

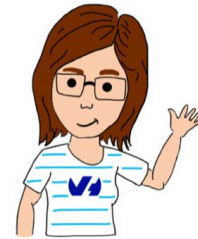


# DEVOPS.BARCELONA

## Let's dive into Terraform provider creation

Aurélie Vache - Horacio Gonzalez

2022-11-04



@AurelieVache



@LostInBrittany

# Aurélie Vache

@aurelievache

DevRel at  OVHcloud

Google Developer Expert –  
Google Cloud

Conferences organizer

Duchess France Leader & Mentor

Tech articles writer & sketchnoter

Gophers lover

... & Retrogaming ♥

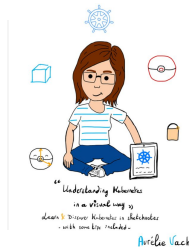
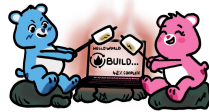
<https://www.youtube.com/c/AurelieVache>



@AurelieVache



@LostInBrittany



Les Productions de MOA

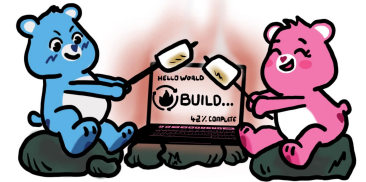


# Horacio Gonzalez



@LostInBrittany

Spaniard lost in Brittany,  
developer, dreamer and  
all-around geek



# OVHcloud: A global leader



**Web Cloud & Telcom**



**Private Cloud**



**Public Cloud**



**Storage**



**Network & Security**



**30 Data Centers**  
in 12 locations



**34 Points of Presence**  
on a 20 TBPS Bandwidth Network



**2200 Employees**  
worldwide



**115K Private Cloud**  
VMS running



**300K Public Cloud**  
instances running



**380K Physical Servers**  
running in our data centers



**1 Million+ Servers**  
produced since 1999



**1.5 Million Customers**  
across 132 countries



**3.8 Million Websites**  
hosting



**1.5 Billion Euros Invested**  
since 2016



**P.U.E. 1.09**  
Energy efficiency indicator

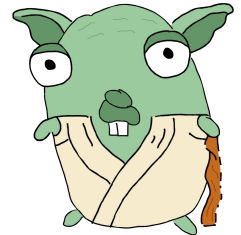
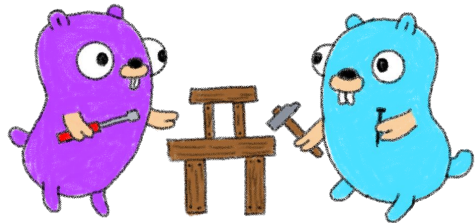


**20+ Years in Business**  
Disrupting since 1999



# Warning

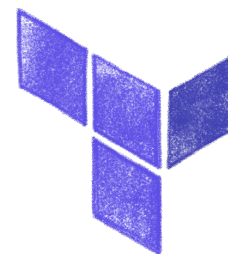
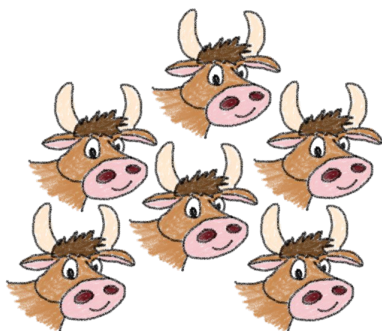
Gophers, gophers everywhere!





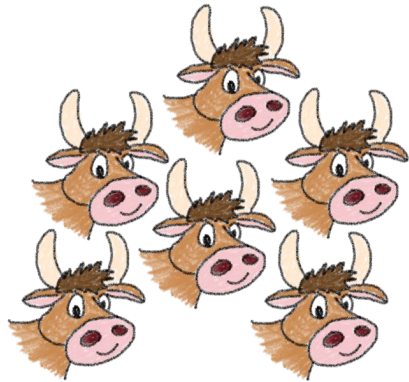
# Terraform

De facto standard for IaC



HashiCorp  
**Terraform**

# Infrastructure as Code (IaC)



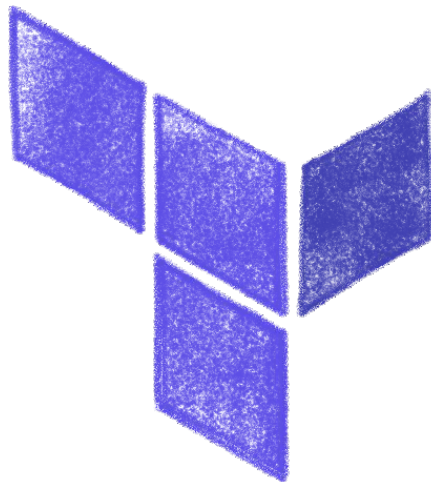
## Types of IaC

- Imperative
- Declarative
- Environment Aware

# Terraform becoming the de facto standard



ANSIBLE



HashiCorp

Terraform







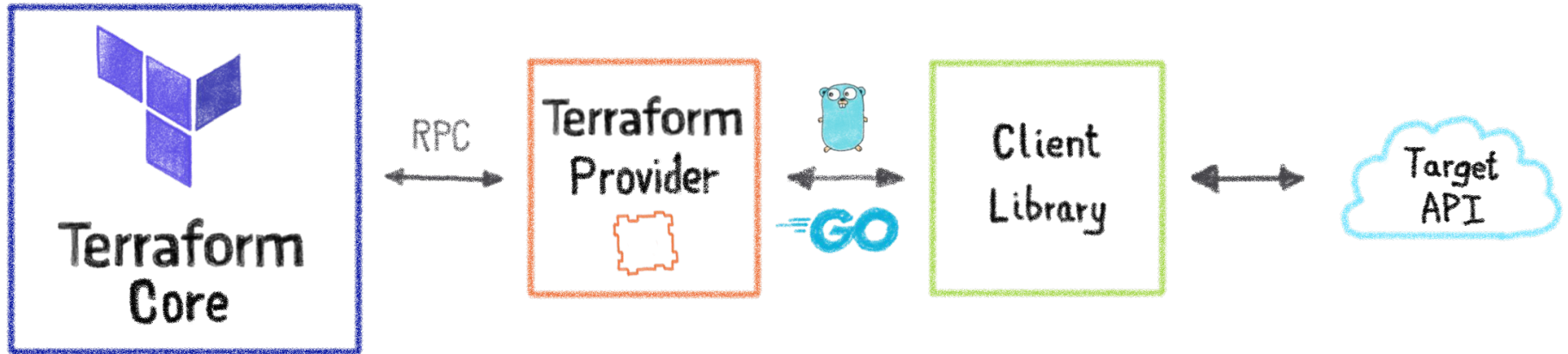
The screenshot shows the HashiCorp Terraform website. At the top left is the HashiCorp Terraform logo, and at the top right is a 'Menu' dropdown. The main heading is 'Automate Infrastructure on Any Cloud', followed by the subtext 'Provision, change, and version resources on any environment.' Below this are two links: 'View tutorials →' and 'View documentation →'. At the bottom, there are two main sections: 'Open Source' with the text 'Self-managed | always free' and a 'Download' button, and 'Terraform Cloud' with the text 'Managed Terraform' and a 'Try Terraform Cloud' button.

