Checking AWS AMI IDs in Terraform using Regula and Open Policy Agent

Written by: Becki Lee

October 7, 2021 · 10 mins read

Editor's note

This blog originally appeared on fugue.co. Fugue joined Snyk in 2022 and is a key component of Snyk IaC.

Last week we announced Fugue IaC, which enables cloud engineering teams to secure their infrastructure as code (IaC) and cloud runtime environment using the same policies. For running IaC checks locally, Fugue developed Regula, an open source tool built on Open Policy Agent (OPA).

Regula itself can be used with or without Fugue. It comes with hundreds of out-of-the-box rules designed to check Terraform and CloudFormation infrastructure as code (IaC) and Kubernetes manifests. These rules are mapped to the CIS Benchmarks and cover many security vulnerabilities.

But sometimes there are organization-specific policies you want to enforce – for instance, requiring Amazon EC2 instances to use approved AMIs. In situations like this, you can write a **custom rule** using OPA's Rego language, and Regula will apply it to your IaC. That way, if your IaC declares a non-approved AMI, the Regula check will fail.

In this blog post:

- We'll write a custom rule to check AWS EC2 AMIs declared in Terraform, explaining the Rego code line by line.
- We'll use our open source tool Regula to test the rule out against a noncompliant Terraform file.
- We'll fix the noncompliant Terraform.

Note that while this blog post uses Regula to check Terraform HCL, you can also use the same exact custom rule with the Fugue SaaS to check cloud runtime resources. This is thanks to Fugue's Unified Policy Engine, which supports using the same rules at IaC and runtime.

You'll find all the code in this GitHub gist.

Let's go!

Getting started

Download code files

In our GitHub gist, select the Download ZIP button and then extract the files. There are two files:

- approved_ami.rego – a custom rule written in Rego
- ami.tf – a Terraform HCL file declaring two EC2 instances, one compliant and one noncompliant

Install Regula

The next step is to install Regula, if you haven't yet. Homebrew users can execute the following commands in their terminal:

```
brew tap fugue/regula
brew install regula
```

You can alternatively install a prebuilt binary for your platform or, if you prefer, run Regula with Docker.

Writing the custom rule

First, we'll determine what type of rule to write. With Regula, there are two rule types: simple and advanced. Simple rules are useful when you're checking a single resource type. Advanced rules are better for more complex logic involving multiple resources or checking for missing resources.

We're just going to check one Terraform resource type, `aws_instance`, so we'll write a simple rule.

Package declaration

Let's start with the basics. All Regula rules must have a package declaration beginning with `rules.` and ending with a short identifier (line 4):

```
4  package rules.approved_ami
```

Metadata

We can optionally add some metadata (lines 6-18):

```
6  __rego__metadoc__ := {
7    "id": "CUSTOM_0002",
8    "title": "AWS EC2 instances must use approved AMIs",
9    "description": "Per company policy, EC2 instances may only use AMI IDs from a
10   "custom": {
11     "controls": {
12       "CORPORATE-POLICY": [
13         "CORPORATE-POLICY_1.2"
14       ]
15     },
16     "severity": "High"
17   }
18 }
```

This isn't required, but it makes Regula's report output even more informative.

Resource type

Next up, we state which Terraform resource type we want to check (line 20):

```
20 resource_type = "aws_instance"
```

This syntax means that the input will be a single AWS EC2 instance. When Regula applies the rule to our Terraform, it will evaluate a single instance at a time.

Approved AMI set

For simplicity's sake, let's say your organization has blessed these two AMI IDs:

- ami-09e67e426f25ce0d7
- ami-03d5c68bab01f3496

(If you're curious, these are the IDs for Ubuntu Server 20.04 LTS HVM, SSD Volume Type in us-east-1 and us-west-2, respectively.)

Now, we'll create a set containing the approved AMI IDs. We'll call it `approved_amis` (lines 22-26):

```
22 approved_amis = {
23   # Ubuntu Server 20.04 LTS (HVM), SSD Volume Type
24   "ami-09e67e426f25ce0d7", # us-east-1
25   "ami-03d5c68bab01f3496" # us-west-2
26 }
```

The deny rule

Finally, we'll write a `deny` rule. This is where the main logic lives! The deny rule defines the conditions in which a resource should fail the check.

We're going to get a little fancy here and return a custom error message that lists the unapproved AMI ID when an instance fails the Regula check.

What our deny rule should do is check whether the currently evaluated instance's AMI ID is in the approved set, and if it isn't, deny should return an error message for that resource (i.e., produce a failing rule result). `deny` is defined in lines 28-31:

```
28 deny[msg] {
29   not approved_amis[input.ami]
30   msg = sprintf("%s is not an approved AMI ID", [input.ami])
31 }
```

Let's take a closer look at the logic:

```
29 not approved_amis[input.ami]
```

`input.ami` represents the `ami` attribute of each EC2 instance in the input document (your Terraform HCL file – or specifically, a JSON representation of it that Regula transforms behind the scenes).

