Terraform provider MVP for dummies

`whoami`

Christian Stankowic

- IT consultant and trainer at SVA
  - SUSE, Red Hat, Debian, etc.
  - Patch management
  - IaC, Automation, DevO(o)ps
  - Terraform, Packer, Vagrant,...
- FOCUS ON: Linux podcast 🎙️ 🇩🇪
- 📙 cstan.io

Agenda

Motivation

Motivation

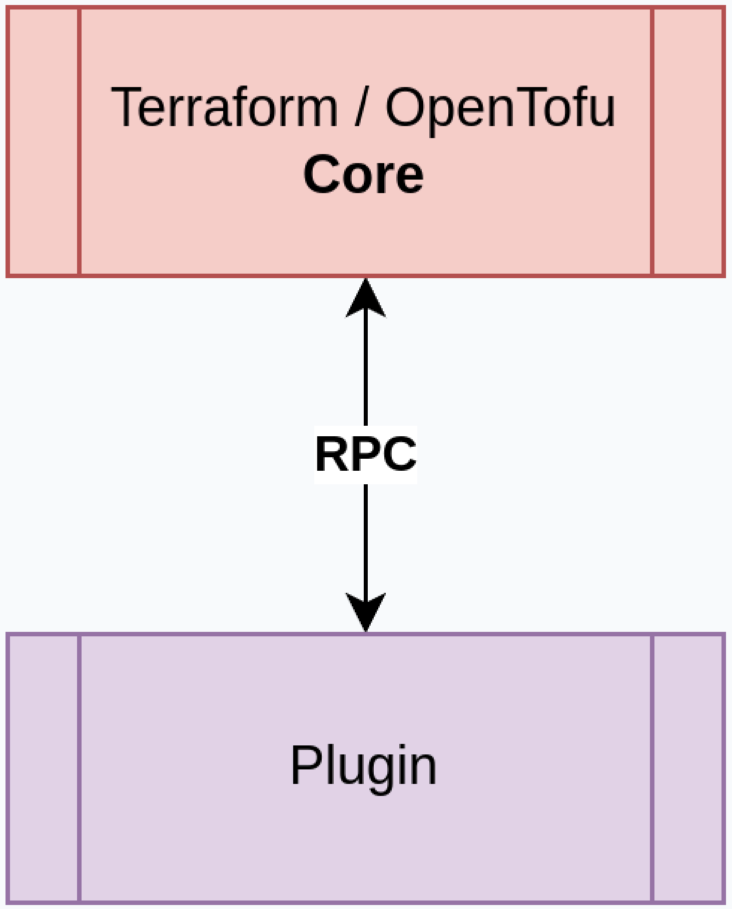
- Broad range of providers for Terraform and OpenTofu
  - currently ~4.780 providers
- Good coverage of mainstream and also niche integrations
- Some things are still missing
  - Uyuni
  - Forgejo
- Recent hackathon nerd-sniped us into writing providers

Extending Terraform

Architecture

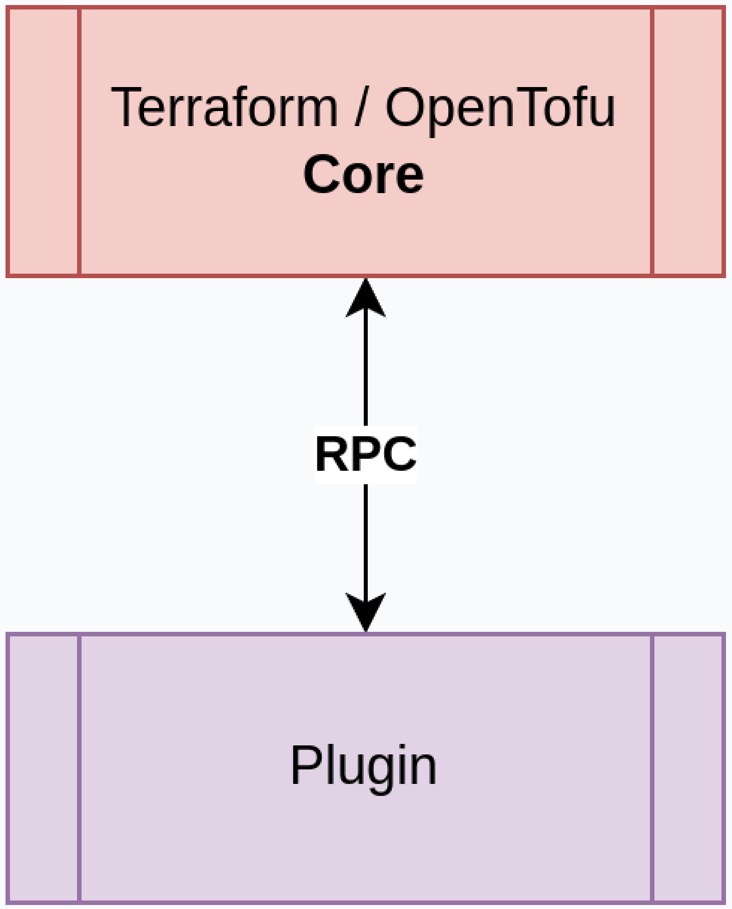
- Plugin-based architecture
  - Terraform Core + Terraform Plugins
- Core uses RPC to communicate with plugins
  - can contain provider or provisioner

