Terraformer PostgreSQL

OpenTofu+PG

A propos

- Julian Vanden Broeck
- Ingénieur automatisation (@Dalibo)
- Dev (🐍+🦫) de solution d'automatisation pour PG 🐘
- membre de Dalibo depuis 09/2019
- Autrefois admin sys

Sommaire

- Brève intro OpenTofu
- Utiliser OpenTofu pour des objets PostgreSQL
- Gérer une ou des instances depuis 0

Intro OpenTofu - Info générique

- Clone Libre de **Terraform**
- Outil d'**Infrastructure as Code (IaC)**
- Permet de **décrire, provisionner et gérer** des infrastructures de manière déclarative

Intro OpenTofu - DSL

- Permet d'utiliser **HCL (HashiCorp Configuration Language)**
- Approche **déclarative** : *quoi* créer, pas *comment*
- Concepts clés :
  - `provider`
  - `resource`
  - `datasource`
  - ...

> *HCL is a toolkit for creating structured configuration languages that are both human- and machine-friendly, for use with command-line tools.*

Intro OpenTofu - Fichier d’état

- Fichier d’état (`.tfstate`) :
  - Représente la **réalité connue** de l’infrastructure
  - Fait le lien entre le code et les ressources réelles
- Peut être :
  - **Local**
  - **Distant** (S3, GCS, Azure Blob, **PostgreSQL**, …)
- Contient des informations critiques, assure la cohérence et permet la collaboration

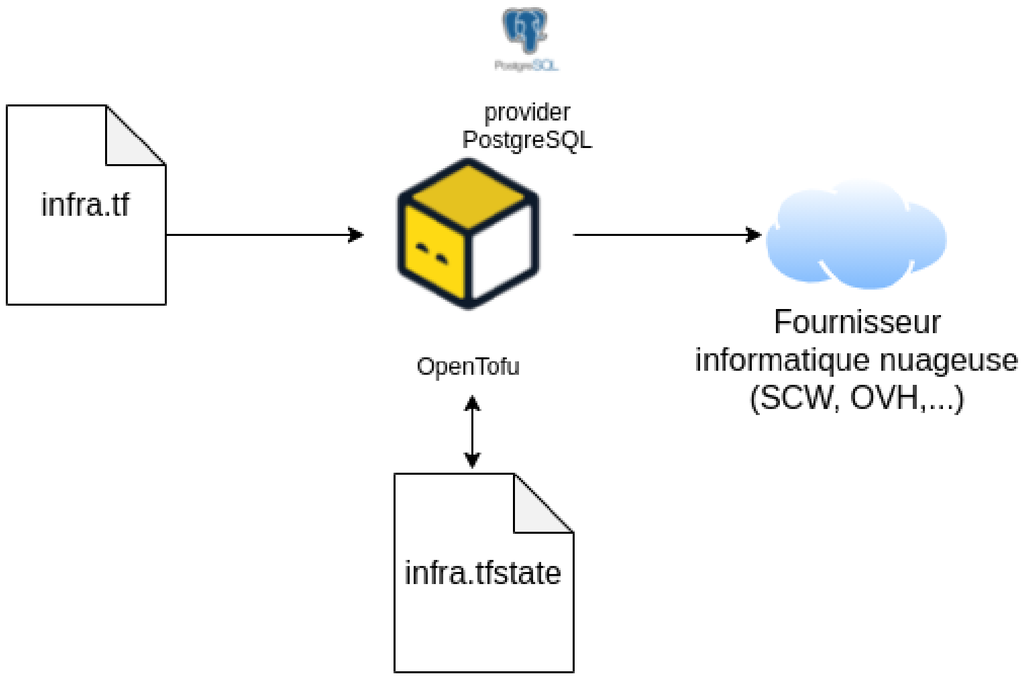
Intro OpenTofu - Exemple basique

```
resource "local_file" "demo" {
  filename = "hello.txt"
  content  = "Hello pgSession 👋"
}
```

ou encore :

```
resource "proxmox_vm_qemu" "minimal-example" {
    name                        = "minimal-example"
    boot                        = "order=scsi0;net0"
    target_node                 = "test"
    network {
        bridge    = "vmbr0"
        firewall  = true
        link_down = false
    }
}
```