`not approved_amis[input.ami]` will evaluate to `true` if the current instance's `ami` attribute is **not** a member of the set.

Put together, the deny rule says "If `input.ami` is **not** in `approved_amis`, the Regula check should fail and return an error message."

We define the error message in this line:

```
msg = sprintf("%s is not an approved AMI ID", [input.ami])
```

This uses the built-in Rego function sprintf to print out the `ami` of the currently evaluated resource.

And that's our custom rule! Here it is in full. Now we're going to take Regula for a spin!

Running the custom rule with Regula

Now, let's test the rule out on a simple Terraform file. The Terraform HCL in our GitHub gist defines two EC2 instances: one "good" instance with the approved AMI ID `ami-09e67e426f25ce0d7` and one "bad" instance with the not-at-all-suspicious AMI ID `ami-totallylegitamiid`.

Run Regula

In your terminal, `cd` into the directory containing the code files and run the following command:

```
regula run ami.tf --include approved_ami.rego --user-only
```

This command runs our custom rule against the Terraform file, and the `--user-only` flag says to only apply the custom rule; for the purposes of this blog post, we're excluding Regula's library of built-in rules. (It's a good practice to use the built-in rules, though, which are included by default when you execute `regula run`.)

We see the following output:

```
CUSTOM_0002: AWS EC2 instances must use approved AMIs [High]

  [1]: aws_instance.bad
       in ami.tf:13:1
       ami-totallylegitamiid is not an approved AMI ID

Found one problem.
```

As you can see, our Terraform file failed the rule with the message `ami-totallylegitamiid is not an approved AMI ID`. This is great, because it means our custom rule worked! Regula showed us that the `aws_instance.bad` instance (line 13, column 1) did not have an approved AMI ID.

See a more detailed report

For more details, let's look at the full report, which is formatted as JSON:

```
regula run ami.tf --include approved_ami.rego --user-only --format json
```

Here we can see the full output, including *all* rule results. Below, note how `aws_instance.bad` has a FAIL result again, whereas `aws_instance.good` has a PASS rule result:


```
{
  "rule_results": [
    {
      "controls": [
        "CORPORATE-POLICY_1.2"
      ],
      "filepath": "ami.tf",
      "input_type": "tf",
      "provider": "aws",
      "resource_id": "aws_instance.bad",
      "resource_type": "aws_instance",
      "rule_description": "Per company policy, EC2 instances may only use AMI IDs f
      "rule_id": "CUSTOM_0002",
      "rule_message": "ami-totallylegitamiid is not an approved AMI ID",
      "rule_name": "approved_ami",
      "rule_result": "FAIL",
      "rule_severity": "High",
      "rule_summary": "AWS EC2 instances must use approved AMIs",
      "source_location": [
        {
          "path": "ami.tf",
          "line": 13,
          "column": 1
        }
      ]
    },
    {
      "controls": [
        "CORPORATE-POLICY_1.2"
      ],
      "filepath": "ami.tf",
      "input_type": "tf",
      "provider": "aws",
      "resource_id": "aws_instance.good",
      "resource_type": "aws_instance",
      "rule_description": "Per company policy, EC2 instances may only use AMI IDs f
      "rule_id": "CUSTOM_0002",
      "rule_message": "",
      "rule_name": "approved_ami",
      "rule_result": "PASS",
      "rule_severity": "High",
      "rule_summary": "AWS EC2 instances must use approved AMIs",
      "source_location": [
        {
          "path": "ami.tf",
          "line": 8,
          "column": 1
        }
      ]
    }
  ],
  "summary": {
    "filepaths": [
      "ami.tf"
    ],
    "rule_results": {
      "FAIL": 1,
      "PASS": 1,
      "WAIVED": 0
    },
    "severities": {
      "Critical": 0,
      "High": 1,
      "Informational": 0,
      "Low": 0,
      "Medium": 0,
      "Unknown": 0
    }
  }
}
```


You can also see the rest of the metadata we defined in the rule earlier.

Fix the Terraform

If you'd like to bring the Terraform into compliance, you can edit `ami.tf` to replace `ami-totallylegitamiid` with `ami-09e67e426f25ce0d7` and then run Regula again. We'll just use the default text format this time:

```
regula run ami.tf --include approved_ami.rego --user-only
```

And we see the following output:

```
No problems found.
```

Success! We've secured our Terraform IaC by using only approved AMIs, as proven by Regula.

Further reading

In this blog post, we wrote a custom rule in Rego, used Regula to check it against a Terraform HCL file, and brought the Terraform into compliance.

If you'd like to learn more about Regula, visit the Regula docs site. You might find the following resources useful:

- Writing Rules
- Example: Writing a Simple Rule
- Usage

Posted in: Cloud Security, IaC Security