- Providers interact with APIs
  - e.g. to create cloud workloads or SaaS objects
- Provisioners interact with provisioned resources in a pragmatic way
  - e.g. deploy a file using SSH
  - last resort when declarative model doesn't work



Architecture

- Plugins expose implementations specific to a particular service
  - e.g. creating users, groups,...
- started as separated process
- include required client libraries
- handle authentication and API communication



Discovery

- Project configuration defines required plugins and versions
  - Version pinning highly recommended
- When running `terraform init`, plugins are downloaded from a registry
  - e.g. registry.terraform.io, registry.opentofu.org or private registries
  - downloaded to `.terraform/providers`
- Writes lockfile with URL, version and hashes

Discovery

```
terraform {
  required_providers {
    hcloud = {
      source  = "hannahmontanacloud/hmcloud"
      version = "~> 1.33.7"
    }
  }
}
```

```
$ terraform init
$ ls .terraform/providers/registry.terraform.io/↩
hannahmontanacloud/hmcloud/1.33.7/linux_amd64/
CHANGELOG.md  LICENSE  terraform-provider-hmcloud_v1.33.7
```

Discovery

```
$ cat .terraform.lock.hcl
# This file is maintained automatically by "terraform init"
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hannahmontanacloud/hmcloud"
  version     = "1.33.7"
  constraints = "~> 1.33.7"
  hashes = [
    "h1:fa9fxdSV9DG+HDcXyRbcGfb6Dk94SBP3TamHb1yOYiI=",
    "zh:086cce10cb005f25f2ä3cv34c80f660ca1cc4b9bc09b",
    ...
    "zh:e0e9b2f6d5e28dbd01fa1ea88062d6223c514627e1e9",
  ]
}
```

Extending Terraform and OpenTofu

- Software and skills
  - Golang 1.21+
  - [Terraform Plugin Framework]
  - Terraform v1.8+ or OpenTofu
  - Git
  - an IDE of your choice

Learning Golang

Learning Golang

- I've been very late to party
  - and still am 🫠
  - used Python in the past
- interesting as it breaks with some old-fashioned paradigms
  - efficient compiler, multi-arch builds
  - package management
  - concurrency, memory safety
- developed at Google by persons behind UNIX, Plan 9 and Inferno

Golang 101: Syntax, Modules

Syntax looks like C, but lacks semicolons. Projects are initialized as modules:

```
$ go mod init example/hello
```

Example `hello.go` file:

```go
package main    // group all files of a directory

import "fmt"    // package for formatted I/O

func main() {
    fmt.Println("Hello CfgMgmtCamp 2025!")
}
```

Golang 101: Data type definitions

Short variable declarations are a real game changer:

```go
var number int
var number int = 1337
number := 1337

var a, b int = 13, 37
a, b := 13, 37
```

```go
const a = 13
const b = 37

const (
    a = 13
    b = 37
)
```

Golang also forces you to write proper code:

```
$ go build .
./hello.go:35:5: number declared and not used
```

Golang 101: Structs

Like C, Golang also supports structs (also known as composites) - a named set of various data types:

```go
type Talk struct {
    title, author   string
    slides          int
}
```

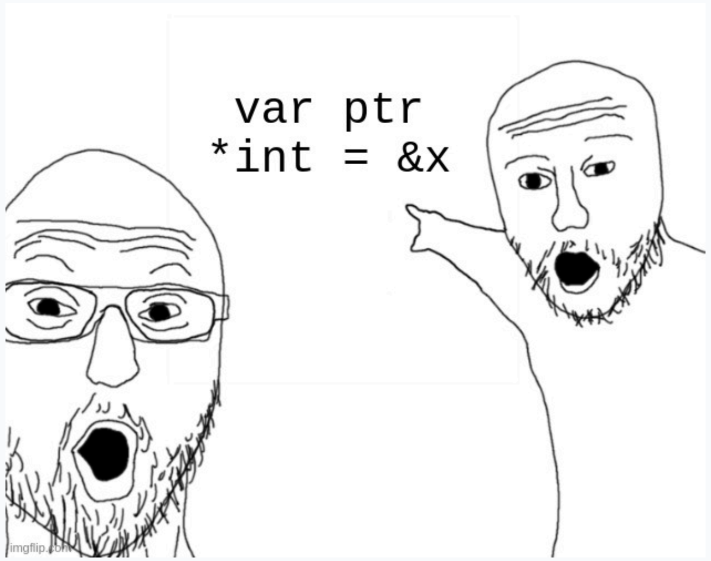
```go
func main() {
    talk := Talk{"Christian Stankowic",
      "TF Provider MVP for Dummies", 40}
}
```