## Terraform

- Build
- Modify
- Version

your infrastructure

# Modular architecture: providers



# Configuration packages: modules



Modules :  
Collection of  
configuration files



# Terraform registry

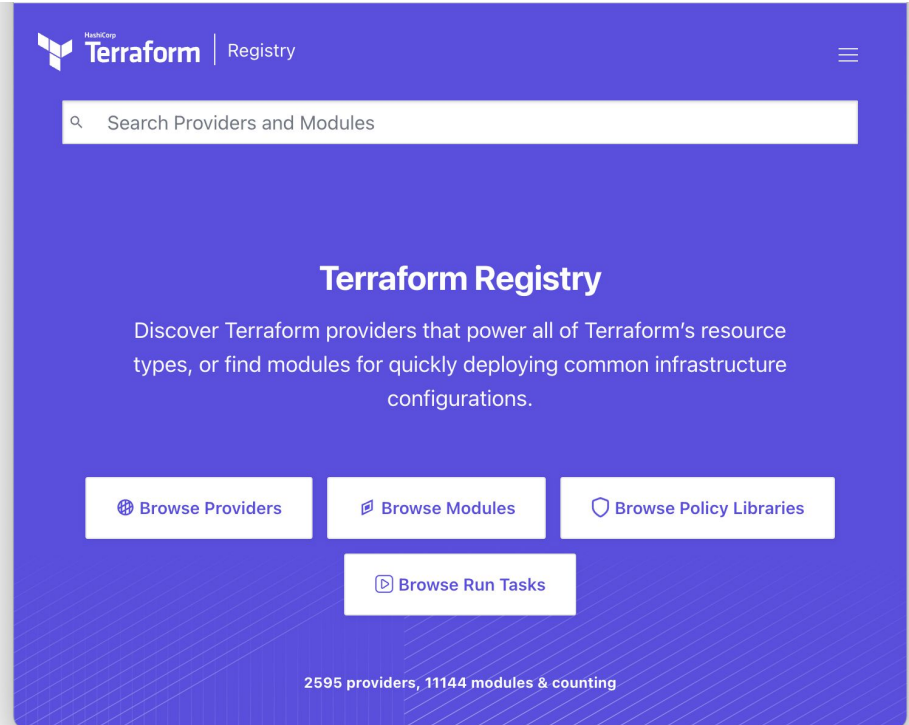


# Terraform Registry

Providers

&

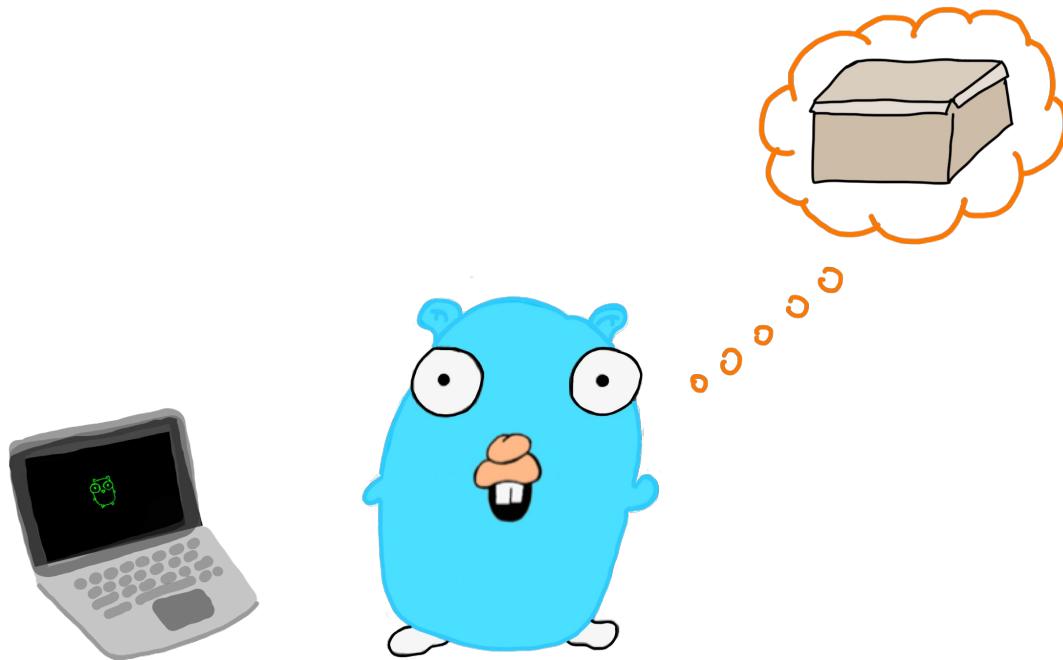
Modules





# Writing Terraform providers

## Defining new Terraform resources

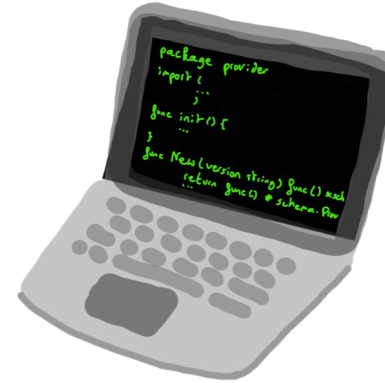
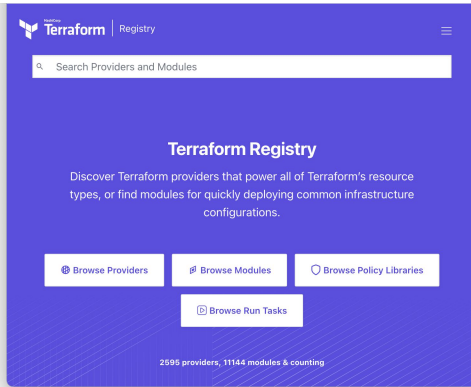


# Provider SDK



<https://developer.hashicorp.com/terraform/plugin/sdkv2>

# Installing Terraform providers



Or

```
$ terraform init

Initializing provider plugins...
- Finding terraform.local/local/myprovider versions matching
"0.0.1"...
- Installing terraform.local/local/myprovider v0.0.1...
```

# Installing providers from registry



```
$ vi provider.tf

terraform {
  required_providers {
    thenamespace = {
      source = "thenamespace/myprovider"
    }
  }
}
```

If your provider is on the official registry at

<https://registry.terraform.io/providers/thenamespace/myprovider>



# Installing providers locally



```
$ go build -o terraform-provider-myprovider
```

```
$ mkdir -p
```

```
~/terraform.d/plugins/terraform.local/local/myprovider/0.0.1/darwin_amd64
```

```
$ mv terraform-provider-myprovider
```

```
~/terraform.d/plugins/terraform.local/local/myprovider/0.0.1/darwin_amd64
```

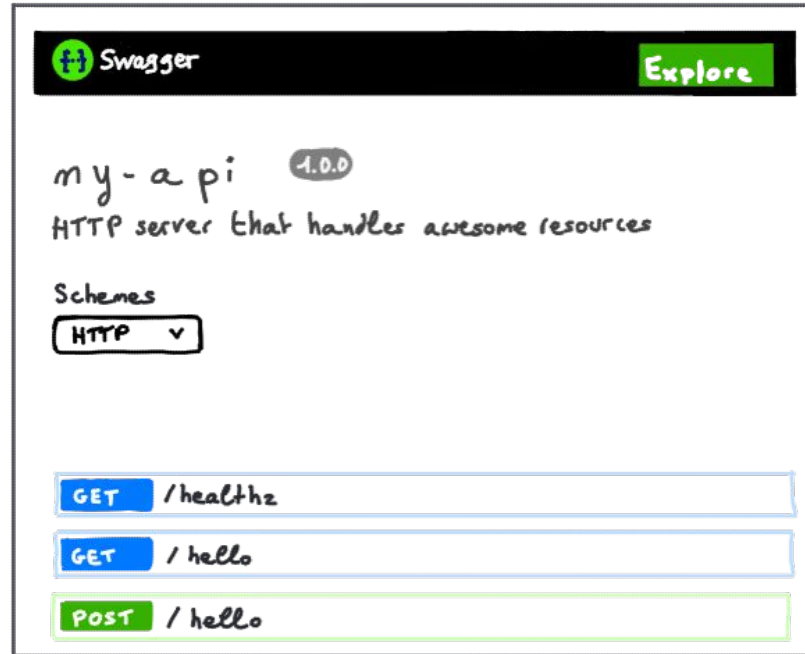
# Installing providers locally



```
$ vi provider.tf

terraform {
  required_providers {
    thenamespace = {
      source  = "terraform.local/local/myprovider"
      version = "0.0.1"
    }
  }
}
}
```

