengan`musl`, lalu jalankan perintah berikut di terminal Anda.

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

Untuk runtime kustom, AWS Lambda memerlukan executable dengan nama `bootstrap` dalam file paket deployment .zip. Ganti nama `cfn-lambda` executable yang dihasilkan menjadi `bootstrap` dan kemudian tambahkan ke arsip.zip.

- Untuk lingkungan macOS, buat file konfigurasi kargo Anda di root proyek Rust atau di dalamnya. `~/.cargo/config`

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

4. Ubah ke direktori `cfn-guard-lambda` root.

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

5. Jalankan perintah berikut di terminal Anda.

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

6. Jalankan perintah berikut untuk mengirimkan `cfn-guard` sebagai fungsi Lambda ke akun Anda.

```
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
```

Untuk membangun dan menjalankan Guard sebagai fungsi Lambda

Untuk memanggil yang dikirimkan `cfn-guard-lambda` sebagai fungsi Lambda, jalankan perintah berikut.

```
aws lambda invoke --function-name cfnGuard \
```

```
  --payload '{"data":"input data","rules":["rule1","rule2"]}' \
  output.json
```

Untuk memanggil struktur permintaan fungsi Lambda

Permintaan untuk `cfn-guard-lambda` mewajibkan bidang berikut:

- `data`— Versi string dari YAML atau JSON template
- `rules`— Versi string dari file set aturan

Prasyarat dan ikhtisar untuk menggunakan aturan Guard

Bagian ini menunjukkan bagaimana Anda dapat menyelesaikan tugas Penjaga inti dalam menulis, menguji, dan memvalidasi aturan terhadap data berformat JSON atau YAML. Selain itu, ini berisi panduan terperinci yang menunjukkan aturan penulisan yang menanggapi kasus penggunaan tertentu.

Topik

Prasyarat

Sebelum Anda dapat menulis aturan kebijakan menggunakan bahasa khusus domain Guard (DSL), Anda harus menginstal antarmuka baris perintah Guard (CLI). Untuk informasi selengkapnya, lihat Menyiapkan Penjaga.

Ikhtisar penggunaan aturan Guard

Saat menggunakan Guard, Anda biasanya melakukan langkah-langkah berikut:

1. Tulis data berformat JSON- atau YAML untuk memvalidasi.
2. Tulis aturan kebijakan Penjaga. Untuk informasi selengkapnya, lihat Aturan Penjaga Menulis.
3. Verifikasi bahwa aturan Anda berfungsi sebagaimana dimaksud dengan menggunakan `test` perintah Guard. Untuk informasi lebih lanjut tentang pengujian unit, lihatAturan Pengujian Guard.
4. Gunakan `validate` perintah Guard untuk memvalidasi data berformat JSON atau YAML Anda terhadap aturan Anda. Lihat informasi yang lebih lengkap di Memvalidasi data input terhadap aturan Guard.

AWS CloudFormation Guard Aturan penulisan

Dalam AWS CloudFormation Guard, aturan adalah policy-as-code aturan. Anda menulis aturan dalam bahasa khusus domain Guard (DSL) yang dapat Anda validasi JSON - atau YAML data yang diformat -. Aturan terdiri dari klausa.

Anda dapat menyimpan aturan yang ditulis menggunakan Guard DSL ke dalam file plaintext yang menggunakan ekstensi file apa pun.

Anda dapat membuat beberapa file aturan dan mengkategorikannya sebagai kumpulan aturan. Set aturan memungkinkan Anda memvalidasi data yang YAML diformat JSON - atau -terhadap beberapa file aturan secara bersamaan.

Topik

Klausul

Klausa adalah ekspresi Boolean yang mengevaluasi ke true (`PASS`) atau false (). `FAIL` Klausul menggunakan operator biner untuk membandingkan dua nilai atau operator unary yang beroperasi pada satu nilai.

Contoh klausa unary

Klausul unary berikut mengevaluasi apakah koleksi `TcpBlockedPorts` kosong.

```
InputParameters.TcpBlockedPorts not empty
```

Klausa unary berikut mengevaluasi apakah `ExecutionRoleArn` properti adalah string.

```
Properties.ExecutionRoleArn is_string
```

Contoh klausa biner

Klausa biner berikut mengevaluasi apakah `BucketName` properti berisi string`encrypted`, terlepas dari casing.

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

Klausul biner berikut mengevaluasi apakah `ReadCapacityUnits` properti kurang dari atau sama dengan 5.000.

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

Sintaks untuk menulis klausa aturan Guard

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

Properti klausa aturan Penjaga

`query`

Ekspresi terpisah dot (`.`) ditulis untuk melintasi data hierarkis. Ekspresi kueri dapat mencakup ekspresi filter untuk menargetkan subset nilai. Kueri dapat ditetapkan ke variabel sehingga Anda dapat menulisnya sekali dan mereferensikannya di tempat lain dalam kumpulan aturan, yang akan memungkinkan Anda mengakses hasil kueri.

Untuk informasi selengkapnya tentang menulis kueri dan pemfilteran, lihat. Mendefinisikan kueri dan pemfilteran

Wajib: Ya

`operator`

Operator unary atau biner yang membantu memeriksa status kueri. Sisi kiri (LHS) dari operator biner harus berupa kueri, dan sisi kanan (RHS) harus berupa kueri atau nilai literal.

Operator biner yang didukung: == != (Sama) | (Tidak sama) | > (Lebih besar dari) | >= (Lebih besar dari atau sama dengan) | < (Kurang dari) | <= (Kurang dari atau sama dengan) | `IN` (Dalam daftar formulir [x, y, z]

Operator unary yang didukung: `exists` | `empty` | `is_string` | `is_list` | `is_struct` `not(!)`

Wajib: Ya

`query|value literal`

Kueri atau nilai literal yang didukung seperti `string` atau`integer(64)`.

Literal nilai yang didukung:

- Semua tipe primitif:`string`,,`integer(64)`,`float(64)`,`bool`, `char` `regex`
- Semua jenis rentang khusus untuk mengekspresikan`integer(64)`,`float(64)`, atau `char` rentang dinyatakan sebagai:
  - `r[<lower_limit>, <upper_limit>]`, yang diterjemahkan ke nilai apa pun k yang memenuhi ekspresi berikut: `lower_limit <= k <= upper_limit`
  - `r[<lower_limit>, <upper_limit>)`, yang diterjemahkan ke nilai apa pun k yang memenuhi ekspresi berikut: `lower_limit <= k < upper_limit`
  - `r(<lower_limit>, <upper_limit>]`, yang diterjemahkan ke nilai apa pun k yang memenuhi ekspresi berikut: `lower_limit < k <= upper_limit`
  - `r(<lower_limit>, <upper_limit>)`,yang diterjemahkan ke nilai apa pun k yang memenuhi ekspresi berikut: `lower_limit < k < upper_limit`
- Array asosiatif (peta) untuk data struktur nilai kunci bersarang. Sebagai contoh:

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

- Array tipe primitif atau tipe array asosiatif

Wajib: Bersyarat; diperlukan ketika operator biner digunakan.

`custom message`

String yang memberikan informasi tentang klausa. Pesan ditampilkan dalam output verbose dari `test` perintah `validate` dan dapat berguna untuk memahami atau men-debug evaluasi aturan pada data hierarkis.

Wajib: Tidak

Menggunakan kueri dalam klausa

Untuk informasi tentang menulis kueri, lihat Mendefinisikan kueri dan pemfilteran dan Menetapkan dan mereferensikan variabel dalam aturan Guard.

Menggunakan operator dalam klausa

Berikut ini adalah contoh CloudFormation template, `Template-1` dan `Template-2`. Untuk mendemonstrasikan penggunaan operator yang didukung, contoh kueri dan klausa di bagian ini mengacu pada contoh templat ini.

Templat-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"
```

Templat-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
```

Contoh klausa yang menggunakan operator unary

- `empty`— Memeriksa apakah koleksi kosong. Anda juga dapat menggunakannya untuk memeriksa apakah kueri memiliki nilai dalam data hierarkis karena kueri menghasilkan koleksi. Anda tidak dapat menggunakannya untuk memeriksa apakah kueri nilai string memiliki string kosong (`""`) yang ditentukan. Untuk informasi selengkapnya, lihat Mendefinisikan kueri dan pemfilteran.

  Klausa berikut memeriksa apakah template memiliki satu atau lebih sumber daya yang ditentukan. Ini mengevaluasi PASS karena sumber daya dengan ID logis `S3Bucket` didefinisikan dalam`Template-1`.

  ```
  Resources !empty
  ```

  Klausa berikut memeriksa apakah satu atau beberapa tag didefinisikan untuk `S3Bucket` sumber daya. Ini mengevaluasi `PASS` karena `S3Bucket` memiliki dua tag yang didefinisikan untuk `Tags` properti di`Template-1`.

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

- `exists`— Memeriksa apakah setiap kemunculan query memiliki nilai dan dapat digunakan sebagai pengganti`!= null`.

  Klausul berikut memeriksa apakah `BucketEncryption` properti didefinisikan untuk. `S3Bucket` Ini mengevaluasi PASS karena `BucketEncryption` didefinisikan untuk `S3Bucket` dalam`Template-1`.