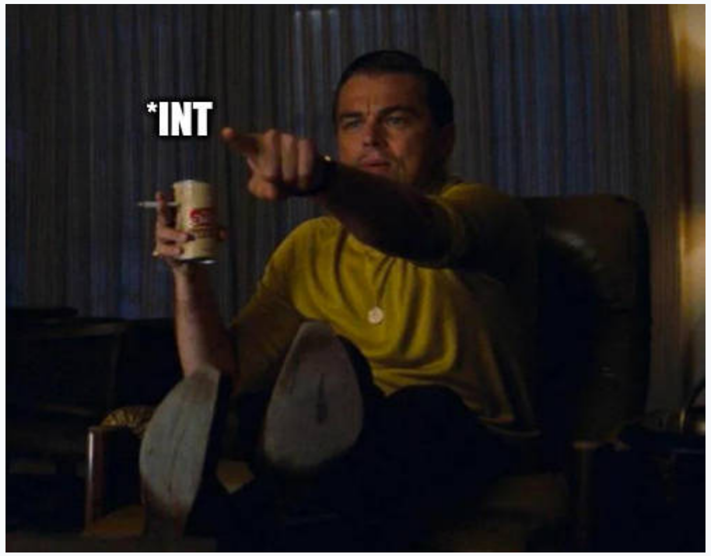
Golang 101: Pointers

- holds the memory address of a value
- used for referencing memory locations containing a variable's value
- significantly improves performance, e.g. when iterating over large amounts of data

```go
i := 1337    // int variable and value
p := &i      // p pointing to i

fmt.Println(p)  // printing address of i
fmt.Println(*p) // printing value of i
```





Golang 101: Functions

Functions have a name, parameter and return types:

```go
func Greeting(name string) string {
    // return a greeting including a name
    message := fmt.Sprintf("Ohai, %v.", name)
    return message
}
```

```go
fmt.Println(Greeting("Simone Giertz"))
```

- IDE addons, e.g. for VSCode, offer additional features (e.g. IntelliSense)
- `gopls` for other editors such as Vim, Helix or Emacs

Golang 101: Building

Building and running code is easy, even cross-plattform and architecture:

```
$ go run .
Hello CfgMgmtCamp 2025!
```

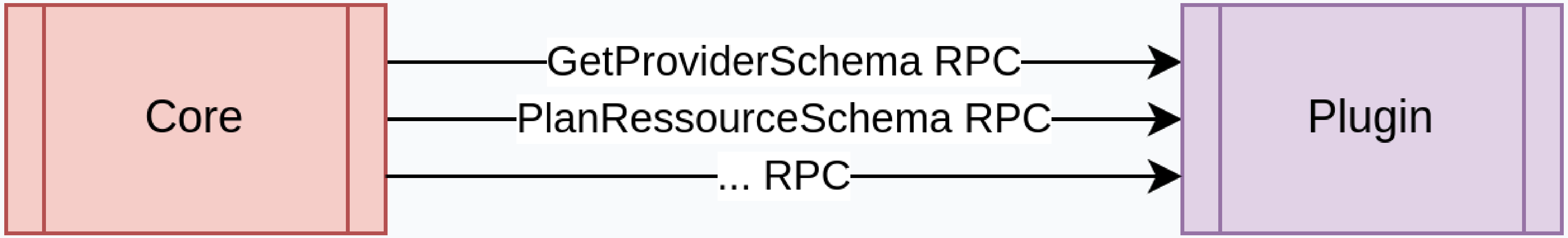
```
$ go build
$ ./hello
Hello CfgMgmtCamp 2025!
```

```
$ GOOS=windows GOARCH=arm64 go build .
$ file hello.exe
hello.exe: PE32+ executable (console) Aarch64,⮐
for MS Windows, 13 sections
```