# Do I need a Terraform provider?

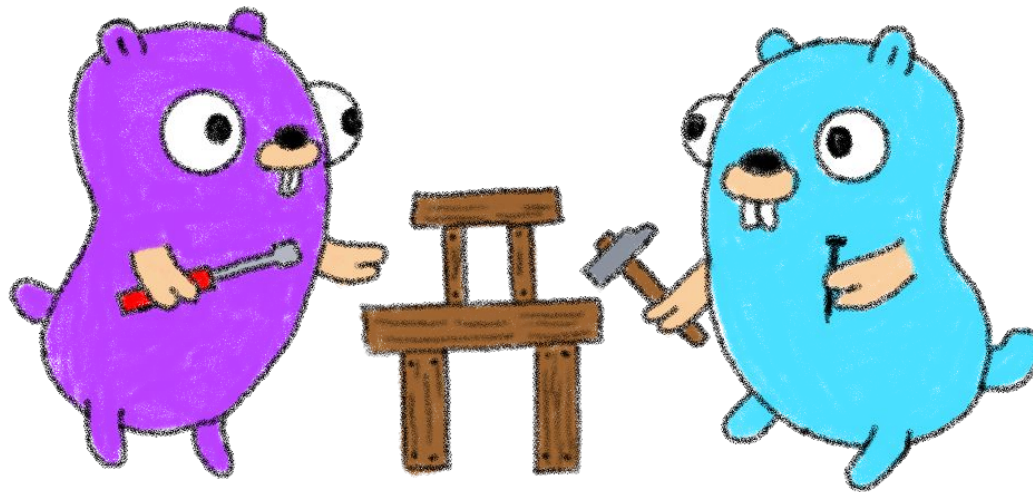


If you have an API,  
you should have a Terraform provider



# Let's create a provider!

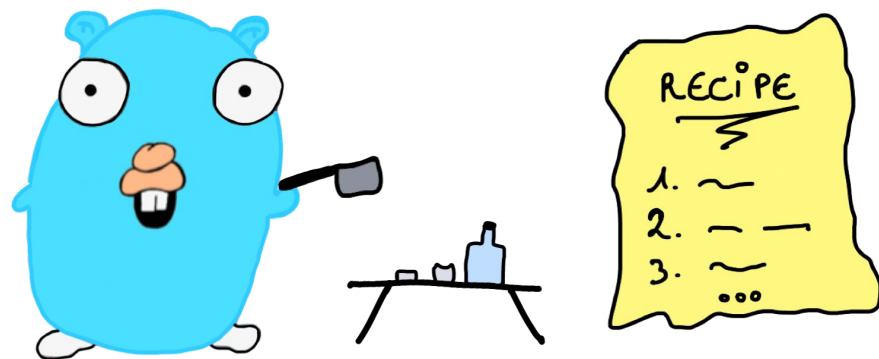
Step by step



# What do we want?



- Handle cute Gophers
- In a simple and easy Terraform provider
- In **Go**, because providers are made in Go 😁





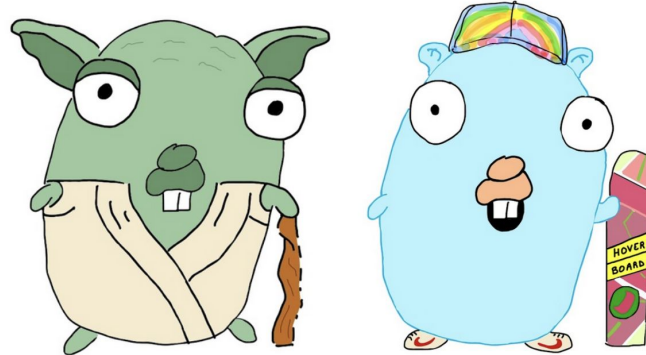
# Everything begins with an API



## gophers-api

This simple API handle a list of Gophers. It allows to:

- list the existing Gophers
- display the information about a Gopher
- create a new Gopher
- delete a Gopher
- update the path and the URL of a Gopher



<https://github.com/scraly/gophers-api>



# Everything begins with an API

Swagger  
Supported by SMARTBEAR

Explore

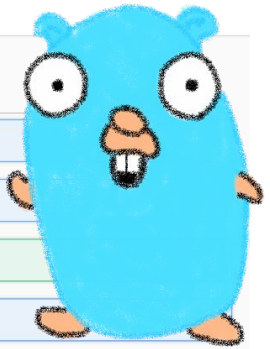
## gophers-api <sup>0.1.0</sup>

HTTP server that handle cute Gophers.

Schemes  
HTTP

### default

- GET /healthz
- GET /gophers
- POST /gopher Add a new Gopher
- GET /gopher
- DELETE /gopher
- PUT /gopher



# For the demos we will use Gitpod



# Gitpod



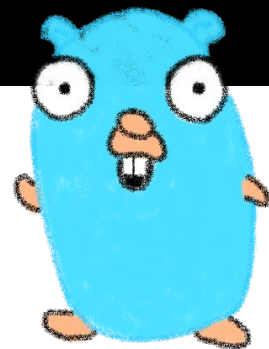
Automated, ephemeral developer environments  
in the web





# Everything begins with an API

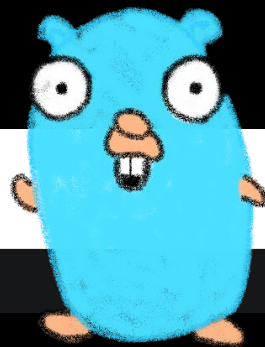
```
$ task swagger.serve
task: [swagger.serve] swagger serve -F swagger ./pkg/swagger/swagger.yml
--no-open
2022/10/31 20:16:51 serving docs at http://localhost:38457/docs
```





# Everything begins with an API

```
$ task run
task: [run] GOFLAGS=-mod=mod go run internal/main.go
2022/10/30 20:22:05 Serving gophers API at http://[::]:8080
```

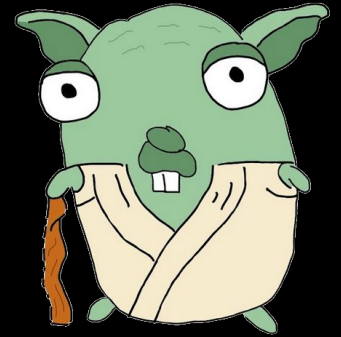


```
$ curl localhost:8080/gophers
[{"name": "5th-element", "displayname": "5th
Element.png", "url": "https://raw.githubusercontent.com/scraly/gophers/main/5th-ele
ment.png"}]
```



# Everything begins with an API

```
$ curl -X POST localhost:8080/gopher -H "Content-Type: application/json" -d \  
'{"name": "yoda-gopher", "displayname": "Yodada  
Gopher", "url": "https://raw.githubusercontent.com/scraly/gophers/main/yoda-gopher.  
png"}'  
  
$ curl -X DELETE localhost:8080/gopher?name=5th-element  
  
$ curl -X PUT localhost:8080/gopher \  
  -H "Content-Type: application/json" -d \  
'{"name": "yoda-gopher", "displayname": "Yoda  
Gopher", "url": "https://raw.githubusercontent.com/scraly/gophers/main/yoda-gopher.  
png"}'
```