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

> **Note**
>
> The `empty` and `not exists` checks mengevaluasi `true` untuk kunci properti yang hilang saat melintasi data input. Misalnya, jika `Properties` bagian tidak ditentukan dalam templat untuk`S3Bucket`, klausa akan `Resources.S3Bucket.Properties.Tag empty` dievaluasi. `true` `empty`Cek `exists` dan tidak menampilkan jalur JSON penunjuk di dalam dokumen dalam pesan kesalahan. Kedua klausa ini sering memiliki kesalahan pengambilan yang tidak mempertahankan informasi traversal ini.

- `is_string`— Memeriksa apakah setiap kemunculan kueri adalah `string` tipe.

  Klausa berikut memeriksa apakah nilai string ditentukan untuk `BucketName` properti `S3Bucket` sumber daya. Ini mengevaluasi PASS karena nilai string `"MyServiceS3Bucket"` ditentukan untuk `BucketName` in`Template-1`.

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

- `is_list`— Memeriksa apakah setiap kemunculan kueri adalah `list` tipe.

  Klausa berikut memeriksa apakah daftar ditentukan untuk `Tags` properti `S3Bucket` sumber daya. Ini mengevaluasi PASS karena dua pasangan kunci-nilai ditentukan untuk in. `Tags` `Template-1`

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

- `is_struct`— Memeriksa apakah setiap kemunculan kueri adalah data terstruktur.

  Klausa berikut memeriksa apakah data terstruktur ditentukan untuk `BucketEncryption` properti `S3Bucket` sumber daya. Ini mengevaluasi PASS karena `BucketEncryption` ditentukan menggunakan tipe `ServerSideEncryptionConfiguration` properti *`(object)`* di`Template-1`.

> **Note**
>
> Untuk memeriksa keadaan terbalik, Anda dapat menggunakan operator (`not !`) dengan`is_string`,`is_list`, dan `is_struct` operator.

Contoh klausa yang menggunakan operator biner

Klausa berikut memeriksa apakah nilai yang ditentukan untuk `BucketName` properti `S3Bucket` sumber daya di `Template-1` berisi string`encrypt`, terlepas dari casing. Ini dievaluasi `PASS` karena nama bucket yang ditentukan `"MyServiceS3Bucket"` tidak berisi string`encrypt`.

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

Klausa berikut memeriksa apakah nilai yang ditentukan untuk `Size` properti `NewVolume` sumber daya `Template-2` berada dalam rentang tertentu: 50 `<=` `Size` `<=` 200. Ini mengevaluasi `PASS` karena `100` ditentukan untuk`Size`.

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

Klausa berikut memeriksa apakah nilai yang ditentukan untuk `VolumeType` properti `NewVolume` sumber daya di `Template-2` adalah`io1`,`io2`, atau`gp3`. Ini mengevaluasi `PASS` karena `io1` ditentukan untuk`NewVolume`.

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

> **Note**
>
> Contoh kueri di bagian ini menunjukkan penggunaan operator menggunakan sumber daya dengan logis IDs `S3Bucket` dan`NewVolume`. Nama sumber daya sering kali ditentukan pengguna dan dapat secara sewenang-wenang dinamai dalam templat infrastruktur sebagai kode (IAc). Untuk menulis aturan yang generik dan berlaku untuk semua `AWS::S3::Bucket` sumber daya yang didefinisikan dalam template, bentuk kueri yang paling umum digunakan adalah`Resources.*[ Type == 'AWS::S3::Bucket' ]`. Untuk informasi selengkapnya, lihat Mendefinisikan kueri dan pemfilteran detail tentang penggunaan dan jelajahi direktori examples di `cloudformation-guard` GitHub repositori.

Menggunakan pesan khusus dalam klausa

Dalam contoh berikut, klausa untuk `Template-2` menyertakan pesan kustom.

```
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>>
```

Menggabungkan klausa

Di Guard, setiap klausa yang ditulis pada baris baru digabungkan secara implisit dengan klausa berikutnya dengan menggunakan konjungsi (logika Boolean). `and` Lihat contoh berikut ini.

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

Anda juga dapat menggunakan disjungsi untuk menggabungkan klausa dengan klausa berikutnya dengan menentukan `or|OR` di akhir klausa pertama.

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

Dalam klausa Penjaga, disjungsi dievaluasi terlebih dahulu, diikuti oleh konjungsi. Aturan penjaga dapat didefinisikan sebagai gabungan dari disjungsi klausa (an `and|AND` of `or|OR` s) yang mengevaluasi ke () atau `true` (PASS). `false` `FAIL` Ini mirip dengan bentuk normal konjungtif.

Contoh-contoh berikut menunjukkan urutan evaluasi klausa.

```
# (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
```

Semua klausa yang didasarkan pada contoh `Template-1` dapat digabungkan dengan menggunakan konjungsi. Lihat contoh berikut ini.

```
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
```

Menggunakan blok dengan aturan Guard

Blok adalah komposisi yang menghilangkan verbositas dan pengulangan dari serangkaian klausa, kondisi, atau aturan terkait. Ada tiga jenis blok:

- Blok kueri
- `when`blok
- Blok aturan bernama

Blok kueri

Berikut ini adalah klausa yang didasarkan pada contoh. `Template-1` Konjungsi digunakan untuk menggabungkan klausa.

```
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
```

Bagian dari ekspresi kueri di setiap klausa diulang. Anda dapat meningkatkan kemampuan komposisi dan menghapus verbositas dan pengulangan dari sekumpulan klausa terkait dengan jalur kueri awal yang sama dengan menggunakan blok kueri. Kumpulan klausa yang sama dapat ditulis seperti yang ditunjukkan pada contoh berikut.

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

Dalam blok kueri, kueri sebelum blok menetapkan konteks untuk klausa di dalam blok.

Untuk informasi selengkapnya tentang menggunakan blok, lihatMenyusun blok aturan bernama.

whenblok

Anda dapat mengevaluasi blok secara kondisional dengan menggunakan `when` blok, yang mengambil formulir berikut.

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

Kata `when` kunci menunjuk awal `when` blok. `condition`adalah aturan penjaga. Blok hanya dievaluasi jika evaluasi kondisi menghasilkan `true` (PASS).

Berikut ini adalah contoh `when` blok yang didasarkan pada`Template-1`.

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

Klausa dalam `when` blok hanya dievaluasi jika nilai yang ditentukan untuk `BucketName` adalah string. Jika nilai yang `BucketName` ditentukan untuk direferensikan di `Parameters` bagian template seperti yang ditunjukkan pada contoh berikut, klausa dalam `when` blok tidak dievaluasi.

```
Parameters:
   S3BucketName:
```

```
    Type: String

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

Blok aturan bernama

Anda dapat menetapkan nama ke seperangkat aturan (kumpulan aturan), dan kemudian mereferensikan blok validasi modular ini, yang disebut blok aturan bernama, dalam aturan lain. Blok aturan bernama mengambil bentuk berikut.

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

`rule`Kata kunci menunjuk awal dari blok aturan bernama.

`rule name`adalah string yang dapat dibaca manusia yang secara unik mengidentifikasi blok aturan bernama. Ini adalah label untuk aturan Guard yang dirangkum. Dalam penggunaan ini, istilah aturan Penjaga mencakup klausa, blok kueri, blok, dan `when` blok aturan bernama. Nama aturan dapat digunakan untuk merujuk pada hasil evaluasi dari kumpulan aturan yang dirangkum, yang membuat blok aturan bernama dapat digunakan kembali. Nama aturan juga menyediakan konteks tentang kegagalan aturan dalam output `validate` dan `test` perintah. Nama aturan ditampilkan bersama dengan status evaluasi blok (`PASS`,`FAIL`, atau`SKIP`) dalam output evaluasi file aturan. Lihat contoh berikut ini.

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

Anda juga dapat mengevaluasi blok aturan bernama secara kondisional dengan menentukan `when` kata kunci diikuti dengan kondisi setelah nama aturan.

Berikut ini adalah contoh `when` blok yang telah dibahas sebelumnya dalam topik ini.

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

Menggunakan blok aturan bernama, yang sebelumnya juga dapat ditulis sebagai berikut.

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

Anda dapat menggunakan kembali dan mengelompokkan blok aturan bernama dengan aturan Penjaga lainnya. Berikut adalah beberapa contoh.