Writing a provider

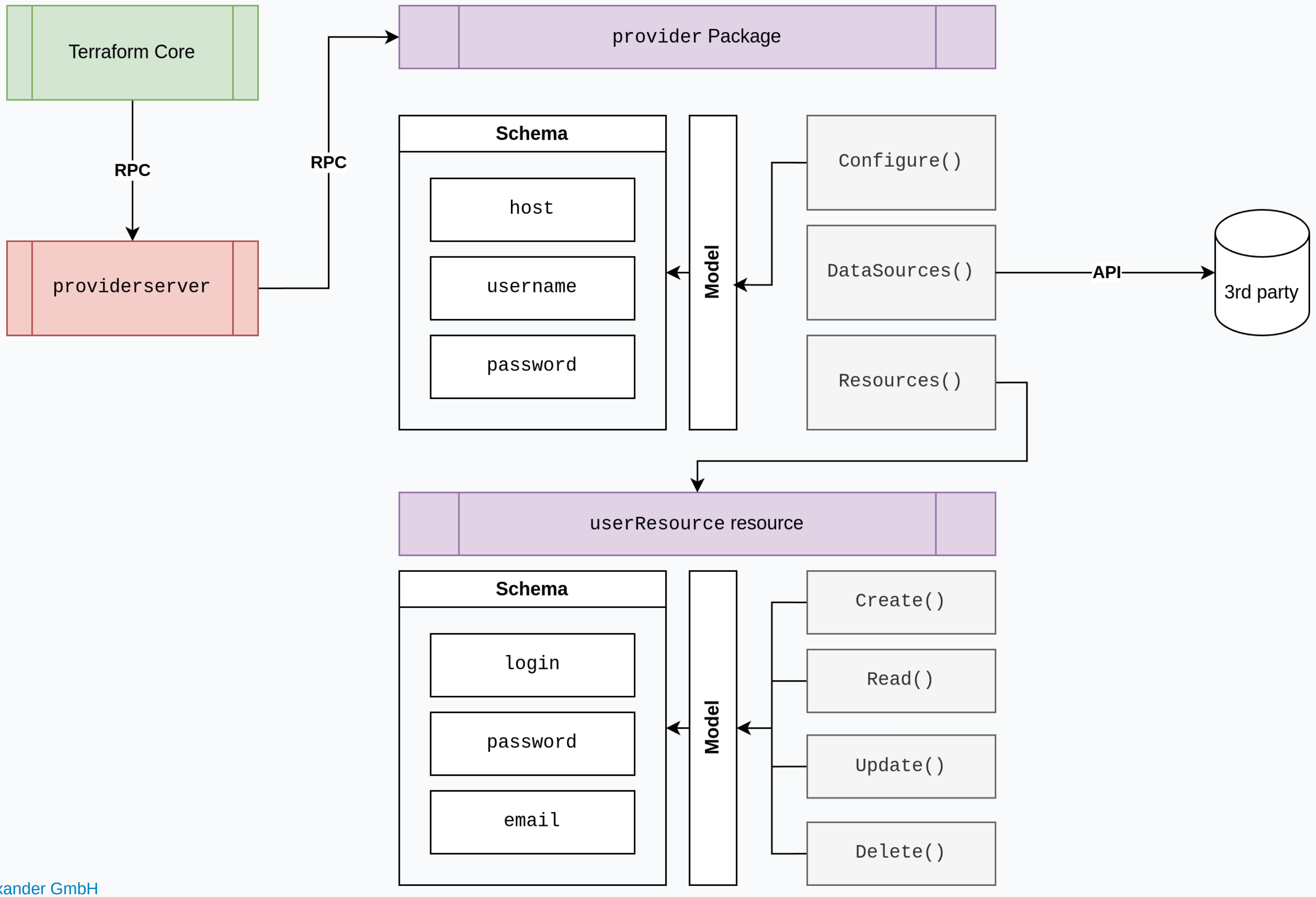
Writing a provider

- Terraform Plugin Framework assists with creating new plugins
  - also recommended for OpenTofu
  - superseded older SDKv2 (0.12 - 1.x)
  - adds new datatypes (e.g. `Tuple` and `Collection`)
- Framework implements Plugin Protocol offering various RPCs, e.g.
  - Metadata
  - Provider schema, functions and configuration
  - Plan and resource changes



Starting point

- Check whether there are already existing Go packages
  - no need to reinvent the wheel
  - keep complexity out of provider code
- Go package search helps
  - `forgejo-sdk` ✨
  - at least some Uyuni API code
- Take a step back
  - Try out with something easy
  - HashiCups application and Plugin Framework tutorial series
  - takes several hours, but it's highly recommended


Terraform Core
RPC
providerserver
RPC
provider Package
Schema
host
username
password
Model
Configure()
DataSources()
Resources()
API
3rd party
userResource resource
Schema
login
password
email
Model
Create()
Read()
Update()
Delete()

Needed steps

1. Implement provider type
    - e.g. data sources (reading already defined information) and resources (for target states)
2. Implement provider server (to handle RPC requests)
3. Implement provider schema, containing connection details
4. Implement provider model, linking to schema
5. Implement `Configure` functions
    - sanity checks, establishing connection
6. Implement all the data source types
    - schema, model and functional code
7. Implement `Create`, `Read`, `Update` and `Delete` functions
8. Implement Logging facilities

Building and testing

```
$ go build -o ~/go/bin/terraform-provider-uyuni
```

File `~/.terraformrc` needs to be altered for local builds and testing:

```
provider_installation {
  dev_overrides {
      "registry.terraform.io/svalabs/uyuni" = ↩
      "/home/christian/go/bin"
  }
  direct {}
}
```

Otherwise Terraform/OpenTofu tries to download a non-existing provider.