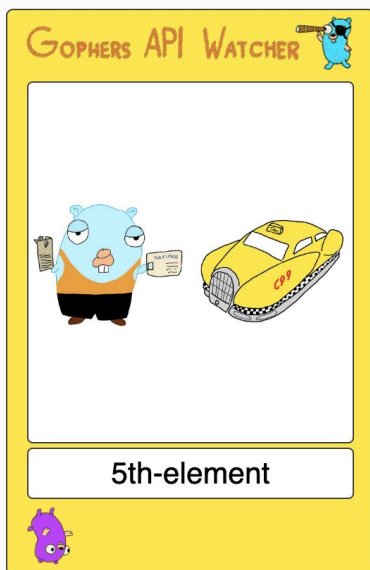
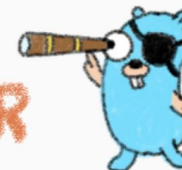




# Gophers deserve to be seen



## GOPHERS API WATCHER

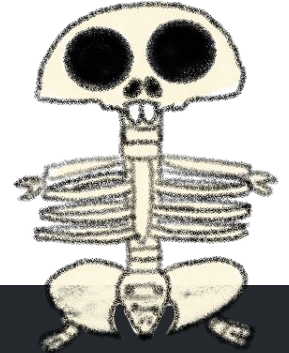


<https://github.com/LostInBrittany/gophers-api-watcher>



# Let's create our provider!

1. Create the skeleton of our provider thanks to scaffolding



Search or jump to... Pull requests Issues Marketplace Explore

hashicorp / terraform-provider-scaffolding Public template Watch

<> Code Issues 4 Pull requests Actions Security Insights

main 1 branch 0 tags Go to file Add file Code Use this template Gitpod

<https://github.com/hashicorp/terraform-provider-scaffolding>


# Let's create our provider!



## Create a new repository from terraform-provider-scaffolding

The new repository will start with the same files and folders as [hashicorp/terraform-provider-scaffolding](#).

Owner \*

 scraly ▾

Repository name \*

terraform-provider-myprovider ✓

Great repository name: terraform-provider-myprovider is available. Suggestion? How about [miniature-bassoon](#)?

Description (optional)



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

**Include all branches**

Copy all branches from hashicorp/terraform-provider-scaffolding and not just main.



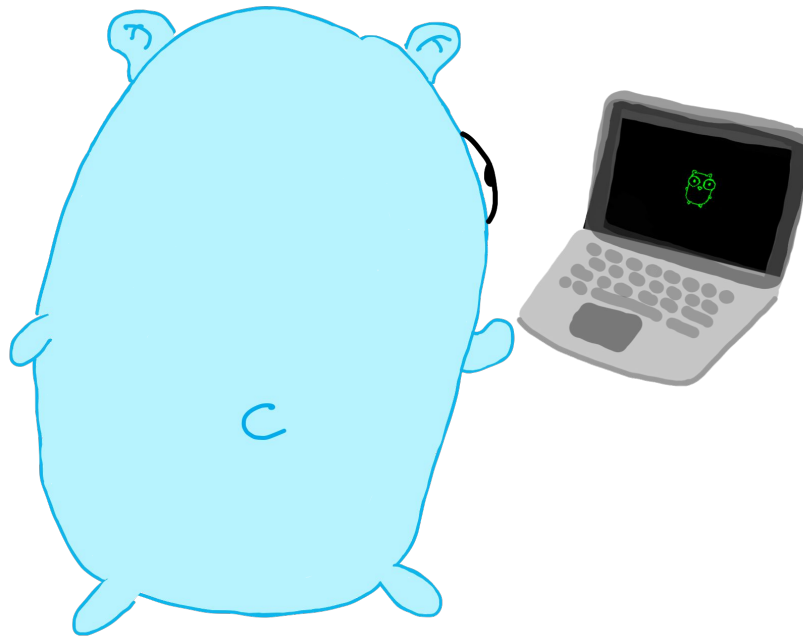
You are creating a public repository in your personal account.

Create repository from template



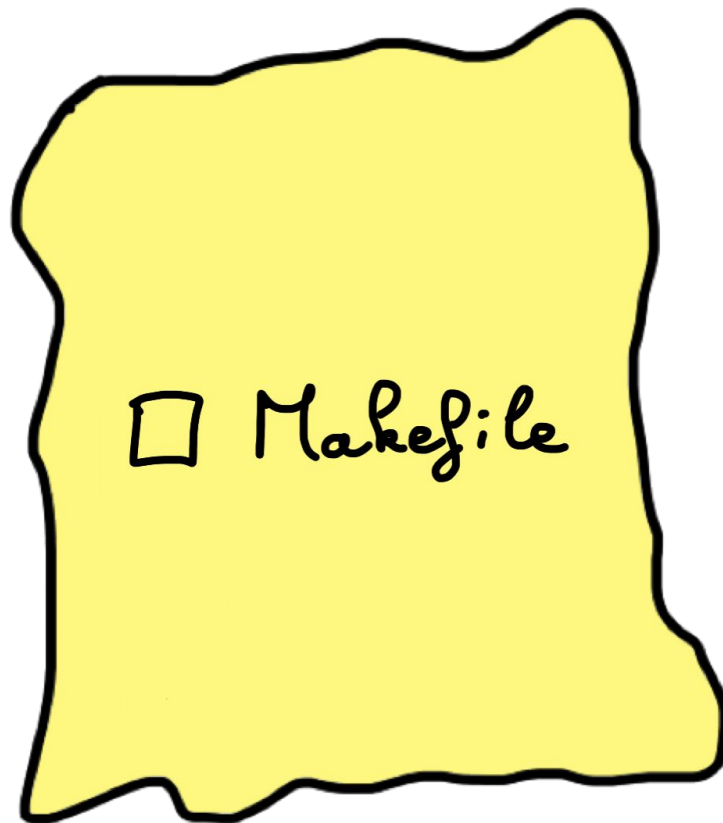
<https://github.com/scraly/terraform-provider-gophers>

# Demo time!





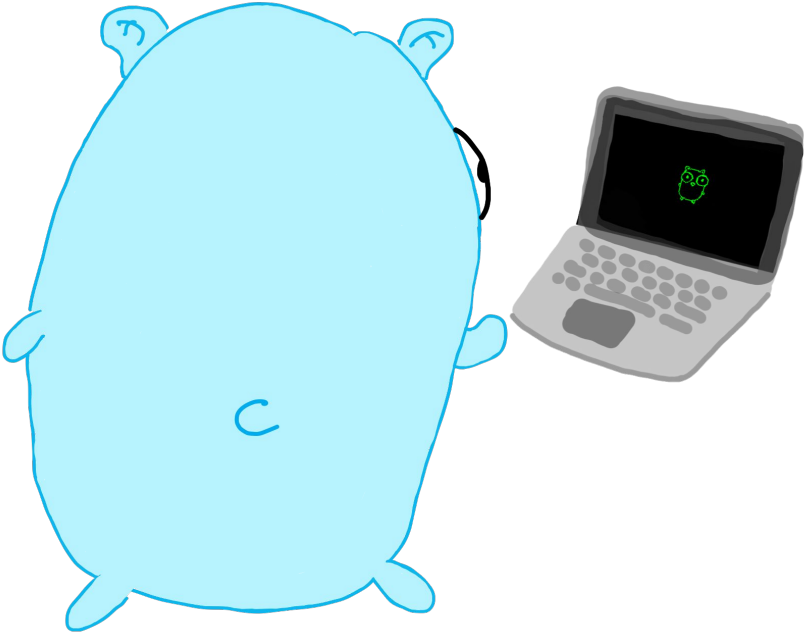
# Provider > Makefile







# Demo time!





# Customizing provider definition

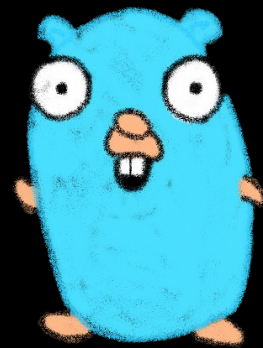
Customize  
provider  
definition



# Test it!

```
$ vi provider.tf
terraform {
  required_providers {
    gophers = {
      source = "terraform.local/local/gophers"
      version = "0.0.1"
    }
  }
}

provider "gophers" {
  endpoint = "http://myawesomeurl.com"
}
```