```
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
    ...
}
```

Mendefinisikan kueri Guard dan pemfilteran

Topik ini mencakup penulisan kueri dan menggunakan pemfilteran saat menulis klausa aturan Guard.

Prasyarat

Penyaringan adalah AWS CloudFormation Guard konsep lanjutan. Kami menyarankan Anda meninjau topik dasar berikut sebelum Anda belajar tentang pemfilteran:

- Apa itu AWS CloudFormation Guard?
- Aturan penulisan, klausa

Mendefinisikan kueri

Ekspresi kueri adalah ekspresi terpisah dot (`.`) sederhana yang ditulis untuk melintasi data hierarkis. Ekspresi kueri dapat mencakup ekspresi filter untuk menargetkan subset nilai. Ketika kueri dievaluasi, mereka menghasilkan kumpulan nilai, mirip dengan set hasil yang dikembalikan dari kueri SQL.

Contoh query berikut mencari AWS CloudFormation template untuk `AWS::IAM::Role` sumber daya.

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

Pertanyaan mengikuti prinsip-prinsip dasar ini:

- Setiap titik (`.`) bagian dari kueri melintasi hierarki ketika istilah kunci eksplisit digunakan, seperti `Resources` atau `Properties.Encrypted`. Jika ada bagian dari kueri yang tidak cocok dengan datum yang masuk, Guard akan melempar kesalahan pengambilan.
- Bagian dot (`.`) dari kueri yang menggunakan wildcard `*` melintasi semua nilai untuk struktur pada tingkat itu.
- Bagian dot (`.`) dari kueri yang menggunakan wildcard array `[*]` melintasi semua indeks untuk array tersebut.

- Semua koleksi dapat disaring dengan menentukan filter di dalam tanda kurung siku`[]`. Koleksi dapat ditemukan dengan cara-cara berikut:
  - Array yang terjadi secara alami dalam datum adalah koleksi. Berikut ini adalah contoh:

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

    Tag: `[{"Key": "Stage", "Value": "PROD"}, {"Key": "App", "Value": "MyService"}]`
  - Saat melintasi semua nilai untuk struktur seperti `Resources.*`
  - Setiap hasil kueri itu sendiri merupakan kumpulan dari mana nilai dapat difilter lebih lanjut. Lihat contoh berikut ini.

    ```
    # 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
    }
    ```

Berikut ini adalah contoh cuplikan CloudFormation template.

```
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
    ...
```

Berdasarkan template ini, jalur yang dilalui adalah `SampleRole` dan nilai akhir yang dipilih adalah. `Type: AWS::IAM::Role`

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

Nilai yang dihasilkan dari query `Resources.*[ Type == 'AWS::IAM::Role' ]` dalam format YAMAL ditunjukkan dalam contoh berikut.

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

Beberapa cara yang dapat Anda gunakan kueri adalah sebagai berikut:

- Tetapkan kueri ke variabel sehingga hasil kueri dapat diakses dengan mereferensikan variabel-variabel tersebut.
- Ikuti kueri dengan blok yang menguji setiap nilai yang dipilih.
- Bandingkan kueri secara langsung dengan klausa dasar.

Menetapkan kueri ke variabel

Guard mendukung penugasan variabel satu-shot dalam lingkup tertentu. Untuk informasi selengkapnya tentang variabel dalam aturan Guard, lihatMenetapkan dan mereferensikan variabel dalam aturan Guard.

Anda dapat menetapkan kueri ke variabel sehingga Anda dapat menulis kueri sekali dan kemudian mereferensikannya di tempat lain dalam aturan Guard Anda. Lihat contoh penugasan variabel berikut yang menunjukkan prinsip kueri yang dibahas nanti di bagian ini.

```
#
# 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'
    }
]
```

Langsung perulangan melalui nilai-nilai dari variabel yang ditugaskan ke kueri

Guard mendukung langsung berjalan terhadap hasil dari kueri. Dalam contoh berikut, `when` blok menguji terhadap`Encrypted`,`VolumeType`, dan `AvailabilityZone` properti untuk setiap `AWS::EC2::Volume` sumber daya yang ditemukan dalam CloudFormation template.

```
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']
        }
    }
}
```

Perbandingan tingkat klausa langsung

Guard juga mendukung kueri sebagai bagian dari perbandingan langsung. Misalnya, lihat yang berikut ini.

```
let resources = Resources.*

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

Dalam contoh sebelumnya, dua klausa (dimulai dengan `some` kata kunci) yang dinyatakan dalam bentuk yang ditampilkan dianggap klausa independen dan dievaluasi secara terpisah.

Klausa tunggal dan formulir klausa blok

Secara keseluruhan, dua contoh klausa yang ditunjukkan di bagian sebelumnya tidak setara dengan blok berikut.

```
let resources = Resources.*

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

Ini memblokir kueri untuk setiap `Tag` nilai dalam koleksi dan membandingkan nilai propertinya dengan nilai properti yang diharapkan. Bentuk gabungan dari klausa di bagian sebelumnya mengevaluasi dua klausa secara independen. Pertimbangkan masukan berikut.

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

Klausul dalam bentuk pertama mengevaluasi. `PASS` Saat memvalidasi klausa pertama dalam bentuk pertama, jalur berikut melintasi`Resources`,, `PropertiesTags`, dan `Key` cocok dengan nilai `NotPRODEnd` dan tidak cocok dengan nilai yang diharapkan. `PROD`

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

```
        - Key: NotPRODEnd
          Value: AppStart
```

Hal yang sama terjadi dengan klausa kedua dari bentuk pertama. Jalur melintasi`Resources`,, `PropertiesTags`, dan `Value` cocok dengan nilainya`AppStart`. Akibatnya, klausa kedua secara independen.

Hasil keseluruhannya adalah a`PASS`.

Namun, formulir blok mengevaluasi sebagai berikut. Untuk setiap `Tags` nilai, itu membandingkan jika keduanya `Key` dan `Value` tidak cocok; `NotAppStart` dan `NotPRODEnd` nilai tidak cocok dalam contoh berikut.

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

Karena evaluasi memeriksa keduanya`Key == /PROD$/`, dan`Value == /^App/`, pertandingan tidak lengkap. Karena itu, hasilnya adalah`FAIL`.

> **Note**
>
> Saat bekerja dengan koleksi, kami sarankan Anda menggunakan formulir klausa blok saat Anda ingin membandingkan beberapa nilai untuk setiap elemen dalam koleksi. Gunakan formulir klausa tunggal ketika koleksi adalah sekumpulan nilai skalar, atau ketika Anda hanya bermaksud untuk membandingkan satu atribut.

Hasil kueri dan klausa terkait

Semua query mengembalikan daftar nilai. Setiap bagian dari traversal, seperti kunci yang hilang, nilai kosong untuk array (`Tags: []`) saat mengakses semua indeks, atau nilai yang hilang untuk peta saat menghadapi map (`Resources: {}`) kosong, dapat menyebabkan kesalahan pengambilan.

Semua kesalahan pengambilan dianggap gagal saat mengevaluasi klausa terhadap kueri tersebut. Satu-satunya pengecualian adalah ketika filter eksplisit digunakan dalam kueri. Saat filter digunakan, klausa terkait dilewati.

Kegagalan blok berikut dikaitkan dengan menjalankan kueri.

- Jika template tidak berisi sumber daya, maka kueri akan dievaluasi`FAIL`, dan klausa tingkat blok terkait juga mengevaluasi. `FAIL`
- Ketika template berisi blok sumber daya kosong seperti`{ "Resources": {} }`, kueri akan dievaluasi`FAIL`, dan klausa tingkat blok terkait juga mengevaluasi. `FAIL`
- Jika template berisi sumber daya tetapi tidak ada yang cocok dengan kueri, maka kueri mengembalikan hasil kosong, dan klausa tingkat blok dilewati.

Menggunakan filter dalam kueri

Filter dalam kueri secara efektif adalah klausa Penjaga yang digunakan sebagai kriteria pemilihan. Berikut ini adalah struktur klausa.

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

Ingatlah poin-poin penting berikut dari AWS CloudFormation Guard Aturan penulisan saat Anda bekerja dengan filter:

- Gabungkan klausa dengan menggunakan Conjunctive Normal Form (CNF).
- Tentukan setiap klausa konjungsi (`and`) pada baris baru.
- Tentukan disjunctions (`or`) dengan menggunakan `or` kata kunci antara dua klausa.

Contoh berikut menunjukkan klausa konjungtif dan disjungtif.

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

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

Menggunakan klausa untuk kriteria seleksi

Anda dapat menerapkan pemfilteran ke koleksi apa pun. Pemfilteran dapat diterapkan langsung pada atribut di input yang sudah menjadi koleksi sepertisecurityGroups: [....]. Anda juga dapat menerapkan pemfilteran terhadap kueri, yang selalu merupakan kumpulan nilai. Anda dapat menggunakan semua fitur klausa, termasuk bentuk normal konjungtif, untuk penyaringan.

Kueri umum berikut sering digunakan saat memilih sumber daya berdasarkan jenis dari CloudFormation template.

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

Query `Resources.*` mengembalikan semua nilai yang ada di `Resources` bagian input. Untuk contoh masukan template diMendefinisikan kueri, query mengembalikan berikut ini.

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

Sekarang, terapkan filter terhadap koleksi ini. Kriteria untuk mencocokkan adalah. `Type == AWS::IAM::Role` Berikut ini adalah output dari query setelah filter diterapkan.

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

Selanjutnya, periksa berbagai klausa untuk `AWS::IAM::Role` sumber daya.

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

Berikut ini adalah contoh query pemfilteran yang memilih semua `AWS::IAM::Policy` dan `AWS::IAM::ManagedPolicy` sumber daya.

```
Resources.*[
```

```
    Type in [ /IAM::Policy/,
              /IAM::ManagedPolicy/ ]
]
```

Contoh berikut memeriksa apakah sumber daya kebijakan ini memiliki yang `PolicyDocument` ditentukan.

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

Membangun kebutuhan penyaringan yang lebih kompleks

Perhatikan contoh item AWS Config konfigurasi berikut untuk informasi grup keamanan ingress dan egress.