Example: Uyuni

- A very first MVP was created in a two-day Hackathon
- Creates `user` objects
  - `login`, `firstname`, `lastname`, `email`, `password`
- Pinned experimental Uyuni API package as dependency ¯\_(ツ)_/¯

Example: Uyuni

```
module terraform-provider-uyuni

go 1.22.7

require (
    github.com/hashicorp/terraform-plugin-framework v1.12.0
    ...
    github.com/uyuni-project/uyuni-tools⮐
    v0.0.0-20240925104919-172b63dcc7ae
)
```

"Fake git tag" based on timestamp and commit hash:

```
commit 172b63dcc7ae78c12cfca75c752844c229599cf4
Author: Ricardo Mestre <ricardo.mestre@suse.com>
Date:   Wed Sep 25 11:49:19 2024 +0100
```

Example: Uyuni

```
terraform {
  required_providers {
    uyuni = {
      source = "registry.terraform.io/svalabs/uyuni"
    }
  }
}

provider "uyuni" {
    host = "192.168.1.100"
    username = "admin"
    password = "admin"
}

data "uyuni_users" "my_users" {}
```

Example: Uyuni

```
resource "uyuni_user" "sgiertz" {
  login = "sgiertz"
  firstname = "Simone"
  lastname = "Giertz"
  email = "sgiertz@foo.bar"
  password = "test123"
}

output "users" {
  value = data.uyuni_users.my_users
}
```

Example: Uyuni

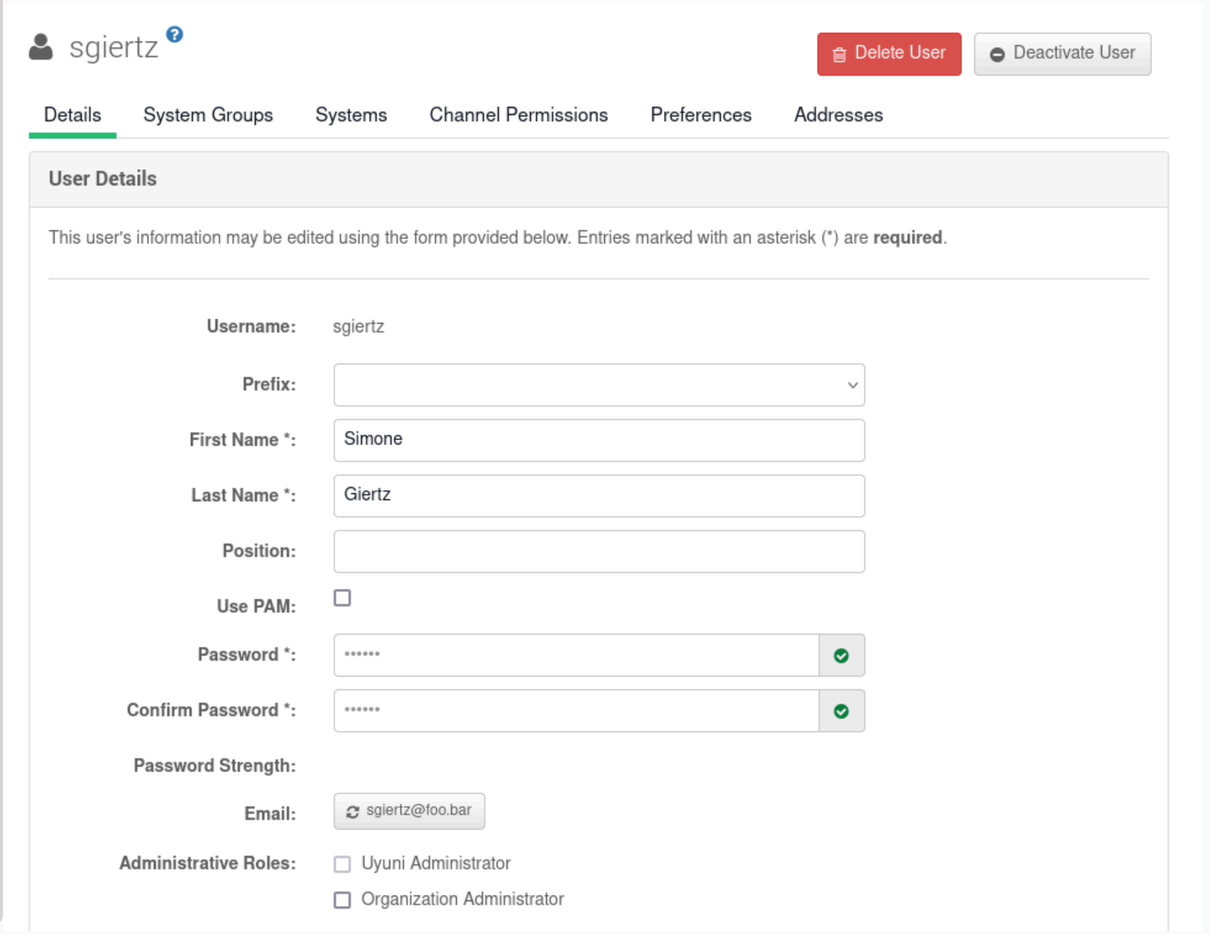
```
$ terraform apply
...
  # uyuni_user.sgiertz will be created
  + resource "uyuni_user" "sgiertz" {
      + email     = "sgiertz@foo.bar"
      + firstname = "Simone"
      + lastname  = "Giertz"
      + login     = "sgiertz"
      + password  = (sensitive value)
    }

Plan: 1 to add, 0 to change, 0 to destroy.


Do you want to perform these actions?
...

  Enter a value: yes

uyuni_user.sgiertz: Creating...
uyuni_user.sgiertz: Creation complete after 1s
```



