terraform-validator

Sep 01, 2020

Getting started

CHAPTER 1

Install

Prerequisite: install Go 1.11+.

1.1 Get the last version from releases

You can download from here the binary. Move it into a directory in your `$PATH` to use it. For example:

```
version_latest=$(curl -s https://api.github.com/repos/thazelart/terraform-validator/
↪releases/latest | grep -oP '"tag_name": "\K(.*)(?=")')
wget "https://github.com/thazelart/terraform-validator/releases/download/${version_
↪latest}/terraform-validator_Linux_x86_64.tar.gz"
tar -zxvf terraform-validator_Linux_x86_64.tar.gz
chmod +x terraform-validator
sudo mv terraform-validator /usr/local/bin
```

1.2 Install from code:

To add terraform-validator, clone this repository and then get : then you can build it:

```
go build
```

move it into a directory in your `$PATH` to use it. For example:

```
mv terraform-validator /usr/local/bin
```

CHAPTER 2

First steps

Show help information:

```
terraform-validator --help
```

Show terraform-validator version:

```
terraform-validator --version
```

Validate a terraform folder located in `./examples`:

```
terraform-validator ./examples
```

CHAPTER 3

Docker

You can also avoid installing *terraform-validator* in your laptop by running it from a docker container:

```
docker run --name terraform-validator \
           -v "$(pwd)":/data \
           thazelart/terraform-validator
```

CHAPTER 4

Introduction

terraform-validator provides some customizations via the `.terraform-validator.yaml` file. The defaults fit for most projects.

your configuration file can contains two parameters: `layers` and `current_layer`.

CHAPTER 5

layers

Layers is a map of layer. By default, layers contains one layer named `default` :

```yaml
layers:
  default:
    files:
      main.tf:
        mandatory: true
        authorized_blocks:
      variables.tf:
        mandatory: true
        authorized_blocks:
          - variable
      outputs.tf:
        mandatory: true
        authorized_blocks:
          - output
      providers.tf:
        mandatory: true
        authorized_blocks:
          - provider
      backend.tf:
        mandatory: true
        authorized_blocks:
          - terraform
      default:
        mandatory: false
        authorized_blocks:
          - resource
          - module
          - data
          - locals
    ensure_terraform_version: false
    ensure_providers_version: false
    ensure_variables_description: false
```

(continues on next page)

(continued from previous page)

```
33      ensure_outputs_description: false
34      block_pattern_name: "^[a-z0-9_]*$"
```

5.1 What is a layer ?

A layer is a set of parameter that define the complete configuration of terraform-validator.

5.2 files

files is a map of filename that contains what rules define each files.

- `mandatory`: set if the file is mandatory.
  Type: `boolean`
  Default: `false`
- `authorized_blocks`: is a list of authorized terraform block types.
  Type: `List`
  Default: `<empty>`
  available terraform block types:
  - variable
  - outputs
  - provider
  - terraform
  - resource
  - module
  - data
  - locals

If a match does not match exactly one of the files, the configuration will be taken from `default`.

```
# .terraform-validator.yaml
...
      # main.tf is mandatory with no block inside
      main.tf:
        mandatory: true
        authorized_blocks:
      # variables.tf is not mandatory with only variable blocks inside
      variables.tf:
        mandatory: true
        authorized_blocks:
          - variable
      # other files will match default config
      default:
        mandatory: false
        authorized_blocks:
          - resource
          - module
          - data
          - locals
...
```

5.3 ensure_terraform_version

Type: `boolean`
Default: `false`
Configure terraform-validator in order to ensure (or not) if the

terraform version has been set.

5.4 ensure_providers_version

Type: `boolean`
Default: `false`
Configure terraform-validator in order to ensure (or not) if the

providers versions has been set.

5.5 ensure_variables_description

Type: `boolean`
Default: `false`
configures terraform-validator to check whether or not the variable

blocks are described.

5.6 ensure_outputs_description

Type: `boolean`
Default: `false`
configures terraform-validator to check whether or not the output

blocks are described.

5.7 block_pattern_name

Type: `string`
Default: `^[a-z0-9_]+$`
Configure the pattern that should match each terraform resources.

CHAPTER 6

Current layer

Type: `string`
Default: `default`
The `current_layer` permit you to select the configuration layer you

want to use in the current folder.

```
# .terraform-validator.yaml
current_layer: my_config
```

CHAPTER 7

Recursivity

7.1 layers

Each time you define a new layer in your `.terraform-validator.yaml`, this is added to terraform-validator configuration for the current directory and its sub-directories.

for example, here is my root directory configuration file:

```
# .terraform-validator.yaml
layers:
  cust1:
    files:
      default:
        authorized_blocks:
          - variable
          - output
          - provider
          - terraform
          - module
          - data
          - locals
    ensure_providers_version: true
    ensure_terraform_version: true
    block_pattern_name: "^[a-z0-9-]*$"
```

Starting from the root directory and for all its sub-directories, `cust1` layer will be available.

If a sub-directory define another `cust1`, it will replace this configuration for this sub-directory and its own sub-directories.

7.2 current_layer

The default `current_layer` is by default the one choosen in the parent directory.

7.3 tips

During on journey through your stacks and terraform modules,

terraform-validator won't run test in folders that do not contain any `.tf` files BUT will read the configurations. | That way, you can easily configure a large number of folders at once. Let's take this example:

```
|-- modules
|   |-- .terraform-validator.yaml
|   `-- mod1
|       |-- README.md
|       |-- main.tf
|       `-- provider.tf
|   `-- mod2
|       |-- README.md
|       |-- main.tf
|       `-- provider.tf
```

The `.terraform-validator.yaml` file inside the `modules` folder define the configuration for both `mod1` and `mod2` !

CHAPTER 8

How to contribute

8.1 Proposing a contribution

If you'd like to contribute, start by searching through the issues and pull requests to see whether someone else has raised a similar idea or question.

If you don't see your idea listed, and you think it fits into the goals of this guide, do one of the following:

```
If your contribution is minor, such as a typo fix, open a pull request.
If your contribution is major, such as a new guide, start by opening an issue first.␣
↪That way, other people can weigh in on the discussion before you do any work.
```

8.2 Code reviews

All submissions, including submissions by project members, require review. We use GitHub pull requests for this purpose. Consult GitHub Help for more information on using pull requests.

CHAPTER 9

Code of conduct

9.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

9.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

9.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at [INSERT CONTACT METHOD]. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

9.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

9.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

9.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

9.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

9.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the project community.

9.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

CHAPTER 10

Thanks

Terraform-validator logo was designed by a friend of mine. He also

helps me with icons integrations and all design matters. | Thank you Alexis Normand for all !

This tool will help you ensure that a terraform folder answer to your norms and conventions rules. This can be really useful in several cases :

- You're a team that want to have a clean and maintainable code.
- You're a lonely developer that develop a lot of modules and you want to have a certain consistency between them.

Features:

make sure that the block names match a certain pattern.

make sure that the code is properly dispatched. To do this you can decide what type of block can contain each file (for example output blocks must be in `outputs.tf`).

ensure that mandatory `.tf` files are present.

ensure that the terraform version has been defined.

ensure that the providers' version has been defined.

make sure that the variables and/or outputs blocks have the description argument filled in.

layered terraform folders (test recursively).

> **Warning:** Terraform 0.12+ is supported only by the versions 2.0.0 and higher.