```
---
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
```

Perhatikan hal berikut:

- `ipPermissions`(aturan ingress) adalah kumpulan aturan di dalam blok konfigurasi.
- Setiap struktur aturan berisi atribut seperti `ipv4Ranges` dan `ipv6Ranges` untuk menentukan kumpulan blok CIDR.

Mari kita menulis aturan yang memilih aturan masuk apa pun yang memungkinkan koneksi dari alamat IP apa pun, dan memverifikasi bahwa aturan tidak mengizinkan port yang diblokir TCP diekspos.

Mulailah dengan bagian query yang mencakup IPv4, seperti yang ditunjukkan pada contoh berikut.

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

`some`Kata kunci berguna dalam konteks ini. Semua kueri mengembalikan kumpulan nilai yang cocok dengan kueri. Secara default, Guard mengevaluasi bahwa semua nilai yang dikembalikan

sebagai hasil dari kueri dicocokkan dengan pemeriksaan. Namun, perilaku ini mungkin tidak selalu menjadi apa yang Anda butuhkan untuk pemeriksaan. Pertimbangkan bagian input berikut dari item konfigurasi.

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

Ada dua nilai yang hadir untuk`ipv4Ranges`. Tidak semua `ipv4Ranges` nilai sama dengan alamat IP yang dilambangkan dengan. `0.0.0.0/0` Anda ingin melihat apakah setidaknya satu nilai cocok`0.0.0.0/0`. Anda memberi tahu Guard bahwa tidak semua hasil yang dikembalikan dari kueri harus cocok, tetapi setidaknya satu hasil harus cocok. `some`Kata kunci memberitahu Guard untuk memastikan bahwa satu atau lebih nilai dari query yang dihasilkan cocok dengan cek. Jika tidak ada nilai hasil kueri yang cocok, Guard akan melempar kesalahan.

Selanjutnya, tambahkan IPv6, seperti yang ditunjukkan pada contoh berikut.

```
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'
]
```

Akhirnya, dalam contoh berikut, validasi bahwa protokol tidak`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' ]
]
```

Berikut ini adalah aturan lengkapnya.

```
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
                    >>
                }
            }

        }
    }
}
```

Memisahkan koleksi berdasarkan jenis yang terkandung

Saat menggunakan templat konfigurasi infrastruktur sebagai kode (IAc), Anda mungkin menemukan koleksi yang berisi referensi ke entitas lain dalam templat konfigurasi. Berikut ini adalah contoh CloudFormation template yang menjelaskan tugas Amazon Elastic Container Service (Amazon ECS) Container ECS) dengan referensi lokal `TaskRoleArn` ke, referensi `TaskArn` ke, dan referensi string langsung.

```
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'
```

Pertimbangkan kueri berikut.

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

Query ini mengembalikan koleksi nilai-nilai yang berisi semua tiga `AWS::ECS::TaskDefinition` sumber daya yang ditampilkan dalam contoh template. Pisahkan `ecs_tasks` yang berisi referensi `TaskRoleArn` lokal dari yang lain, seperti yang ditunjukkan pada contoh berikut.

```
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
    >>
}
```

Menetapkan dan mereferensikan variabel dalam aturan Guard

Anda dapat menetapkan variabel dalam file AWS CloudFormation Guard aturan untuk menyimpan informasi yang ingin Anda referensikan dalam aturan Guard Anda. Guard mendukung penugasan variabel satu tembakan. Variabel dievaluasi dengan malas, artinya Guard hanya mengevaluasi variabel ketika aturan dijalankan.

Topik

Menetapkan variabel

Gunakan `let` kata kunci untuk menginisialisasi dan menetapkan variabel. Sebagai praktik terbaik, gunakan kasus ular untuk nama variabel. Variabel dapat menyimpan literal statis atau properti dinamis yang dihasilkan dari kueri. Dalam contoh berikut, variabel `ecs_task_definition_task_role_arn` menyimpan nilai string statis`arn:aws:iam:123456789012:role/my-role-name`.

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

Dalam contoh berikut, variabel `ecs_tasks` menyimpan hasil kueri yang mencari semua `AWS::ECS::TaskDefinition` sumber daya dalam AWS CloudFormation template. Anda dapat merujuk `ecs_tasks` untuk mengakses informasi tentang sumber daya tersebut ketika Anda menulis aturan.

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

Variabel referensi

Gunakan `%` awalan untuk referensi variabel.

Berdasarkan contoh `ecs_task_definition_task_role_arn` variabel diMenetapkan variabel, Anda dapat referensi `ecs_task_definition_task_role_arn` di `query|value literal`

bagian klausa aturan Guard. Menggunakan referensi itu memastikan bahwa nilai yang ditentukan untuk `TaskDefinitionArn` properti sumber `AWS::ECS::TaskDefinition` daya apa pun dalam CloudFormation template adalah nilai string statis`arn:aws:iam:123456789012:role/my-role-name`.

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

Berdasarkan contoh `ecs_tasks` variabel di[Menetapkan variabel](), Anda dapat mereferensikan `ecs_tasks` dalam kueri (misalnya, %ecs_tasks.properties). Pertama, Guard mengevaluasi variabel `ecs_tasks` dan kemudian menggunakan nilai yang dikembalikan untuk melintasi hierarki. Jika variabel `ecs_tasks` menyelesaikan nilai non-string, maka Guard melempar kesalahan.

> **Note**
>
> Saat ini, Guard tidak mendukung variabel referensi di dalam pesan kesalahan khusus.

Lingkup variabel

Lingkup mengacu pada visibilitas variabel yang didefinisikan dalam file aturan. Sebuah nama variabel hanya dapat digunakan sekali dalam lingkup. Ada tiga tingkat di mana variabel dapat dideklarasikan, atau tiga kemungkinan cakupan variabel:

- File-level — Biasanya dideklarasikan di bagian atas file aturan, Anda dapat menggunakan variabel tingkat file di semua aturan dalam file aturan. Mereka terlihat ke seluruh file.

  Dalam contoh berikut aturan file, variabel `ecs_task_definition_task_role_arn` dan `ecs_task_definition_execution_role_arn` diinisialisasi pada file-tingkat.

  ```
  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
}
```

- Rule-level — Dideklarasikan dalam aturan, variabel tingkat aturan hanya terlihat oleh aturan tertentu. Referensi apa pun di luar aturan menghasilkan kesalahan.

  Dalam contoh berikut aturan file, variabel `ecs_task_definition_task_role_arn` dan `ecs_task_definition_execution_role_arn` diinisialisasi pada aturan-tingkat. Hanya `ecs_task_definition_task_role_arn` dapat direferensikan dalam aturan `check_ecs_task_definition_task_role_arn` bernama. Anda hanya dapat mereferensikan `ecs_task_definition_execution_role_arn` variabel dalam aturan `check_ecs_task_definition_execution_role_arn` bernama.

```
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
}
```

- Tingkat blok — Dideklarasikan dalam blok, seperti `when` klausa, variabel tingkat blok hanya terlihat oleh blok tertentu. Referensi apa pun di luar blok menghasilkan kesalahan.

  Dalam contoh berikut aturan file, variabel `ecs_task_definition_task_role_arn` dan `ecs_task_definition_execution_role_arn` diinisialisasi pada blok-tingkat dalam blok tipe. `AWS::ECS::TaskDefinition` Anda hanya dapat mereferensikan `ecs_task_definition_execution_role_arn` variabel `ecs_task_definition_task_role_arn` dan dalam blok `AWS::ECS::TaskDefinition` tipe untuk aturan masing-masing.

```
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
    }
}
```

Contoh variabel dalam file aturan Guard

Bagian berikut memberikan contoh penugasan variabel statis dan dinamis.

Penugasan statis

Berikut ini adalah contoh CloudFormation template.

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

Berdasarkan template ini, Anda dapat menulis aturan yang disebut `check_ecs_task_definition_task_role_arn` yang memastikan bahwa `TaskRoleArn` properti dari semua sumber daya `AWS::ECS::TaskDefinition` template adalah`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
}
```

Dalam lingkup aturan, Anda dapat menginisialisasi variabel yang dipanggil `ecs_task_definition_task_role_arn` dan menetapkan untuk itu nilai string statis. `'arn:aws:iam::123456789012:role/my-role-name'` Klausa aturan memeriksa apakah nilai yang ditentukan untuk `TaskRoleArn` properti `EcsTask` sumber daya adalah `arn:aws:iam::123456789012:role/my-role-name` dengan mereferensikan `ecs_task_definition_task_role_arn` variabel di bagian tersebut`query|value literal`.

Penugasan dinamis

Berikut ini adalah contoh CloudFormation template.

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

Berdasarkan template ini, Anda dapat menginisialisasi variabel yang disebut `ecs_tasks` dalam lingkup file dan menetapkan untuk itu query. `Resources.*[ Type == 'AWS::ECS::TaskDefinition'` Guard menanyakan semua sumber daya dalam template input dan menyimpan informasi tentang mereka di`ecs_tasks`. Anda juga dapat menulis aturan yang disebut `check_ecs_task_definition_task_role_arn` yang memastikan bahwa `TaskRoleArn` properti dari semua sumber daya `AWS::ECS::TaskDefinition` template `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'
}
```

Klausa aturan memeriksa apakah nilai yang ditentukan untuk `TaskRoleArn` properti `EcsTask` sumber daya adalah `arn:aws:iam::123456789012:role/my-role-name` dengan mereferensikan `ecs_task_definition_task_role_arn` variabel di bagian tersebut`query`.