User Details

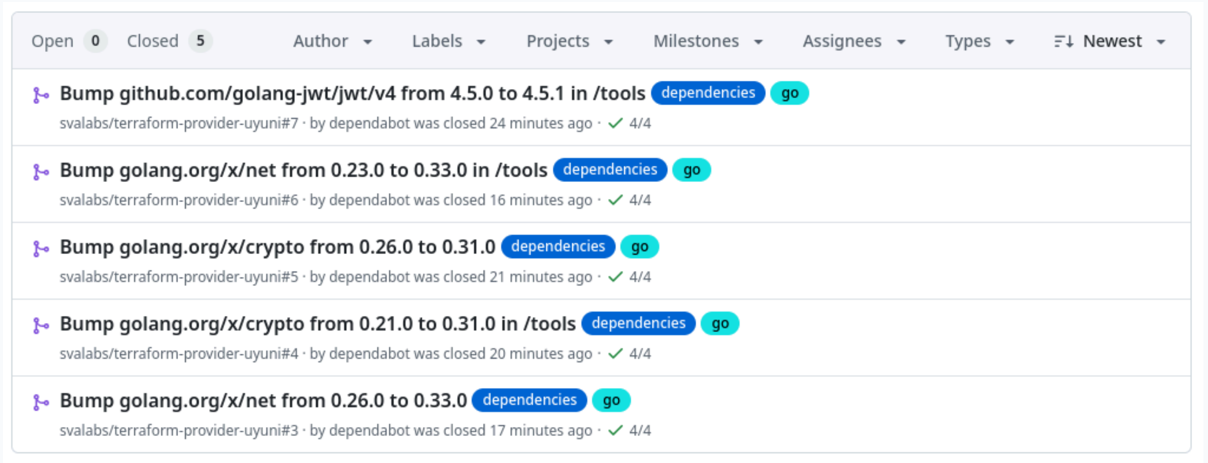
This user's information may be edited using the form provided below. Entries marked with an asterisk (*) are **required**.

Username:	sgiertz
Prefix:	
First Name *:	Simone
Last Name *:	Giertz
Position:	
Use PAM:	☐
Password *:	••••••
Confirm Password *:	••••••
Password Strength:	
Email:	sgiertz@foo.bar
Administrative Roles:	☐ Uyuni Administrator ☐ Organization Administrator

GitHub Actions

The scraffolding repository contains various actions:

- Ensures that code can be built
- Checks linting
- Running acceptance tests in a matrix



Bump github.com/golang-jwt/jwt/v4 from 4.5.0 to 4.5.1 in /tools dependencies go
svalabs/terraform-provider-uyuni#7 · by dependabot was closed 24 minutes ago · ✓ 4/4

Bump golang.org/x/net from 0.23.0 to 0.33.0 in /tools dependencies go
svalabs/terraform-provider-uyuni#6 · by dependabot was closed 16 minutes ago · ✓ 4/4

Bump golang.org/x/crypto from 0.26.0 to 0.31.0 dependencies go
svalabs/terraform-provider-uyuni#5 · by dependabot was closed 21 minutes ago · ✓ 4/4

Bump golang.org/x/crypto from 0.21.0 to 0.31.0 in /tools dependencies go
svalabs/terraform-provider-uyuni#4 · by dependabot was closed 20 minutes ago · ✓ 4/4

