Regula
Home
Getting Started
Usage
Rules List
Report Output
Configuring Regula
Regula and Fugue
Integrations
Custom Rule Development
Examples
Example: Waiving and Disabling Rules
Example: Writing a Simple Rule
Example: Debugging a Rule
Contributing
CHANGELOG
About

Example: Debugging a Rule

In this example, we'll demonstrate how to debug a simple custom rule using Regula's REPL (which stands for read-eval-print loop). The rule has an error in it -- let's find it and fix it!

The (broken) rule

Here's a custom rule we wrote that is *supposed* to check whether a Google storage bucket has object versioning enabled. If it does, Regula should return a `PASS` rule result; if it doesn't, Regula should return a `FAIL`.

```
# This rule is intentionally broken!
package rules.bucket_versioning

__rego__metadoc__ := {
  "id": "CUSTOM_0004",
  "title": "Google storage buckets should have versioning enabled",
  "description": "Object versioning protects data from being overwritten or uninte
  "custom": {
    "controls": {
      "CORPORATE-POLICY": [
        "CORPORATE-POLICY_1.4"
      ]
    },
    "severity": "Medium"
  }
}

default allow = false

resource_type = "google_storage_bucket"

allow {
  input.versioning.enabled == true
}
```

Test the broken rule

Here's the Terraform file we want to check:

```
resource "google_storage_bucket" "good" {
  project = "my-project"
  name          = "good-bucket"
  location      = "US"

  versioning {
    enabled = true
  }

  lifecycle_rule {
    condition {
      num_newer_versions = 10
    }
    action {
      type = "Delete"
    }
  }
}

resource "google_storage_bucket" "bad" {
  project = "my-project"
  name          = "bad-bucket"
  location      = "US"

  versioning {
    enabled = false
  }
}
```

As you can see, we have one "good" bucket with versioning enabled and one "bad" bucket with versioning disabled. We expect the "good" resource to return a `PASS` rule result and the "bad" resource to return a `FAIL`.

Let's see what happens when we run Regula on `bucket.tf`. We're going to use the `--include` flag to include the custom rule (`google_bucket_versioning.rego`) and the `--no-built-ins` flag to disable the library of built-in rules, since we only want to see results for our custom rule:

```
regula run bucket.tf --include google_bucket_versioning.rego --no-built-ins
```

We see this output:

```
CUSTOM_0004: Google storage buckets should have versioning enabled [Medium]

  [1]: google_storage_bucket.bad
       in bucket.tf:20:1

  [2]: google_storage_bucket.good
       in bucket.tf:1:1

Found 2 problems.
```

Uh oh! The "bad" bucket failed the check as expected, but so did the "good" bucket. Something must be wrong with our rule. 🤔

Let's fire up Regula's REPL and investigate!

Use the REPL

We'll start the REPL by loading the rule module (`google_bucket_versioning.rego`) and the input document (`bucket.tf`):

```
regula repl google_bucket_versioning.rego bucket.tf
```

Open the package

Now, let's specify the package of the rule we want to examine. (You can load multiple rule modules at once, so it's important to tell Regula which one you want to look at -- even if there's only one, as in this case.) The package name comes from the package declaration at the very beginning of our rule module, `rules.bucket_versioning`:

```
package rules.bucket_versioning
```

Import the test inputs

When you load an IaC file into Regula's REPL, Regula generates a Rego module containing JSON-formatted test inputs. We can use this test input to evaluate our rule. To do so, we have to **import** the input by specifying its package name.

To specify the package name for the desired input file (`bucket.tf`), take the filepath and convert separators to dots (`.`) and other punctuation to underscores (`_`). So, the package name becomes `bucket_tf`. (Learn more about test input package names here.)

Then, when we import the module, we prepend the package name with `data` in order to access the data inside of it. Here's the command we end up with:

```
import data.bucket_tf
```

Evaluate the `allow` rule

Let's take a quick look at our `allow` rule. Here's the Rego again, for reference:

```
allow {
  input.versioning.enabled == true
}
```

For some reason, it's not working as expected. Something must be wrong with the syntax. Let's test the allow rule, using the "good" bucket as input.

But before we run a command, let's talk about the test input `mock_resources`. Regula generates three types of test inputs from an IaC file:

- `mock_resources` is used as input for **simple rules**
- `mock_input` is used as input for **advanced rules**
- `mock_config` is used as input when checking configuration outside of resources, such as provider config

So the input type we're concerned about right now is `mock_resources`, because ours is a simple rule.