Menegakkan konfigurasi AWS CloudFormation template

Mari kita telusuri contoh kasus penggunaan produksi yang lebih kompleks. Dalam contoh ini, kami menulis aturan Penjaga untuk memastikan kontrol yang lebih ketat tentang bagaimana ECS tugas Amazon didefinisikan.

Berikut ini adalah contoh CloudFormation template.

```
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'
```

Berdasarkan template ini, kami menulis aturan berikut untuk memastikan bahwa persyaratan ini terpenuhi:

- Setiap `AWS::ECS::TaskDefinition` sumber daya dalam template memiliki peran tugas dan peran eksekusi yang dilampirkan.
- Peran tugas dan peran eksekusi adalah AWS Identity and Access Management (IAM) peran.
- Peran didefinisikan dalam template.
- `PermissionsBoundary`Properti ditentukan untuk setiap peran.

```
# 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
    }
}
```

Menyusun blok aturan bernama di AWS CloudFormation Guard

Saat menulis blok aturan bernama menggunakan AWS CloudFormation Guard, Anda dapat menggunakan dua gaya komposisi berikut:

- Ketergantungan bersyarat
- Ketergantungan korelasional

Menggunakan salah satu dari gaya komposisi ketergantungan ini membantu meningkatkan kegunaan kembali dan mengurangi verbositas dan pengulangan dalam blok aturan bernama.

Topik

Prasyarat

Pelajari tentang blok aturan bernama dalam aturan Menulis.

Komposisi ketergantungan bersyarat

Dalam gaya komposisi ini, evaluasi `when` blok atau blok aturan bernama memiliki ketergantungan bersyarat pada hasil evaluasi dari satu atau lebih blok atau klausa aturan bernama lainnya. Contoh berikut file Guard rules berisi blok named-rule yang menunjukkan dependensi kondisional.

```
# 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
    ...
}
```

Dalam contoh file aturan sebelumnya, `Example-1` memiliki hasil yang mungkin sebagai berikut:

- Jika `rule_name_A` dievaluasi`PASS`, aturan Penjaga yang dienkapsulasi oleh dievaluasi. `rule_name_B`
- Jika `rule_name_A` dievaluasi`FAIL`, aturan Penjaga yang dienkapsulasi oleh `rule_name_B` tidak dievaluasi. `rule_name_B`mengevaluasi untuk`SKIP`.

- Jika `rule_name_A` dievaluasi`SKIP`, aturan Penjaga yang dienkapsulasi oleh `rule_name_B` tidak dievaluasi. `rule_name_B`mengevaluasi untuk`SKIP`.

> **Note**
>
> Kasus ini terjadi jika `rule_name_A` secara kondisional tergantung pada aturan yang mengevaluasi `FAIL` dan menghasilkan `rule_name_A` evaluasi. `SKIP`

Berikut ini adalah contoh item konfigurasi database manajemen konfigurasi (CMDB) dari AWS Config item untuk informasi grup keamanan ingress dan egress. Contoh ini menunjukkan komposisi ketergantungan bersyarat.

```
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
                }
            }
        }
    }
}
```

Dalam contoh sebelumnya, tergantung secara `check_parameter_validity` kondisional `check_resource_type_and_parameter` dan bergantung pada `check_ip_procotol_and_port_range_validity` kondisional. `check_parameter_validity` Berikut ini adalah item konfigurasi database manajemen konfigurasi (CMDB) yang sesuai dengan aturan sebelumnya.

```
---
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
```

Komposisi ketergantungan korelasional

Dalam gaya komposisi ini, evaluasi when blok atau blok aturan bernama memiliki ketergantungan korelasional pada hasil evaluasi dari satu atau lebih aturan Penjaga lainnya. Ketergantungan korelasional dapat dicapai sebagai berikut.

```
# 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
    ...
}
```

Untuk membantu Anda memahami komposisi ketergantungan korelasional, tinjau contoh berikut dari file aturan 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'
}
```

Dalam file aturan sebelumnya, `ensure_elbs_are_internal_and_secure` memiliki ketergantungan korelasional pada. `ensure_all_elbs_are_secure` Berikut ini adalah contoh CloudFormation template yang sesuai dengan aturan sebelumnya.

```
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...'
```

Menulis klausa untuk melakukan evaluasi sadar konteks

AWS CloudFormation Guard klausa dievaluasi terhadap data hierarkis. Mesin evaluasi Guard menyelesaikan kueri terhadap data yang masuk dengan mengikuti data hierarkis seperti yang ditentukan, menggunakan notasi putus-putus sederhana. Seringkali, beberapa klausa diperlukan untuk mengevaluasi terhadap peta data atau koleksi. Guard menyediakan sintaks yang nyaman untuk menulis klausa tersebut. Mesin sadar kontekstual dan menggunakan data terkait yang terkait untuk evaluasi.

Berikut ini adalah contoh konfigurasi Kubernetes Pod dengan kontainer, di mana Anda dapat menerapkan evaluasi konteks-sadar.

```
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
```

Anda dapat membuat klausa Guard untuk mengevaluasi data ini. Saat mengevaluasi file aturan, konteksnya adalah seluruh dokumen input. Berikut ini adalah contoh klausul yang memvalidasi limit enforcement untuk container yang ditentukan dalam sebuah 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
            >>
        }
```

```
    }
}
```

Pemahaman **`context`** dalam evaluasi

Pada tingkat blok aturan, konteks yang masuk adalah dokumen lengkap. Evaluasi untuk `when` kondisi terjadi terhadap konteks root yang masuk ini di mana `kind` atribut `apiVersion` and berada. Dalam contoh sebelumnya, kondisi ini mengevaluasi`true`.

Sekarang, lintasi hierarki yang `spec.containers[*]` ditunjukkan pada contoh sebelumnya. Untuk setiap lintasan hierarki, nilai konteks berubah sesuai dengan itu. Setelah traversal `spec` blok selesai, konteksnya berubah, seperti yang ditunjukkan pada contoh berikut.

```
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
```

Setelah melintasi `containers` atribut, konteksnya ditampilkan dalam contoh berikut.

```
- 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
```

Memahami loop

Anda dapat menggunakan ekspresi `[*]` untuk mendefinisikan loop untuk semua nilai yang terkandung dalam array untuk `containers` atribut. Blok dievaluasi untuk setiap elemen di dalamnya`containers`. Dalam cuplikan aturan contoh sebelumnya, klausa yang terkandung di dalam blok menentukan pemeriksaan yang akan divalidasi terhadap definisi kontainer. Blok klausa yang terkandung di dalamnya dievaluasi dua kali, satu kali untuk setiap definisi kontainer.

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

Untuk setiap iterasi, nilai konteks adalah nilai pada indeks yang sesuai.

> **Note**
>
> Satu-satunya format akses indeks yang didukung adalah `[<integer>]` atau`[*]`. Saat ini, Guard tidak mendukung rentang seperti`[2..4]`.

Array

Seringkali di tempat-tempat di mana array diterima, nilai tunggal juga diterima. Misalnya, jika hanya ada satu kontainer, array dapat dijatuhkan dan input berikut diterima.

```
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
```

Jika atribut dapat menerima array, pastikan bahwa aturan Anda menggunakan formulir array. Dalam contoh sebelumnya, Anda menggunakan `containers[*]` dan tidak. `containers` Guard mengevaluasi dengan benar saat melintasi data ketika hanya menemukan bentuk nilai tunggal.

> **Note**
>
> Selalu gunakan formulir array saat mengekspresikan akses untuk klausa aturan ketika atribut menerima array. Guard mengevaluasi dengan benar bahkan dalam kasus bahwa satu nilai digunakan.

Menggunakan formulir **`spec.containers[*]`** alih-alih **`spec.containers`**

Kueri penjaga mengembalikan kumpulan nilai yang diselesaikan. Saat Anda menggunakan formulir`spec.containers`, nilai yang diselesaikan untuk kueri berisi array yang dirujuk oleh`containers`, bukan elemen di dalamnya. Ketika Anda menggunakan formulir`spec.containers[*]`, Anda merujuk ke setiap elemen individu yang terkandung. Ingatlah untuk menggunakan `[*]` formulir setiap kali Anda berniat untuk mengevaluasi setiap elemen yang terkandung dalam array.

Menggunakan **`this`** untuk mereferensikan nilai konteks saat ini

Saat Anda membuat aturan Guard, Anda dapat mereferensikan nilai konteks dengan menggunakan`this`. Seringkali`this`, implisit karena terikat pada nilai konteks. Misalnya,`this.apiVersion`,`this.kind`, dan `this.spec` terikat pada root atau dokumen.

Sebaliknya, `this.resources` terikat pada setiap nilai untuk`containers`, seperti `/spec/containers/0/` dan`/spec/containers/1`. Demikian pula, `this.cpu` dan `this.memory` memetakan ke batas, secara khusus `/spec/containers/0/resources/limits` dan`/spec/containers/1/resources/limits`.

Dalam contoh berikutnya, aturan sebelumnya untuk konfigurasi Kubernetes Pod ditulis ulang untuk digunakan secara eksplisit. `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
            >>
        }
    }
}
```

Anda tidak perlu menggunakan secara `this` eksplisit. Namun, `this` referensi dapat berguna saat bekerja dengan skalar, seperti yang ditunjukkan pada contoh berikut.