Bump golang.org/x/net from 0.26.0 to 0.33.0 dependencies go
svalabs/terraform-provider-uyuni#3 · by dependabot was closed 17 minutes ago · ✓ 4/4

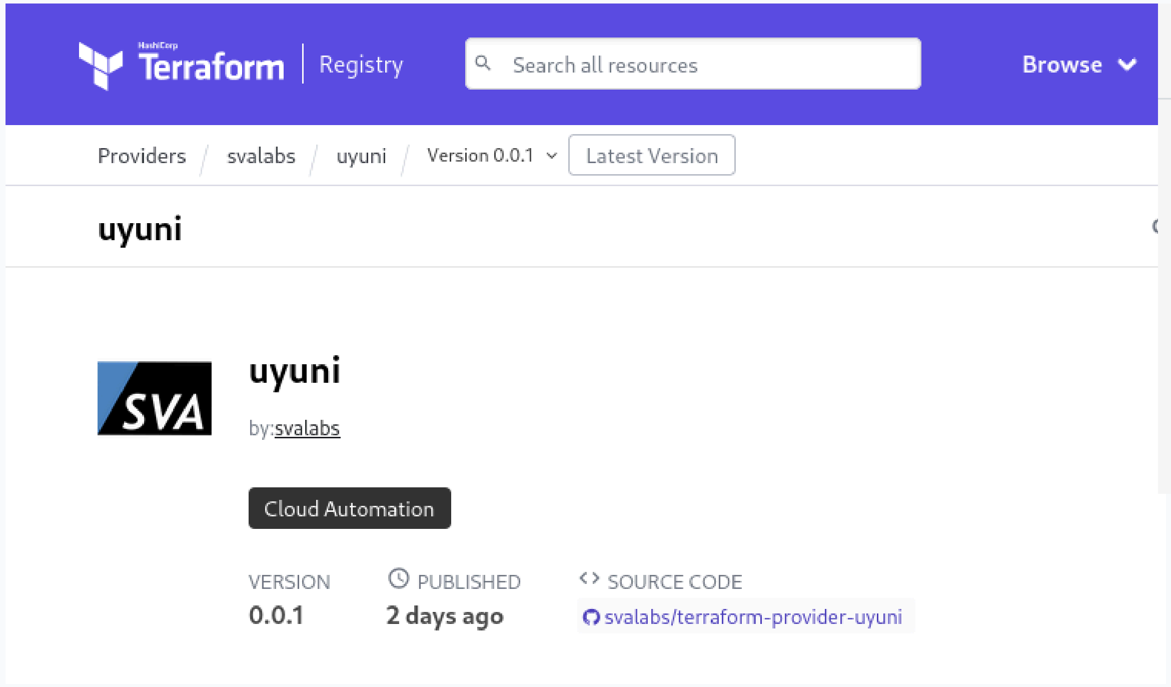
> **Note**
>
> Keep in mind to update definitions from upstream regularly.

Also, ensure to use Dependabot to simplify updating Golang modules.

Releasing

- Verify that manifest and a valid `goreleaser` configuration exists
  - Tool that automates all that boring release tasks
  - Cross-platform builds, SBOM generation, changelogs,...
- Create GPG keypair and add GitHub Action secrets
  - `GPG_PRIVATE_KEY`, `PASSPHRASE`
- Create and publish an annotated git tag, automation kicks in
- Sign-in to registry and add GPG public key
- Add and publish provider
  - Terraform Registry has an assistant
  - OpenTofu Registry has an GitHub issue template