You can view the `mock_resources` in the REPL like so:

```
bucket_tf.mock_resources
```

Here's the output:

```
{
  "google_storage_bucket.bad": {
    "_filepath": "bucket.tf",
    "_provider": "google",
    "_tags": {},
    "_type": "google_storage_bucket",
    "id": "google_storage_bucket.bad",
    "location": "US",
    "name": "bad-bucket",
    "project": "my-project",
    "versioning": [
      {
        "enabled": false
      }
    ]
  },
  "google_storage_bucket.good": {
    "_filepath": "bucket.tf",
    "_provider": "google",
    "_tags": {},
    "_type": "google_storage_bucket",
    "id": "google_storage_bucket.good",
    "lifecycle_rule": [
      {
        "action": [
          {
            "type": "Delete"
          }
        ],
        "condition": [
          {
            "num_newer_versions": 10
          }
        ]
      }
    ],
    "location": "US",
    "name": "good-bucket",
    "project": "my-project",
    "versioning": [
      {
        "enabled": true
      }
    ]
  }
}
```

Simple rules operate on one resource at a time, so to evaluate the `allow` rule, we need to specify a single resource as the input. In this case, let's choose `"google_storage_bucket.good"` (the resource ID of the "good" bucket) from `bucket_tf.mock_resources`.

And that's how we end up with the command below:

```
allow with input as bucket_tf.mock_resources["google_storage_bucket.good"]
```

When we run that command in the REPL, we get this output:

```
false
```

This confirms our suspicions that something is wrong with our rule -- we expect `allow` to be `true` for the "good" bucket, but we got the opposite result.

Examine the input

Maybe we've specified the `input.versioning.enabled` property incorrectly. We can check by examining that property in the input:

```
bucket_tf.mock_resources["google_storage_bucket.good"].versioning.enabled
```

We get this output:

```
undefined
```

Now, we *know* we enabled versioning for this bucket. Why is it returning `undefined`? There's definitely something wrong with how we specified the field in Rego. Let's back up a bit and just check `versioning` instead of `versioning.enabled`:

```
bucket_tf.mock_resources["google_storage_bucket.good"].versioning
```

We see this output:

```
[
  {
    "enabled": true
  }
]
```

Aha! `versioning` is actually an **array**. In Rego, you can iterate through an array with the `_` operator, which is a wildcard variable. So instead of using `input.versioning.enabled`, we should use `input.versioning[_].enabled`. Let's test it out!

Evaluate an expression

To test our new logic, we'll enter the following command to evaluate the expression `input.versioning[_].enabled == true` with our "good" bucket as the input again:

```
input.versioning[_].enabled == true with input as data.bucket_tf.mock_resources["g
```

And we see this output:

```
true
```

That confirms it! Now we can edit our Rego file to use the updated logic. Go ahead and exit the REPL:

```
exit
```

Fix the Rego file

We've made our changes to the rule file `google_bucket_versioning.rego`, and it looks like this now:

```
# Fixed rule!
package rules.bucket_versioning

__rego__metadoc__ := {
  "id": "CUSTOM_0004",
  "title": "Google storage buckets should have versioning enabled",
  "description": "Object versioning protects data from being overwritten or uninte
  "custom": {
    "controls": {
      "CORPORATE-POLICY": [
        "CORPORATE-POLICY_1.4"
      ]
    },
    "severity": "Medium"
  }
}

default allow = false

resource_type = "google_storage_bucket"

allow {
  input.versioning[_].enabled == true
}
```

As you can see, we've updated the `allow` logic to use `input.versioning[_].enabled` rather than `input.versioning.enabled`.

Test the fixed rule

Since we've updated our rule file now, we can run the same `regula run` command we used earlier:

```
regula run bucket.tf --include google_bucket_versioning.rego --no-built-ins
```

And we see this output:

```
CUSTOM_0004: Google storage buckets should have versioning enabled [Medium]

  [1]: google_storage_bucket.bad
       in bucket.tf:20:1

Found one problem.
```

Hooray! Our rule works as intended. The "bad" bucket failed, and the "good" bucket passed. Time to celebrate! 🎉

What's next?

Now that you've successfully debugged a simple custom rule, why not read up on test inputs or writing tests? Or, continue onward to learn how to contribute your rules.