# Adding datasource: gophers



Add the gophers datasource



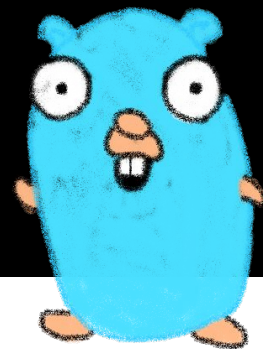


# Test it!

```
$ vi gophers_data.tf

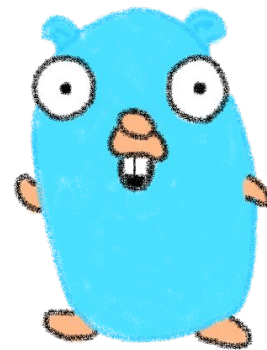
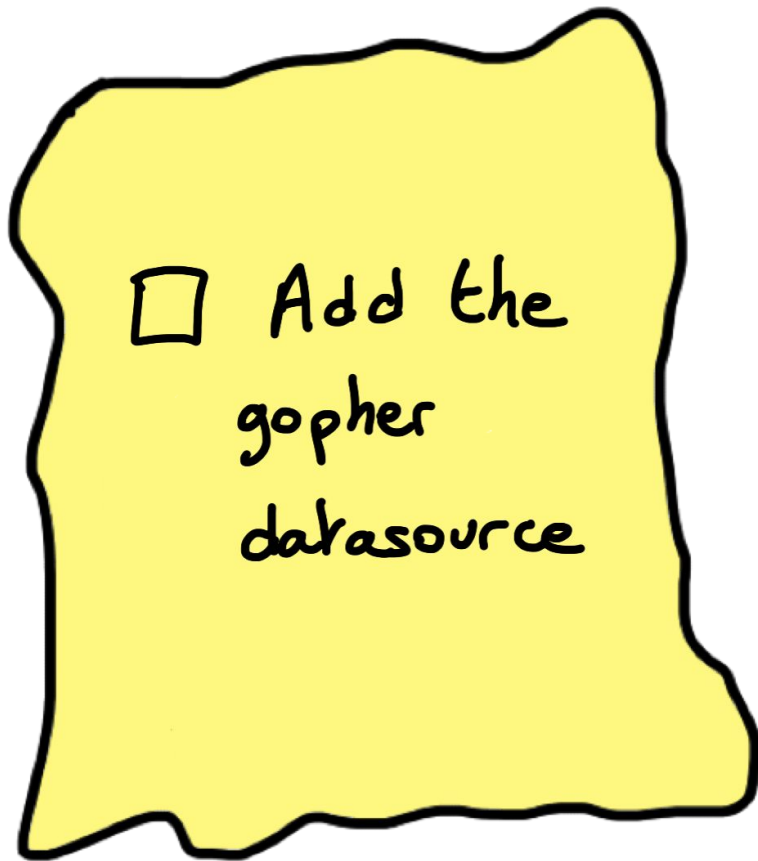
# List of available gophers
data "gophers" "my_gophers" {
}

output "return_gophers" {
  value = length(data.gophers.my_gophers.gophers) >= 1
}
```





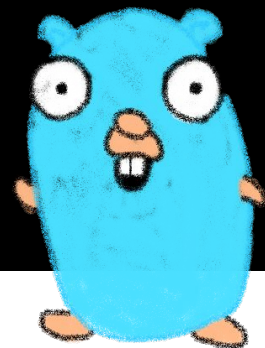
# Adding datasource: gopher



# Test it!

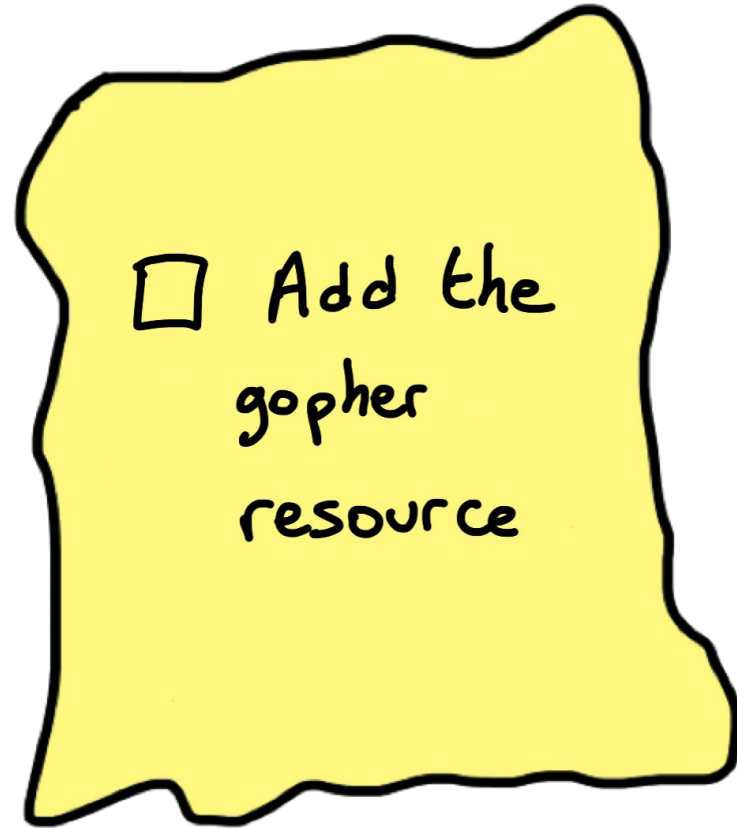


```
$ vi gopher_data.tf  
  
# Display information about a Gopher  
data "gophers_gopher" "moultipass" {  
  name = "5th-element"  
}
```





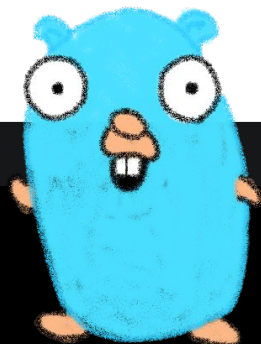
# Adding resource: gopher



**C**reate  
**R**ead  
**U**pdate  
**D**elete



# Test it!



```
$ vi gopher_resource.tf

resource "gophers_gopher" "x-files" {
  name          = "x-files"
  displayname   = "X Files"
  url           = "https://raw.githubusercontent.com/scraly/gophers/main/x-files.png"
}
```



# Testing the provider locally

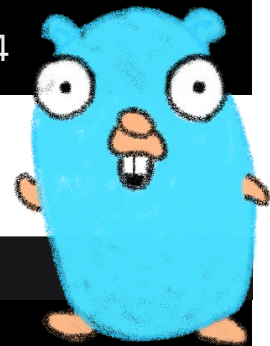
```
$ go build -o terraform-provider-gophers

$ mkdir -p
~/terraform.d/plugins/terraform.local/local/gophers/0.0.1/darwin_arm64

$ mv terraform-provider-gophers
~/terraform.d/plugins/terraform.local/local/gophers/0.0.1/darwin_arm64
```