```
InputParameters.TcpBlockedPorts[*] {
    this in r[0, 65535)
    <<
        result: NON_COMPLIANT
```

```
        message: TcpBlockedPort not in range (0, 65535)
    >>
}
```

Pada contoh sebelumnya, `this` digunakan untuk merujuk ke setiap nomor port.

Potensi kesalahan dengan penggunaan implisit **`this`**

Saat membuat aturan dan klausa, ada beberapa kesalahan umum saat mereferensikan elemen dari nilai konteks implisit. `this` Misalnya, pertimbangkan datum masukan berikut untuk dievaluasi (ini harus lulus).

```
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
```

Ketika diuji terhadap template sebelumnya, aturan berikut menghasilkan kesalahan karena membuat asumsi yang salah untuk memanfaatkan implisit. `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
                >>
            }
        }
    }
}
```

Untuk menelusuri contoh ini, simpan file aturan sebelumnya dengan nama `any_ip_ingress_check.guard` dan data dengan nama file. `ip_ingress.yaml` Kemudian, jalankan `validate` perintah berikut dengan file-file ini.

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

Dalam output berikut, mesin menunjukkan bahwa usahanya untuk mengambil properti `InputParameters.TcpBlockedPorts[*]` pada nilai`/configuration/ipPermissions/0`, `/configuration/ipPermissions/1` gagal.

```
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[*]
```

Untuk membantu memahami hasil ini, tulis ulang aturan menggunakan referensi `this` eksplisit.

```
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.InputParametersreferensi` setiap nilai yang terkandung di dalam variabel`any_ip_permissions`. Kueri yang ditetapkan ke variabel memilih `configuration.ipPermissions` nilai yang cocok. Kesalahan menunjukkan upaya untuk mengambil `InputParamaters` dalam konteks ini, tetapi `InputParameters` berada dalam konteks root.

Blok dalam juga mereferensikan variabel yang berada di luar cakupan, seperti yang ditunjukkan pada contoh berikut.

```
{
    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`mengacu pada setiap nilai port di`[21, 22, 110]`, tetapi juga mengacu pada `fromPort` dan`toPort`. Keduanya termasuk dalam lingkup blok luar.

Menyelesaikan kesalahan dengan penggunaan implisit **`this`**

Gunakan variabel untuk secara eksplisit menetapkan dan mereferensikan nilai. Pertama, `InputParameter.TcpBlockedPorts` adalah bagian dari konteks input (root). Pindah `InputParameter.TcpBlockedPorts` keluar dari blok dalam dan tetapkan secara eksplisit, seperti yang ditunjukkan pada contoh berikut.

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

Kemudian, lihat variabel ini secara eksplisit.

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

Lakukan hal yang sama untuk `this` referensi batin di dalam`%ports`.

Namun, semua kesalahan belum diperbaiki karena loop di dalamnya `ports` masih memiliki referensi yang salah. Contoh berikut menunjukkan penghapusan referensi yang salah.

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

```
}
```

Selanjutnya, jalankan `validate` perintah lagi. Kali ini, berlalu.

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

Berikut ini adalah output dari `validate` perintah.

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

Untuk menguji pendekatan ini untuk kegagalan, contoh berikut menggunakan perubahan payload.

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

90 berada dalam kisaran 89—109 yang memiliki IPv6 alamat apa pun yang diizinkan. Berikut ini adalah output dari `validate` perintah setelah menjalankannya lagi.

```
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 Aturan pengujian

Anda dapat menggunakan kerangka pengujian unit AWS CloudFormation Guard bawaan untuk memverifikasi bahwa aturan Guard berfungsi sebagaimana dimaksud. Bagian ini memberikan panduan tentang cara menulis file pengujian unit dan cara menggunakannya untuk menguji file aturan Anda dengan perintah. `test`

File pengujian unit Anda harus memiliki salah satu ekstensi berikut: `.json`, `.JSON`, `.jsn`, `.yaml`, `.YAML`, atau `.yml`.

Topik

- Prasyarat
- Ikhtisar file pengujian unit Guard
- Panduan penulisan file pengujian unit aturan Penjaga

Prasyarat

Tulis aturan Guard untuk mengevaluasi data input Anda. Untuk informasi selengkapnya, lihat Aturan Penjaga Menulis.

Ikhtisar file pengujian unit Guard

File pengujian unit penjaga adalah JSON - atau YAML -file berformat yang berisi beberapa input dan hasil yang diharapkan untuk aturan yang ditulis di dalam file aturan Guard. Mungkin ada beberapa

sampel untuk menilai harapan yang berbeda. Kami menyarankan Anda memulai dengan menguji input kosong dan kemudian menambahkan informasi secara progresif untuk menilai berbagai aturan dan klausa.

Selain itu, kami menyarankan Anda memberi nama file pengujian unit menggunakan akhiran `_test.json` atau`_tests.yaml`. Misalnya, jika Anda memiliki file aturan bernamamy`_rules.guard`, beri nama file pengujian unit Andamy`_rules_tests.yaml`.

Sintaks

Berikut ini menunjukkan sintaks file pengujian unit dalam YAML format.

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

Properti

Berikut ini adalah properti dari file uji Guard.

`input`

Data untuk menguji aturan Anda. Sebaiknya tes pertama Anda menggunakan input kosong, seperti yang ditunjukkan pada contoh berikut.

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

Untuk pengujian selanjutnya, tambahkan data input untuk diuji.

Wajib: Ya

`expectations`

Hasil yang diharapkan ketika aturan tertentu dievaluasi terhadap data input Anda. Tentukan satu atau beberapa aturan yang ingin Anda uji selain hasil yang diharapkan untuk setiap aturan. Hasil yang diharapkan harus salah satu dari yang berikut:

- `PASS`— Ketika dijalankan terhadap data input Anda, aturan akan dievaluasi`true`.
- `FAIL`— Ketika dijalankan terhadap data input Anda, aturan akan dievaluasi`false`.
- `SKIP`— Saat dijalankan terhadap data input Anda, aturan tidak dipicu.

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

Wajib: Ya

Panduan penulisan file pengujian unit aturan Penjaga

Berikut ini adalah file aturan bernama`api_gateway_private.guard`. Maksud dari aturan ini adalah untuk memeriksa apakah semua jenis sumber daya Amazon API Gateway yang ditentukan dalam CloudFormation templat hanya digunakan untuk akses pribadi. Ini juga memeriksa apakah setidaknya satu pernyataan kebijakan memungkinkan akses dari cloud pribadi virtual (VPC).

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

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

# 2) All AWS::ApiGateway::RestApi resources deployed must have at least one AWS
 Identity and Access Management (IAM) policy condition key to allow access from a VPC.
#
# Expectations:
# 1) SKIP when there are no AWS::ApiGateway::RestApi resources in the template.
# 2) PASS when:
#     ALL AWS::ApiGateway::RestApi resources in the template have
 the EndpointConfiguration property set to Type: PRIVATE.
#     ALL AWS::ApiGateway::RestApi resources in the template have one IAM condition key
 specified in the Policy property with aws:sourceVpc or :SourceVpc.
# 3) FAIL otherwise.

#
```

```
#

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

    }
}

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

Panduan ini menguji maksud aturan pertama: Semua `AWS::ApiGateway::RestApi` sumber daya yang digunakan harus bersifat pribadi.

1. Buat file pengujian unit `api_gateway_private_tests.yaml` yang disebut yang berisi tes awal berikut. Dengan pengujian awal, tambahkan input kosong dan harapkan aturan `check_rest_api_is_private` akan dilewati karena tidak ada `AWS::ApiGateway::RestApi` sumber daya sebagai input.

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

2. Jalankan tes pertama di terminal Anda menggunakan `test` perintah. Untuk `--rules-file` parameter, tentukan file aturan Anda. Untuk `--test-data` parameter, tentukan file pengujian unit Anda.

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

Hasil untuk tes pertama adalahPASS.

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

3. Tambahkan tes lain ke file pengujian unit Anda. Sekarang, perluas pengujian untuk memasukkan sumber daya kosong. Berikut ini adalah `api_gateway_private_tests.yaml` file yang diperbarui.

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

4. Jalankan `test` dengan file pengujian unit yang diperbarui.

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

Hasil untuk tes kedua adalahPASS.

```
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. Tambahkan dua tes lagi ke file pengujian unit Anda. Perpanjang pengujian untuk memasukkan yang berikut:

   - `AWS::ApiGateway::RestApi`Sumber daya tanpa properti yang ditentukan.

     > **Note**
     >
     > Ini bukan CloudFormation template yang valid, tetapi berguna untuk menguji apakah aturan berfungsi dengan benar bahkan untuk input yang salah bentuk.

     Harapkan pengujian ini akan gagal karena `EndpointConfiguration` properti tidak ditentukan dan oleh karena itu tidak disetel ke`PRIVATE`.

   - `AWS::ApiGateway::RestApi`Sumber daya yang memenuhi intent pertama dengan `EndpointConfiguration` properti yang disetel ke `PRIVATE` tetapi tidak memenuhi maksud kedua karena tidak memiliki pernyataan kebijakan yang ditentukan. Harapkan tes ini akan lulus.

   Berikut ini adalah file pengujian unit yang diperbarui.

```
---
- 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. Jalankan `test` dengan file pengujian unit yang diperbarui.

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

Hasil ketiga adalah`FAIL`, dan hasil keempat adalah`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. Komentari tes 1-3 dalam file pengujian unit Anda. Akses konteks verbose untuk tes keempat saja. Berikut ini adalah file pengujian unit yang diperbarui.

