Terraform CLI Cheat Sheet

A propos de la Terraform CLI

Terraform, un outil créé par Hashicorp en 2014, écrit en Go, permet de construire, modifier et controller la version de votre infrastructure. Cet outil a une Command Line Interface (CLI) très intuitive et puissante.

Installation

Installation grâce à 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
```

Installation possible également grâce à tfenv : un manager de version pour Terraform

Premièrement, téléchargez le binaire de tfenv et mettez le dans votre PATH.

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

Ensuite, vous pouvez installer la version de Terraform désirée :

```
$ tfenv install 0.11.10
```

Utilisation

Afficher la version

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

Initialiser Terraform

```
$ terraform init
```

C'est la première commande que vous devez éxécuter. Sinon, terraform plan, apply, destroy et import ne fonctionneront pas. Cette commande terraform init va installer :

- les modules terraform
- éventuellement un backend
- et les plugins/providers

Initialiser Terraform et forcer de ne pas demander une entrée

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

Changer la configuration du backend durant l'initialisation

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

`-reconfigure` est utilisé afin de demander à terraform de ne pas copier le state existant vers le lieu du nouveau remote state.

Get

Cette commande est utile lorsque vous avez défini des modules. Les modules sont vendored donc lorsque l'on en édite un, il faut encore récupérer (faire un get) du nouveau contenu du module.

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

Lorsque vous utilisez des modules, la première chose à faire est de faire un terraform get. La commande va faire un pull des modules dans le répertoire .terraform. A ce moment là vous avez le module vendoré.

Plan

L'étape plan vérifie la configuration à exécuter et écrit un plan à appliquer chez le fournisseur d'infrastructure.

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

C'est une fonctionnalité importante de Terraform qui permet à l'utilisateur de voir quelles actions Terraform exécutera avant d'apporter des modifications, ce qui augmentera la confiance qu'un changement aura l'effet désiré une fois appliqué.

Lorsque vous exécutez la commande terraform plan, terraform analyse tous les fichiers *.tf de votre répertoire et créé le plan.

Apply

Vous avez maintenant le state souhaité donc vous pouvez exécuter le plan.

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

Bon à savoir : Depuis terraform v0.11+, en mode interactif (pas exécuter dans une chaine de CI/CD/ dans un pipeline autonome), vous pouvez simplement exécuter la commande `terraform apply` qui affichera les actions que TF effectuera.

En générant le plan et en l'appliquant dans la même commande, Terraform peut garantir que le plan d'exécution ne changera pas, sans qu'il soit nécessaire de l'écrire sur disque. Cela réduit le risque de laisser des données potentiellement sensibles, ou d'accidentellement le versionner.

```
$ terraform apply
```

Appliquer et aprouver automatiquement

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

Appliquer et définir de nouvelle valeurs de variables

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

Appliquer uniquement un module

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

Cette option *-target* fonctionne également avec la commande *terraform plan*.

Destroy

```
$ terraform destroy
```

Supprime toutes les resources !

Un plan de suppresion peut être créé avant :

```
$ terraform plan –destroy
```

L'option `-target` permet de détruire uniquement une resource, par exemple un bucket S3 :

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

Debug

La commande Terraform console est utile pour tester les interpolations avant de les utiliser dans les ressources à créer ou modifier. Terraform console va lire le state confgiuré même si ce dernier est en remote.

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

Graph

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

Graph visuel des dépendances des ressources Terraform.

Validate

La commande validate est utilisée pour valider / vérifier la syntaxe des fichiers Terraform. Une vérification de la syntaxe est effectuée sur tous les fichiers terraform du répertoire et affiche une erreur si l'un des fichiers ne se valide pas. La vérification de la syntaxe ne couvre pas tous les problèmes courants liés à la syntaxe.

```
$ terraform validate
```

Providers

Vous pouvez utiliser un grand nombre de providers/plugins dans vos définition de ressources terraform. Il peut donc être utile de disposer d'un arbre de providers utilisés par les modules de votre projet.