Or

```
$ make install
```

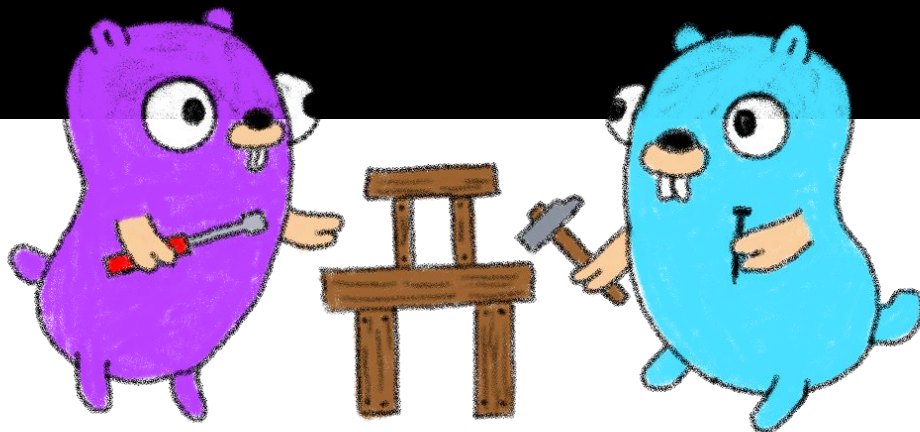




# Testing the provider locally

```
$ rm .terraform.lock.hcl && terraform init
```

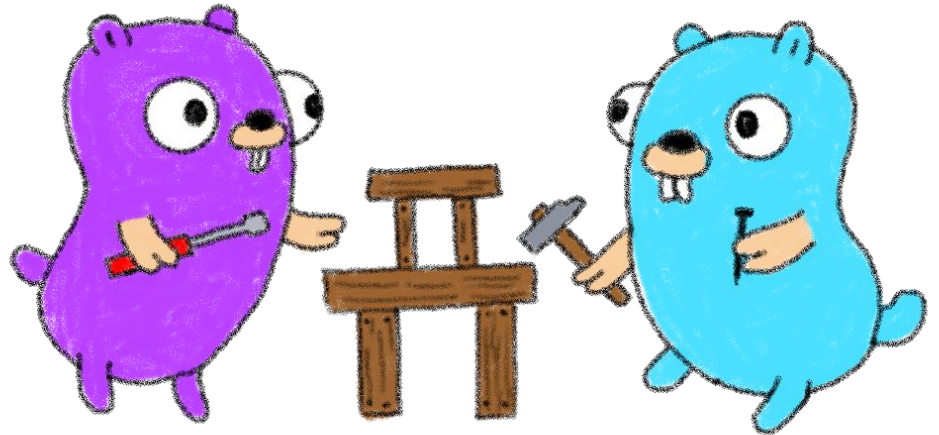
```
$ terraform apply
```





# Testing the provider locally

```
$ terraform destroy
```

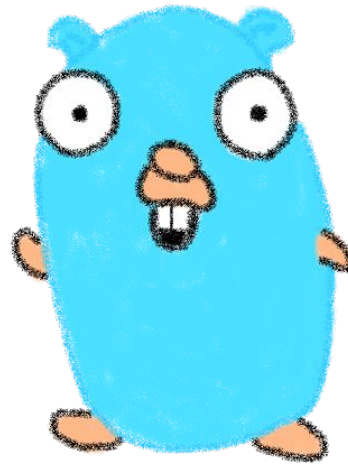






# OVHcloud Terraform Provider

To easily manage OVHcloud products



# OVHcloud Terraform Provider



ovh

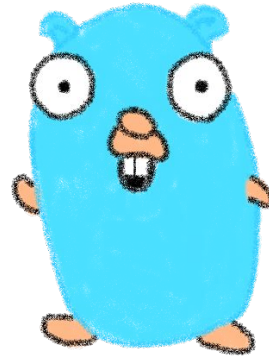
Partner by: [ovh](#)

Public Cloud

VERSION  
**0.22.0**

🕒 PUBLISHED  
**25 days ago**

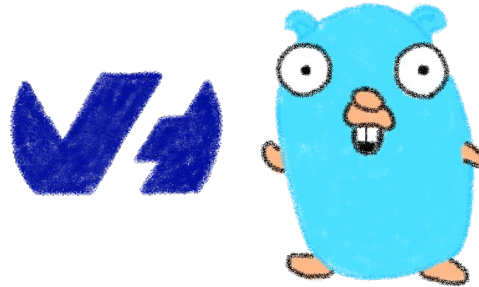
🔗 SOURCE CODE  
[ovh/terraform-provider-ovh](#)



Provider Downloads	All versions ▾
Downloads this week	9 902
Downloads this month	37 487
Downloads this year	385 075
Downloads over all time	708 757

<https://registry.terraform.io/providers/ovh/ovh/latest/docs>

# OVHcloud Terraform Provider



Contributors 54



+ 43 contributors

Releases 18

 **v0.22.0** Latest  
25 days ago

+ 17 releases

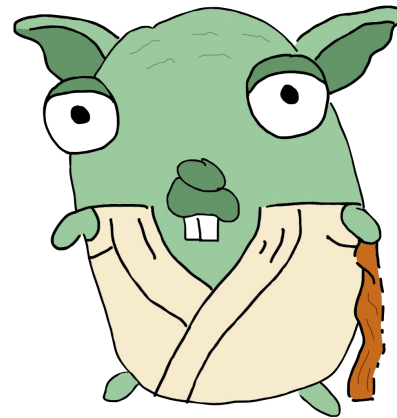
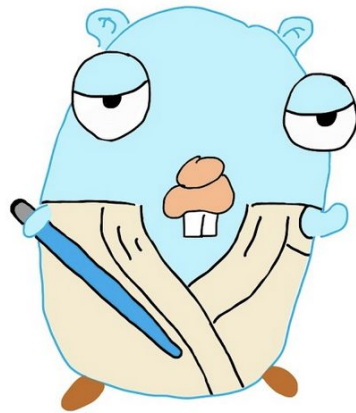
<https://github.com/ovh/terraform-provider-ovh>





# Best practices

But we have learnt with our providers



# Doc is not optional



```
$ tfplugindocs generate
```

Generate the doc of your provider.  
Based on the schema the provider  
exposes.