```
---
#- 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. Periksa hasil evaluasi dengan menjalankan `test` perintah di terminal Anda, menggunakan `--verbose` bendera. Konteks verbose berguna untuk memahami evaluasi. Dalam hal ini, ini memberikan informasi rinci tentang mengapa tes keempat berhasil dengan `PASS` hasil.

```
cfn-guard test \
  --rules-file api_gateway_private.guard \
```

```
  --test-data api_gateway_private_tests.yaml \
  --verbose
```

Berikut adalah output dari run itu.

```
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)
```

Pengamatan utama dari output adalah garis`Clause(Location[file:api_gateway_private.guard, line:22, column:5], Check: Properties.EndpointConfiguration.Types[*] EQUALS String("PRIVATE")), PASS)`, yang menyatakan bahwa cek lulus. Contoh ini juga menunjukkan kasus `Types` di mana diharapkan menjadi array, tetapi satu nilai diberikan. Dalam hal ini, Guard terus mengevaluasi dan memberikan hasil yang benar.

9. Tambahkan kasus uji seperti kasus uji keempat ke file pengujian unit Anda untuk `AWS::ApiGateway::RestApi` sumber daya dengan `EndpointConfiguration` properti

yang ditentukan. Kasus uji akan gagal alih-alih lulus. Berikut ini adalah file pengujian unit yang diperbarui.

```
---
#- 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. Jalankan `test` perintah dengan file pengujian unit yang diperbarui menggunakan `--verbose` bendera.

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

Hasilnya `FAIL` seperti yang diharapkan karena `REGIONAL` ditentukan untuk `EndpointConfiguration` tetapi tidak diharapkan.