```
$ terraform providers
.
├── provider.aws ~> 1.54.0
├── module.my_module
│   ├── provider.aws (inherited)
└── module.elastic
    └── provider.aws (inherited)
```

State

Pull le remote state dans une copie locale

```
$ terraform state pull > terraform.tfstate
```

Push state dans un stockage de backend remote

```
$ terraform state push
```

Cette commande est utile si par exemple à l'origine vous avez utilisé un tf state en local et qu'ensuite vous definissez un stockage de backend, dans un bucket S3 ou avec COonsul par exemple.

Comment demander à Terraform que vous avez déplacé une ressource dans un module ?

Si vous avez déplacé une ressource existante , vous devez mettre a jour le state :

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

Comment importer une ressource existante dans Terraform ?

Si vous avez une ressource existante dans votre provider d'infrastructure, vous pouvez l'importer dans votre state Terraform :

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

Workspaces

Terraform workspace est une fonctionnalité qui permet de gérer plusieurs environnements distincts.

Au lieu de créer un répertoire pour chaque environnement à gérer, nous devons simplement créer l'espace de travail/le workspace nécessaire et l'utiliser :

Créer un workspace

Cette commande créé un nouveau workspace et le sélectionne

```
$ terraform workspace new dev
```

Sélectionner un workspace

```
$ terraform workspace select dev
```

Lister les workspaces

```
$ terraform workspace list
  default
* dev
  prelive
```

Afficher le workspace courant

```
$ terraform workspace show
dev
```

Outils

jq

jq est une "command-line JSON processor" légère. Combiné avec les outputs Terraform cela peut être très puissant.

Installation

Pour Linux :

```
$ sudo apt-get install jq
```

ou

```
$ yum install jq
```

Pour OS X :

```
$ brew install jq
```

Utilisation

Il suffit de définir des outputs dans un module, et lorsque vous exécutez *terraform apply*, les outputs sont affichés :

```
$ terraform apply
...
Apply complete! Resources: 0 added, 0 changed,
 0 destroyed.

Outputs:

elastic_endpoint = vpc-toto-12fgfd4d5f4ds5fngetwe4.
eu-central-1.es.amazonaws.com
```

Vous pouvez extraire la valeur désirée afin de l'utiliser dans un script par exemple. Avec jq c'est facile :

```
$ terraform output -json
{
    "elastic_endpoint": {
        "sensitive": false,
        "type": "string",
        "value": "vpc-toto-12fgfd4d5f4ds5fngetwe4.
        eu-central-1.es.amazonaws.com"
    }
}

$ terraform output -json | jq '.elastic_endpoint.value
"vpc-toto-12fgfd4d5f4ds5fngetwe4.eu-central-1.
es.amazonaws.com"
```

Terraforming

Si vous avez un compte AWS existant avec des composants existants tels que des buckets S3, SNS, VPC… Vous pouvez utiliser l'outil terraforming, un outil écrit en Ruby, qui extrait les ressources AWS existantes et les convertissent en fichiers Terraform !

Installation

`$ sudo apt install ruby` ou `$ sudo yum install ruby`

et

```
$ gem install terraforming
```

Utilisation

Pré-requis :

Comme avec Terraform, vous devez settez les credentials AWS

```
$ export AWS_ACCESS_KEY_ID="an_aws_access_key"
$ export AWS_SECRET_ACCESS_KEY="a_aws_secret_key"
$ export AWS_DEFAULT_REGION="eu-central-1"
```

Vous pouvez également spécifier le profile des credentials dans *~/.aws/credentials* avec l'option *–profile*.

```
$ cat ~/.aws/credentials
[aurelie]
aws_access_key_id = xxx
aws_secret_access_key = xxx
aws_default_region = eu-central-1

$ terraforming s3 --profile aurelie
```

Utilisation

```
$ terraforming --help
Commands:
terraforming alb # ALB
...
terraforming vpc # VPC
```

Exemple :

```
$ terraforming s3 > aws_s3.tf
```

Remarques : terraforming ne peut pas extraire, pour le moment, les ressources de type API Gateway donc vous devez les écrire manuellement.

Authors :


@aurelievache
Cloud Dev(Ops) chez Continental

v1.0.2