OpenTofu

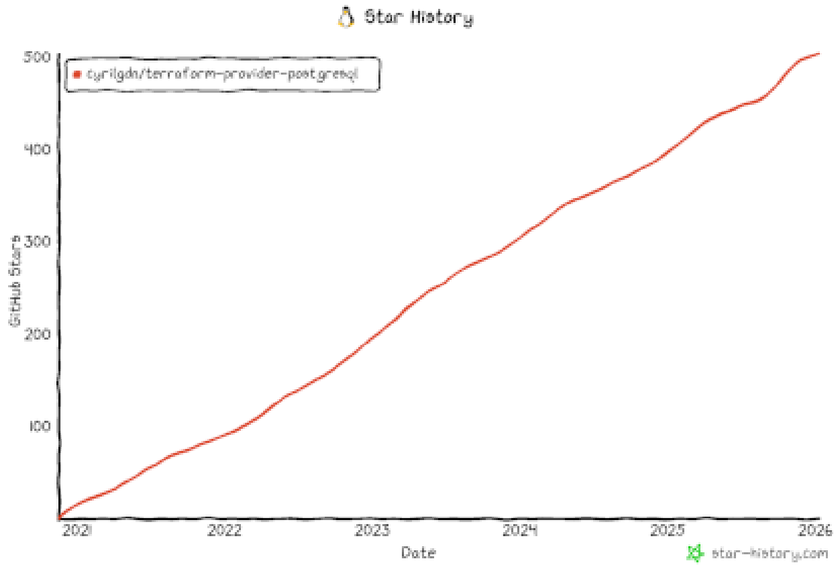

provider
PostgreSQL
infra.tf
OpenTofu
Fournisseur
informatique nuageuse
(SCW, OVH,...)
infra.tfstate

Fournisseurs pour PostgreSQL

2 types de fournisseurs (AKA provider) :

- Pour gérer des objets d'une instance (base de données, privilèges, slot de réplication,…)
- Pour gérer des instances chez des fournisseurs d'informatique nuageuse (Scaleway, OVH, AWS,…)

Fournisseur pour PG - “Leader”



- https://github.com/cyrilgdn/terraform-provider-postgresql
- https://search.opentofu.org/provider/cyrilgdn/postgresql/latest

Fournisseur pour PG - “Leader”

- Initié par Cyril Gaudin (cyrilgdn), Objectif, remplacer le projet (déprécié) initié par Harshicorp
- 500 étoiles 🤩
- 28 publications 📦
- 120 contributeurs 🩷
- 268 forks 🍴
- - de 5 ans de dev 💻

Fournisseur pour PG - ressources

- postgresql_database
- postgresql_default_privileges
- postgresql_extension
- postgresql_function
- postgresql_grant
- postgresql_grant_role
- postgresql_physical_replication_slot
- postgresql_replication_slot
- postgresql_role
- postgresql_schema
- …

Fournisseur pour PG - datasources

- postgresql_schemas
- postgresql_sequences
- postgresql_tables

Exemple - role+bases (1/2)

On initialise une instance (avec pglift, on en parle + tard) :

```
$ pglift instance create ex01 --surole-password
```

```
resource "postgresql_role" "dba_role" {
  name         = "dba_role"
  login        = true
  password     = "mypass"
  valid_until = "2027-01-01 00:00:00+00"
}

resource "postgresql_database" "databases"{
  count = 10
  owner              = postgresql_role.dba_role.name
  name               = format("db%d", count.index)
  allow_connections = true
}
```

Ensuite `tofu {init,plan,apply}`

Exemple - role+bases (2/2)

```
$ pglift role -i 18/ex01 get dba_role --output-format=json
{
  "name": "dba_role",
  "login": true,
  ...
  "valid_until": "2027-01-01T00:00:00Z",
  "validity": "2027-01-01T00:00:00Z",
  "memberships": [],
  "hba_records": [],
  "pgpass": false
}
```

```
$ pglift database -i 18/ex01 list --output-format=json | \
  jq -r '.[] | [.name, .owner] | @tsv'
db0     dba_role
...
db8     dba_role
db9     dba_role
postgres        postgres
template1       postgres
```

Ok et pour des instances ?