Releasing



Providers / svalabs / uyuni / Version 0.0.1 Latest Version

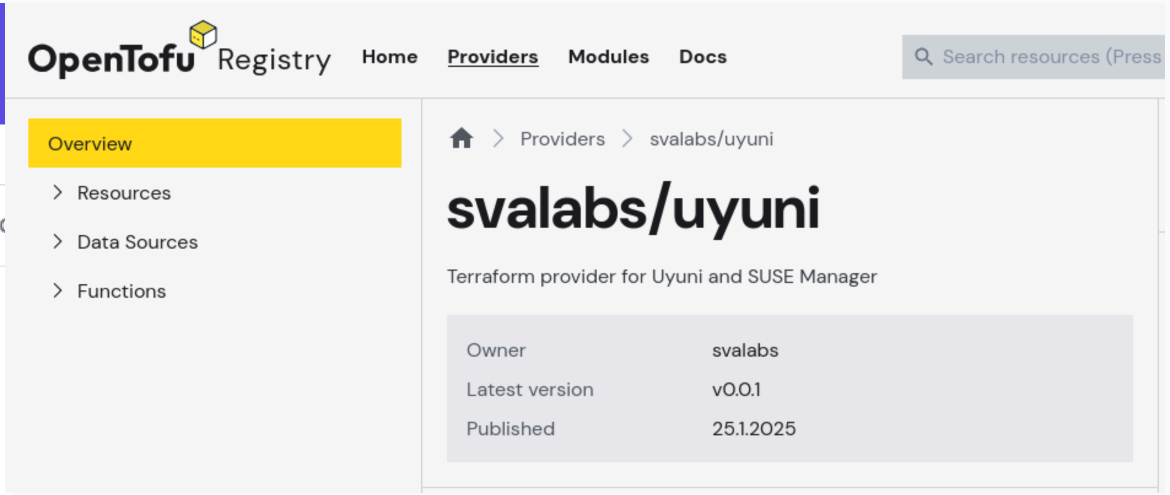
uyuni

uyuni

by:svalabs

Cloud Automation

VERSION	PUBLISHED	SOURCE CODE
0.0.1	2 days ago	svalabs/terraform-provider-uyuni



Providers > svalabs/uyuni

svalabs/uyuni

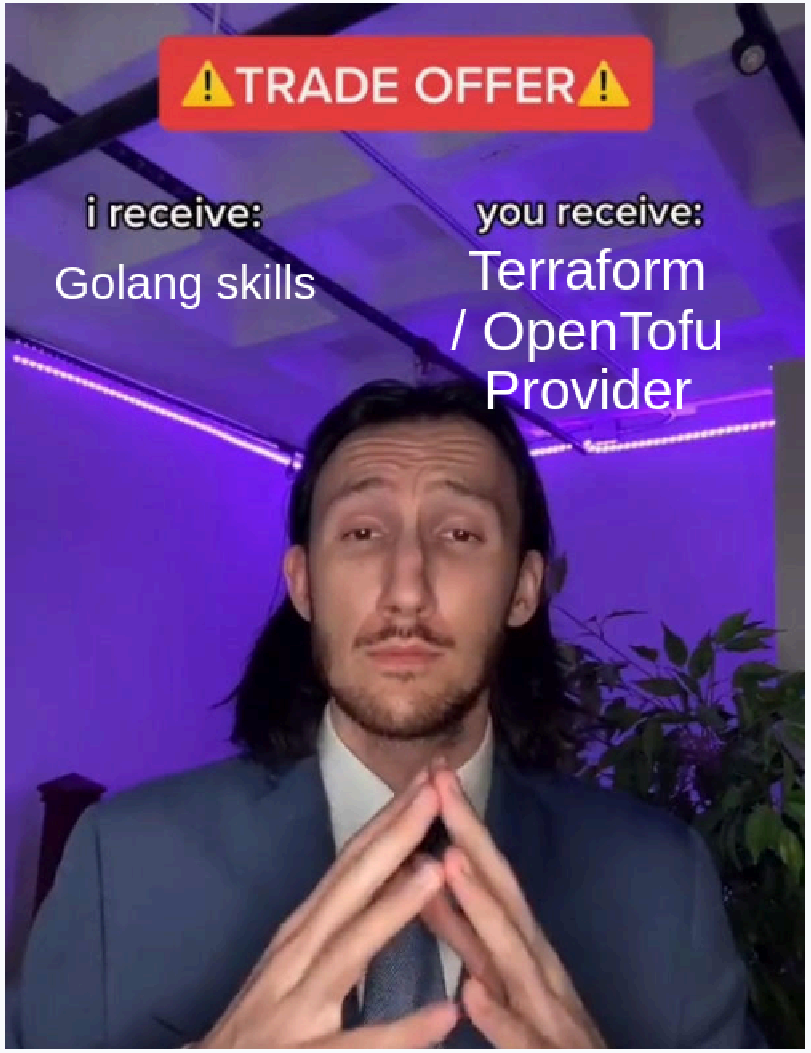
Terraform provider for Uyuni and SUSE Manager

Owner	svalabs
Latest version	v0.0.1
Published	25.1.2025

🎉

TODOs

- Our journey just began, there is a lot to do:
  - Implement acceptance testing
  - Create dedicated Golang module
  - Implement additional ressources
- Don't underestimate the learning curve
  - but it's worth it!



What's next?

Links

- Uyuni: https://www.uyuni-project.org/
- svalabs/uyuni provider: https://registry.terraform.io/providers/svalabs/uyuni
- Forgejo: https://forgejo.org/
- svalabs/forgejo provider: https://registry.terraform.io/providers/svalabs/forgejo
- Tutorial: https://developer.hashicorp.com/terraform/tutorials/providers-plugin-framework

Your own idea?

Learn some Golang basics, check-out the tutorial and see where you'll end.

It ain't much, but it's honest work

Thanks!