```
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)
```

Output verbose dari `test` perintah mengikuti struktur file aturan. Setiap blok dalam file aturan adalah blok dalam output verbose. Blok paling atas adalah setiap aturan. Jika ada `when` kondisi yang bertentangan dengan aturan, mereka muncul di blok kondisi saudara kandung. Dalam contoh berikut, kondisi `%api_gws !empty` diuji dan berlalu.

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

Setelah kondisi berlalu, kami menguji klausa aturan.

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

`%api_gws`adalah aturan blok yang sesuai dengan `BlockClause` level dalam output (baris: 21). Klausa aturanadalah seperangkat klausa konjungsi (AND), di mana setiap klausa konjungsi adalah sekumpulan disjungsi. `OR` Konjungsi memiliki satu klausa,. `Properties.EndpointConfiguration.Types[*] == "PRIVATE"` Oleh karena itu, output verbose menunjukkan satu klausa. Jalur `/Resources/apiGw/Properties/EndpointConfiguration/Types/1` menunjukkan nilai mana dalam input yang dibandingkan, yang dalam hal ini adalah elemen untuk `Types` diindeks pada 1.

DiMemvalidasi data input terhadap aturan Guard, Anda dapat menggunakan contoh di bagian ini untuk menggunakan `validate` perintah untuk mengevaluasi data input terhadap aturan.

Menggunakan parameter input dengan AWS CloudFormation Guard aturan

AWS CloudFormation Guard memungkinkan Anda untuk menggunakan parameter input untuk pencarian data dinamis selama validasi. Fitur ini sangat berguna ketika Anda perlu mereferensikan data eksternal dalam aturan Anda. Namun, saat menentukan kunci parameter input, Guard mensyaratkan bahwa tidak ada jalur yang bertentangan.

Cara menggunakan

1. Gunakan `-i` tanda `--input-parameters` atau untuk menentukan file yang berisi parameter input. Beberapa file parameter input dapat ditentukan dan akan digabungkan untuk membentuk konteks umum. Kunci parameter input tidak dapat memiliki jalur yang bertentangan.
2. Gunakan `-d` tanda `--data` or untuk menentukan file template aktual yang akan divalidasi.

Contoh penggunaan

1. Buat file parameter input (Misalnya,`network.yaml`):

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

2. Referensikan parameter ini di file aturan penjaga Anda (Misalnya,`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. Buat template data yang gagal (Misalnya,`security_groups_fail.yaml`):

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

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

4. Jalankan perintah validasi:

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

Dalam perintah ini:

- `-r`menentukan file aturan
- `-i`menentukan file parameter masukan
- `-d`menentukan file data (template) yang akan divalidasi

Beberapa parameter masukan

Anda dapat menentukan beberapa file parameter input:

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

Semua file yang ditentukan dengan `-i` akan digabungkan untuk membentuk satu konteks untuk pencarian parameter.

Memvalidasi data input terhadap aturan AWS CloudFormation Guard

Anda dapat menggunakan AWS CloudFormation Guard `validate` perintah untuk memvalidasi data terhadap aturan Guard. Untuk informasi selengkapnya tentang `validate` perintah, termasuk parameter dan opsinya, lihat memvalidasi.

Prasyarat

- Tulis aturan Guard untuk memvalidasi data input Anda. Untuk informasi selengkapnya, lihat Aturan Penjaga Menulis.
- Uji aturan Anda untuk memastikan bahwa mereka bekerja sebagaimana dimaksud. Untuk informasi selengkapnya, lihat Aturan Pengujian Guard.

Menggunakan **`validate`** perintah

Untuk memvalidasi data input Anda terhadap aturan Guard Anda, seperti AWS CloudFormation template, jalankan `validate` perintah Guard. Untuk `--rules` parameter, tentukan nama file aturan. Untuk `--data` parameter, tentukan nama file data input.

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

Jika Guard berhasil memvalidasi template, `validate` perintah mengembalikan status keluar `0` (`$?`dalam bash). Jika Guard mengidentifikasi pelanggaran aturan, `validate` perintah mengembalikan laporan status aturan yang gagal. Gunakan flag ringkasan (`-s all`) untuk melihat pohon evaluasi rinci yang menunjukkan bagaimana Guard mengevaluasi setiap aturan.

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

Memvalidasi beberapa aturan terhadap beberapa file data

Untuk membantu mempertahankan aturan, Anda dapat menulis aturan ke dalam beberapa file dan mengatur aturan sesuai keinginan. Kemudian, Anda dapat memvalidasi beberapa file aturan

terhadap file data atau beberapa file data. `validate`Perintah dapat mengambil direktori file untuk `--rules` opsi `--data` dan. Misalnya, Anda dapat menjalankan perintah berikut di mana `/path/to/dataDirectory` berisi satu atau lebih file data dan `/path/to/ruleDirectory` berisi satu atau beberapa file aturan.

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

Anda dapat menulis aturan untuk memeriksa apakah berbagai sumber daya yang ditentukan dalam beberapa CloudFormation templat memiliki penugasan properti yang sesuai untuk menjamin enkripsi saat istirahat. Untuk kemudahan pencarian dan pemeliharaan, Anda dapat memiliki aturan untuk memeriksa enkripsi saat istirahat di setiap sumber daya dalam file terpisah `s3_bucket_encryption.guard`ec2_volume_encryption.guard, dipanggil,, dan `rds_dbinstance_encrytion.guard` dalam direktori dengan jalur`~/GuardRules/encryption_at_rest`. CloudFormationTemplate yang Anda butuhkan untuk memvalidasi ada di direktori dengan jalur`~/CloudFormation/templates`. Dalam hal ini, jalankan `validate` perintah sebagai berikut.

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

Pemecahan masalah AWS CloudFormation Guard

Jika Anda mengalami masalah saat bekerja dengan AWS CloudFormation Guard, lihat topik di bagian ini.

Topik

Klausul gagal ketika tidak ada sumber daya dari jenis yang dipilih

Ketika kueri menggunakan filter seperti`Resources.*[ Type == 'AWS::ApiGateway::RestApi' ]`, jika tidak ada `AWS::ApiGateway::RestApi` sumber daya dalam input, klausa akan dievaluasi. `FAIL`

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

Untuk menghindari hasil ini, tetapkan filter ke variabel dan gunakan pemeriksaan `when` kondisi.

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

Guard tidak mengevaluasi CloudFormation template dengan referensi bentuk pendek Fn::GetAtt

Penjaga tidak mendukung bentuk pendek fungsi intrinsik. Misalnya, menggunakan`!Join`, `!Sub` dalam AWS CloudFormation templat YAML berformat -tidak didukung. Sebaliknya, gunakan bentuk fungsi CloudFormation intrinsik yang diperluas. Misalnya, gunakan`Fn::Join`, `Fn::Sub` dalam CloudFormation templat YAML berformat -saat mengevaluasi mereka terhadap aturan Guard.

Untuk informasi selengkapnya tentang fungsi intrinsik, lihat referensi fungsi intrinsik di Panduan Pengguna.AWS CloudFormation

Topik pemecahan masalah umum

- Verifikasi bahwa `string` literal tidak berisi string escaped yang disematkan. Saat ini, Guard tidak mendukung string escape yang disematkan dalam `string` literal.
- Verifikasi bahwa `!=` perbandingan Anda membandingkan tipe data yang kompatibel. Misalnya, a `string` dan an `int` bukan tipe data yang kompatibel untuk perbandingan. Saat melakukan `!=` perbandingan, jika nilainya tidak kompatibel, kesalahan terjadi secara internal. Saat ini, kesalahan ditekan dan dikonversi `false` untuk memenuhi PartialEqsifat di Rust.

AWS CloudFormation Guard CLIparameter dan referensi perintah

Parameter dan perintah global berikut tersedia melalui antarmuka baris AWS CloudFormation Guard perintah (CLI).

Topik

Menjaga parameter CLI global

Anda dapat menggunakan parameter berikut dengan AWS CloudFormation Guard CLI perintah apa pun.

`-h`, `--help`

Mencetak informasi bantuan.

`-V`, `--version`

Mencetak informasi versi.

parse-pohon

Menghasilkan pohon parse untuk AWS CloudFormation Guard aturan yang ditentukan dalam file aturan.

Sintaks

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

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

Parameter

`-h, --help`

Mencetak informasi bantuan.

`-j, --print-json`

Mencetak output dalam JSON format.

`-y, --print-yaml`

Mencetak output dalam YAML format.

`-V, --version`

Mencetak informasi versi.

Opsi

`-o, --output`

Menulis pohon yang dihasilkan ke file output.

`-r, --rules`

Menyediakan file aturan.

Contoh

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

aturan

Mengambil file AWS CloudFormation template yang YAML diformat JSON - atau -dan membuat otomatis seperangkat AWS CloudFormation Guard aturan yang cocok dengan properti sumber

daya template. Perintah ini adalah cara yang berguna untuk memulai penulisan aturan atau untuk membuat ready-to-use aturan dari template bagus yang dikenal.

Sintaks

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

Parameter

`-h, --help`

Mencetak informasi bantuan.

`-V, --version`

Mencetak informasi versi.

Opsi

`-o, --output`

Menulis aturan yang dihasilkan ke file output. Mengingat potensi ratusan atau bahkan ribuan aturan muncul, kami sarankan untuk menggunakan opsi ini.

`-t, --template`

Menyediakan path ke file CloudFormation template dalam JSON atau YAML format.

Contoh

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

pengujian

Memvalidasi file AWS CloudFormation Guard aturan terhadap file pengujian unit Guard dalam JSON atau YAML format untuk menentukan keberhasilan aturan individu.

Sintaks

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

Parameter

`-h`, `--help`

Mencetak informasi bantuan.

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

Mengurutkan berdasarkan waktu modifikasi terakhir dalam direktori

`-V`, `--version`

Mencetak informasi versi.

`-v`, `--verbose`

Meningkatkan verbositas output. Dapat ditentukan beberapa kali.

Output verbose mengikuti struktur file aturan Guard. Setiap blok dalam file aturan adalah blok dalam output verbose. Blok paling atas adalah setiap aturan. Jika ada `when` kondisi yang bertentangan dengan aturan, mereka muncul sebagai blok kondisi saudara kandung.

Opsi

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

Memberikan nama file aturan.

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

Menyediakan nama file atau direktori untuk file data dalam salah satu JSON atau YAML format.

argumen

<alphabetical>

Mengurutkan menurut abjad di dalam direktori.

Contoh

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

Output

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

Lihat juga

Aturan Pengujian Guard

validasi

Memvalidasi data terhadap AWS CloudFormation Guard aturan untuk menentukan keberhasilan atau kegagalan.

Sintaks

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

Parameter

`-a, --alphabetical`

Memvalidasi file dalam direktori yang diurutkan menurut abjad.

`-h, --help`

Mencetak informasi bantuan.

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

Memvalidasi file dalam direktori yang diurutkan berdasarkan waktu modifikasi terakhir.

`-P`, `--payload`

Memungkinkan Anda memberikan aturan dan data dalam JSON format berikut melalui`stdin`:

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

Sebagai contoh:

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

Untuk “aturan”, tentukan daftar string file aturan. Untuk “data”, tentukan daftar string file data.

Jika Anda menentukan `--payload` bendera, jangan tentukan `--data` opsi `--rules` atau.

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

Mencetak output dalam JSON format.

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

Menunjukkan kegagalan klausa termasuk ringkasan.

`-V`, `--version`

Mencetak informasi versi.

`-v`, `--verbose`

Meningkatkan verbositas output. Dapat ditentukan beberapa kali.

Opsi

`-d`, `--data` (string)

Menyediakan nama file atau direktori untuk file data dalam salah satu JSON atau YAML format. Jika Anda menyediakan direktori, Guard mengevaluasi aturan yang ditentukan terhadap semua file data dalam direktori. Direktori harus berisi hanya file data; tidak dapat berisi file data dan aturan.

Jika Anda menentukan `--payload` bendera, jangan tentukan `--data` opsi.

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

Menulis ke file output.

Default: `single-line-summary`

Nilai yang diizinkan: `json` | `yaml` | `single-line-summary`

`-r`, `--rules` (string)

Menyediakan nama file aturan atau direktori file aturan. Jika Anda menyediakan direktori, Guard mengevaluasi semua aturan dalam direktori terhadap data yang ditentukan. Direktori harus berisi hanya file aturan; tidak dapat berisi file data dan aturan.

Jika Anda menentukan `--payload` bendera, jangan tentukan `--rules` opsi.

`--show-summary`(tali)

Menentukan verbositas ringkasan evaluasi aturan Guard. Jika Anda menentukan`all`, Guard menampilkan ringkasan lengkap. Jika Anda menentukan`pass,fail`, Guard hanya menampilkan informasi ringkasan untuk aturan yang diteruskan atau gagal. Jika Anda menentukan`none`, Guard tidak menampilkan informasi ringkasan. Secara default, `all` ditentukan.

Nilai yang diizinkan: `all` | `pass,fail` | `none`

`-t`, `--type` (string)

Menyediakan format data input Anda. Saat Anda menentukan tipe data input, Guard menampilkan nama logis sumber daya CloudFormation template dalam output. Secara default, Guard menampilkan jalur dan nilai properti, seperti`Property [/Resources/vol2/Properties/Encrypted`.

Nilai yang diizinkan: `CFNTemplate`

Contoh

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

Output

Jika Guard berhasil memvalidasi template, `validate` perintah mengembalikan status keluar `0` (`$?`dalam bash). Jika Guard mengidentifikasi pelanggaran aturan, `validate` perintah mengembalikan laporan status aturan yang gagal. Gunakan bendera verbose (`-v`) untuk melihat pohon evaluasi terperinci yang menunjukkan bagaimana Guard mengevaluasi setiap aturan.

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

Lihat juga

Memvalidasi data input terhadap aturan Guard

Keamanan di AWS CloudFormation Guard

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program AWS Kepatuhan Program AWS Kepatuhan . Untuk mempelajari tentang program kepatuhan yang berlaku untuk Guard, lihat AWS Layanan dalam Lingkup oleh AWS Layanan Program Kepatuhan .
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup kepekaan data Anda, persyaratan perusahaan, serta peraturan perundangan yang berlaku

Dokumentasi berikut membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menginstal Guard sebagai AWS Lambda fungsi (cfn-guard-lambda):

- Keamanan dalam Panduan AWS Command Line Interface Pengguna
- Keamanan dalam Panduan AWS Lambda Pengembang
- Keamanan dalam Panduan AWS Identity and Access Management Pengguna

AWS CloudFormation Guard Sejarah dokumen

Tabel berikut menjelaskan rilis dokumentasi untuk AWS CloudFormation Guard.

- Pembaruan dokumentasi terbaru: 30 Juni 2023
- Versi Terbaru: 3.0.0

Perubahan	Deskripsi	Tanggal
Versi 3.0.0 rilis	Versi 3.0.0 memperkenalkan perbaikan berikut: • Topik pengantar dan instalasi diperbarui untuk rilis Guard 3.0.0. • Menambahkan insturctions instalasi untuk Homebrew dan Chocolatey. • Informasi terkait migrasi aturan Guard diperbaru i untuk mencerminkan perubahan dalam Guard versi 3.0.0. • Menambahkan tautan yang menonjol ke AWS CloudFormation Guard GitHub repositori.	30 Juni 2023
Versi 2.1.3 rilis	Versi 2.1.3 memperkenalkan perbaikan berikut: Informasi tentang peningkatan Guard 2.1.3 telah ditambahk an. Referensi ke Guard 2.0	9 Juni 2023

	telah diperbarui ke Guard 2.1.3.	
Versi 2.0.4 rilis	Versi 2.0.4 memperkenalkan perbaikan berikut: `--payload` Bendera ditambahkan ke `validate` perintah. Untuk informasi selengkapnya, lihat memvalidasi dalam CLI referensi Penjaga.	19 Oktober 2021

Versi 2.0.3 rilis	Versi 2.0.3 memperkenalkan perbaikan berikut:
	
	• Anda dapat memberikan nama pengujian untuk setiap pengujian dalam file pengujian unit Anda. Untuk informasi selengkapnya, lihat Aturan Pengujian Guard.
	• Opsi berikut ditambahkan ke `validate` perintah:
	  • `--output-format`
	  • `--show-summary`
	  • `--type`
	
	  Untuk informasi selengkapnya, lihat memvalidasi dalam CLI referensi Penjaga.
		27 Juli 2021

Rilis awal	Rilis awal Panduan AWS CloudFormation Guard Pengguna.	15 Juli 2021

AWS Glosarium

Untuk AWS terminologi terbaru, lihat AWS glosarium di Referensi.Glosarium AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.