Plusieurs options :

- Instance(s) chez un fournisseur d'informatique nuageuse (Amazon, SCW,…), provider du fournisseur.
- On bricole une solution maison (Exemple : API qui pilote de Ansible)
- Orientation prise par Dalibo, **réutiliser pglift**, notre solution d'industrialisation (**techno connue, libre et permettant une certaine indépendance**).

pglift Multi interface

- Élément central dans la *solution d'industrialisation* de Dalibo
- Librairie Python avec *plusieurs interfaces* (CLI, Ansible, REST API ?)
- Permet de déployer *PostgreSQL+composants satellites* (sauvegarde - pgBackRest, HA - Patroni, inventaire de parc - temBoard,…)

pglift CLI - (1/2)

```
$ pglift --help
Usage: pglift [OPTIONS] COMMAND [ARGS]...

  Deploy production-ready instances of PostgreSQL
...
Commands:
  instance  Manage instances.
  pgconf    Manage configuration of a PostgreSQL instance.
  role      Manage roles.
  database  Manage databases.
  pghba     Manage entries in HBA configuration of a PostgreSQL instance.
  wal       Manage WAL replay of a PostgreSQL instance.
```

pglift CLI - (2/2)

```
$ pglift instance create ex02 --surole-password \
  --pgbackrest-stanza=stzex02 --port 8877
```

```
$ pglift instance exec ex02 -- psql
[18/ex02] postgres@~=#
```

```
$ pglift instance backup ex02
```

pglift Ansible

```yaml
- name: Create and configure my postgresql instances
  hosts: localhost
  tasks:
    - name: Create production instance
      dalibo.pglift.instance:
        name: prod
        state: started
        port: 8989
        settings:
          max_connections: 200
          shared_buffers: 1GB
        surole_password: sup3rPassw0rd
        pgbackrest:
          stanza: stzprod
```

```
$ ansible-playbook ex02/instances-create.yml
```

pglift API

⚠️ Aperçu technique +++ ⚠️

pglift == un modèle “unique” pour décrire les instances. Il est donc “simple” d’ajouter une interface / API REST.

Interface qui est (auto)documentée :

Exemple : http://127.0.0.1:8000/docs

pglift API - Création d'une instance

On initialise une instance :

```
$ http PUT :8000/instances/ name=instance_pgsession port=8989 version=18
<task_id>
```

Et le résultat :

```
$ http GET :8000/tasks/<task_id>/status
```

```
$ pglift instance get 18/instance_pgsession \
  --output-format=json
```

pglift API - Modification d'une instance

On peut ensuite modifier cette instance (modification de `max_connections`) :

```
$ http PUT :8000/instances/ name=instance_pgsession version=18 \
  settings:='{"max_connections": 20}'
<task_id>
```

On consulte ce paramètre :

```
$ http GET :8000/instances/18/instance_pgsession | \
  jq .settings.max_connections
20
```

```
$ http DELETE :8000/instances/18/instance_pgsession
```

pglift API - Et donc ?

- Plusieurs interfaces
- Une approche déclarative, idéalement pour (presque) tous les objets
- Embryon d’API
- Ouvre la porte à plusieurs clients, pourquoi pas **OpenTofu**

pglift+OpenTofu - Exemple simple

```
resource "pglift_database" "prod_db"{
  name = "mypretty_db"
  owner = "postgres"
  instance = {
    name = pglift_instance.prod_instance[0].name
    version = pglift_instance.prod_instance[0].version
  }
}

resource "pglift_instance" "prod_instance" {
  count     = 2
  name      = format("prod0%d", count.index)
  port      = (6880+count.index)
  version   = 18
  data_checksums = true
  settings = <<JSON
      {
```

pglift+OpenTofu - show, refresh & modif

Quid si on modifie une instance :

```
$ pglift pgconf -i 18/prod00 set max_connections=130
```

```
$ tofu show
$ tofu refresh
$ tofu show
```

```
$ tofu plan
$ tofu apply
```

```
$ pglift pgconf -i 18/prod00 show max_connections
```

Repo

pglift & co dispo librement : https://gitlab.com/dalibo/

Questions ?