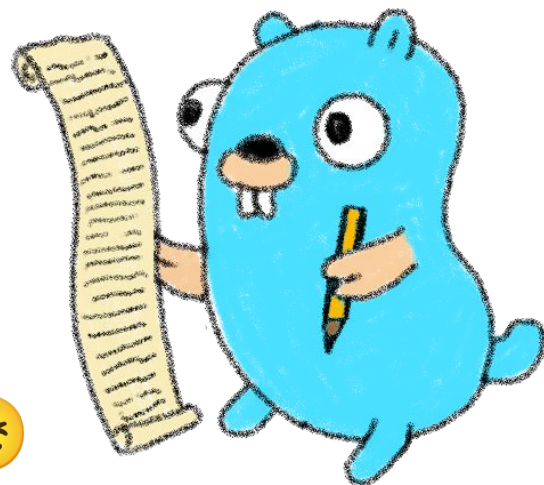
<https://github.com/hashicorp/terraform-plugin-docs>



# Write useful examples in your doc

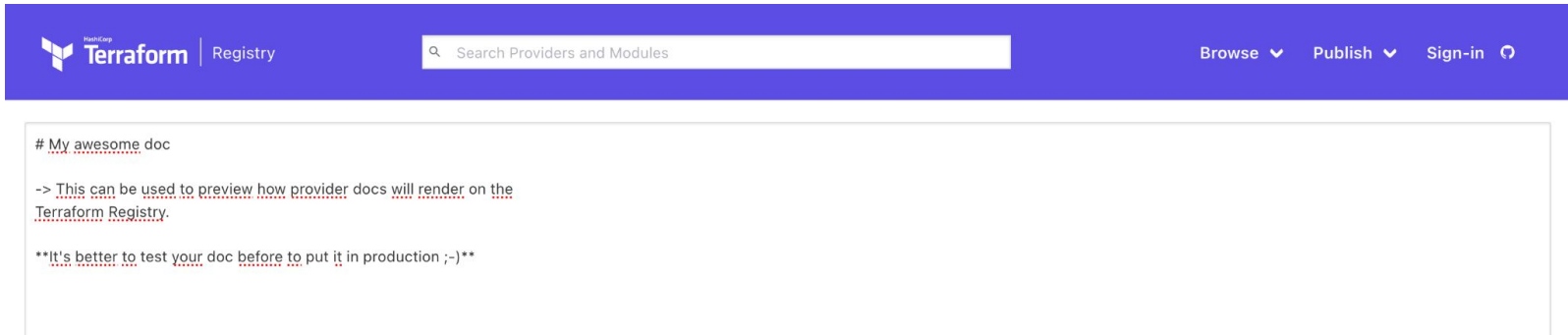
Examples in your documentation should be:

- Useful
- Up-to-date
- Working



Users will copy paste your examples! 😊

# And... test your doc!



PREVIEW DOCUMENTATION

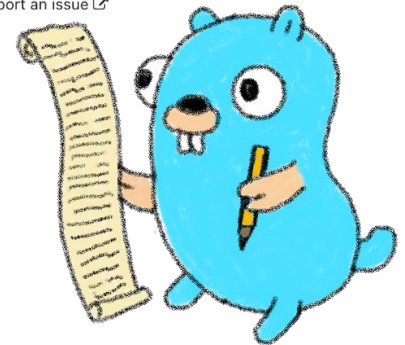
## My awesome doc

### Note

This can be used to preview how provider docs will render on the Terraform Registry.

It's better to test your doc before to put it in production ;-)

Report an issue [↗](#)



## Use the doc preview tool

<https://registry.terraform.io/tools/doc-preview>



# Acceptance tests



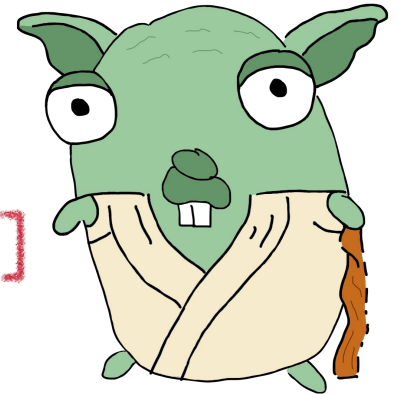
```
$ make testacc
```

```
$ make testacc TESTARGS="-run TestAccDataSourceGopher"
```

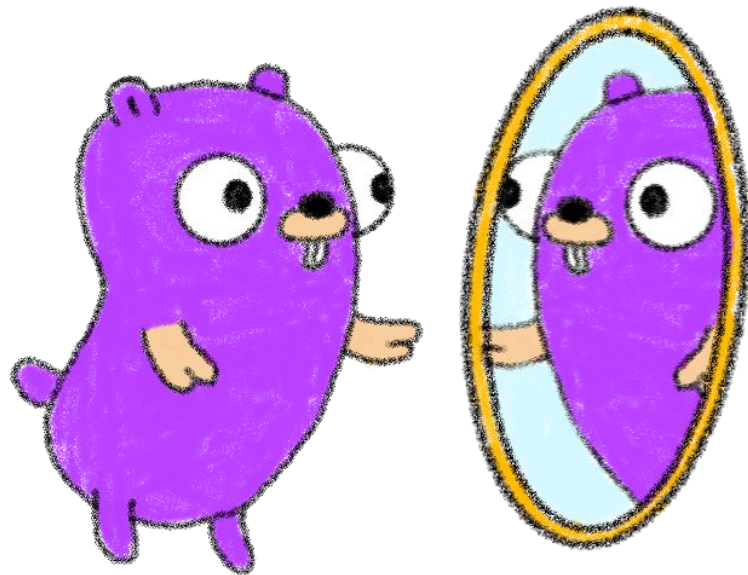


# Have the simplest JSON structures

```
{  
  "message":  
    ["The", "simplest", "JSON", "structures", "you", "use"]  
}
```



# Provider is a reflection of your API client



Think about API first design



# Use the logs for debugging

```
• • •  
$ TF_LOG=INFO terraform plan
```





# Set timeouts / retry



```
Timeouts: &schema.ResourceTimeout{  
    Create: schema.DefaultTimeout(20 * time.Minute),  
    Update: schema.DefaultTimeout(20 * time.Minute),  
    Delete: schema.DefaultTimeout(20 * time.Minute),  
},
```



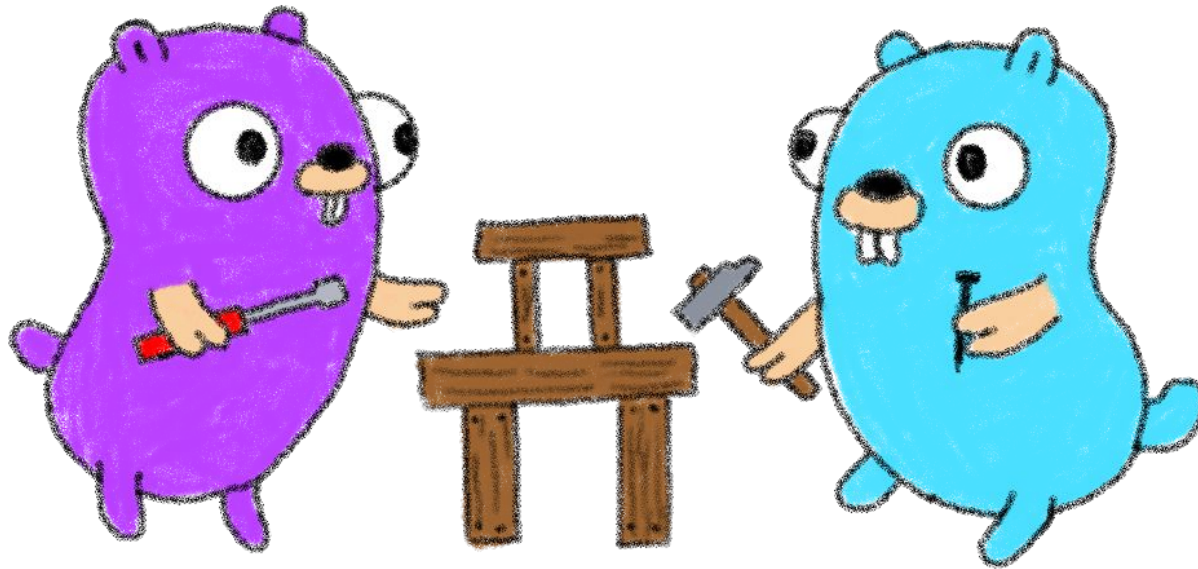
Timeout/retry par resource

# Read the code



See how other open source providers are written

# The “3 P” rule

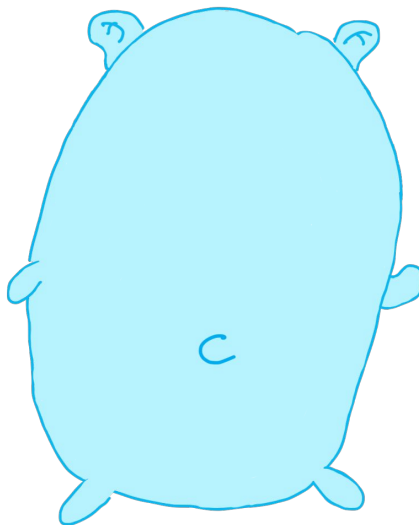


Practice, practice, practice



# One more thing...

## Or two or three



# A handy cheat sheet



## Terraform CLI Cheat Sheet

### About Terraform CLI

Terraform, a tool created by HashiCorp in 2014, written in Go, aims to build, change and version control your infrastructure. This tool has a powerful and very intuitive Command Line Interface.

### Installation

#### Install through curl

```
$ curl -O https://releases.hashicorp.com/terraform/0.11.10/terraform_0.11.10_linux_amd64.zip
$ sudo unzip terraform_0.11.10_linux_amd64.zip
-d /usr/local/bin/
$ rm terraform_0.11.10_linux_amd64.zip
```

#### OR install through tfenv: a Terraform version manager

First of all, download the tfenv binary and put it in your PATH.

```
$ git clone https://github.com/Zordrak/tfenv.git
- ./tfenv
$ echo 'export PATH="$HOME/.tfenv/bin:$PATH"'
>> $HOME/.bashrc
```

Then, you can install desired version of terraform:

```
$ tfenv install 0.11.10
```

### Usage

#### Show version

```
$ terraform --version
Terraform v0.11.10
```

#### Init Terraform

```
$ terraform init
```

It's the first command you need to execute. Unless, terraform plan, apply, destroy and import will not work. The command terraform init will install:

- terraform modules
- eventually a backend
- and provider(s) plugins

#### Init Terraform and don't ask any input

```
$ terraform init -input=false
```

#### Change backend configuration during the init

```
$ terraform init -backend-config=cfg/s3.dev.tf -reconfigure
```

-reconfigure is used in order to tell terraform to not copy the existing state to the new remote state location.

#### Get

This command is useful when you have defined some modules. Modules are vendored so when you edit them, you need to get again modules content.

```
$ terraform get -update=true
```

When you use modules, the first thing you'll have to do is to do a terraform get. This pulls modules into the .terraform directory. Once you do that, unless you do another terraform get -update=true, you've essentially vendored those modules.

#### Plan

The plan step check configuration to execute and write a plan to apply to target infrastructure provider.

```
$ terraform plan -out plan.out
```

It's an important feature of Terraform that allows a user to see which actions Terraform will perform prior to making any changes, increasing confidence that a change will have the desired effect once applied.

When you execute terraform plan command, terraform will scan all \*.tf files in your directory and create the plan.

#### Apply

Now you have the desired state so you can execute the plan.

```
$ terraform apply plan.out
```

**Good to know:** Since terraform v0.11+, in an interactive mode (non CI/CD/autonomous pipeline), you can just execute terraform apply command which will print out which actions TF will perform.

By generating the plan and applying it in the same command, Terraform can guarantee that the execution plan won't change, without needing to write it to disk. This reduces the risk of potentially-sensitive data being left behind, or accidentally checked into version control.

```
$ terraform apply
```

#### Apply and auto approve

```
$ terraform apply -auto-approve
```

#### Apply and define new variables value

```
$ terraform apply -auto-approve
-var tags-repository_url=${GIT_URL}
```

#### Apply only one module

```
$ terraform apply -target=module.s3
```

This -target option works with *terraform plan* too.

#### Destroy

```
$ terraform destroy
```

Delete all the resources!

A deletion plan can be created before:

```
$ terraform plan -destroy
```

-target option allow to destroy only one resource, for example a S3 bucket:

```
$ terraform destroy -target aws_s3_bucket.my_bucket
```

#### Debug

```
$ echo "aws_iam_user.notif.arn" | terraform console
arn:aws:iam::123456789:user/notif
```

#### Graph

```
$ terraform graph | dot -Tpng > graph.png
```

Visual dependency graph of terraform resources.

#### State

**How to tell to Terraform you moved a resource in a module?**

If you moved an existing resource in a module, you need to update the state:

```
$ terraform state mv aws_iam_role.role1 module.mymodule
```

**How to import existing resource in Terraform?**

If you have an existing resource in your infrastructure provider, you can import it in your Terraform state:

```
$ terraform import aws_iam_policy.elastic_post
arn:aws:iam::123456789:policy/elastic_post
```

#### Workspaces

To manage multiple distinct sets of infrastructure resources/environments.

Instead of create a directory for each environment to manage, we need to just create needed workspace and use them:

Create workspace



<https://github.com/scraly/terraform-cheat-sheet/>

# A little gift ;-)



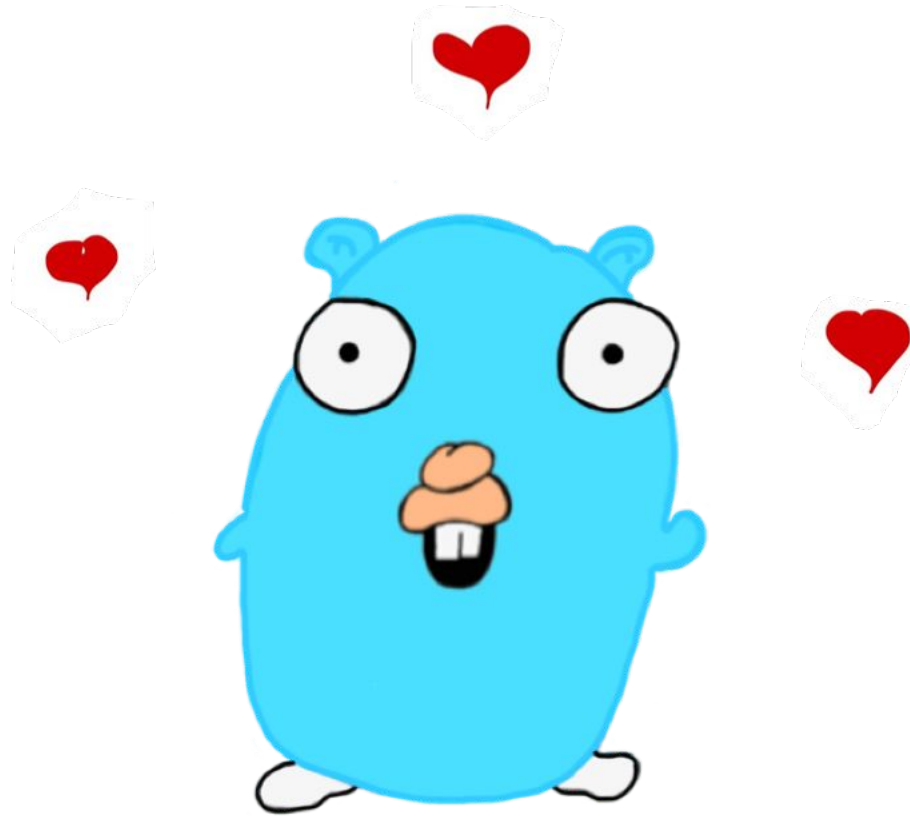
Would you like to try our  
Public Cloud services?

We are raffling  
3 vouchers worth 400€



 OVHcloud

# Thank you!



<https://bit.ly/tf-provider-bcn>