

TP 3 : Provisionement et Terraform

Prérequis

Installer Docker sur la machine. (Voir TP précédent)
Puis installer Terraform

- Sur Ubuntu

```
wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o \
/usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(grep -oP '(?<=UBUNTU_CODENAME=).*' \
/etc/os-release || lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform
```

- Sur MacOS, avec Brew :

```
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
```

- Sur Windows, avec Chocolatey :

```
choco install terraform
```

Une fois l'installation de Terraform complétée, vérifier sa version en lançant
`terraform version`

Simplifiez vous la vie

- Installer l'autocomplétion de Terraform et utiliser un alias

```
terraform -install-autocomplete
alias tf="terraform"
```

1 Premières ressources Terraform

Nous allons créer nos premières ressources Terraform et inspecter le fichier d'état.

1.1 Allons récupérer le provider pour Docker

La référence du provider Docker est disponible à l'adresse suivante : <https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs>

Pour l'utiliser, nous devons l'indiquer dans notre configuration. Créez un nouveau fichier `main.tf` et copiez-y le contenu suivant. Rappelez-vous que le nom du fichier n'a pas d'importance.

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.2"
    }
  }
}
```

Le bloc `terraform` contient un bloc intégré `required_providers` qui comprend une liste d'arguments. Chaque clé d'argument est notre identifiant pour un provider ; la valeur contient les arguments `source` et `version`. Le `source` est l'identifiant sur le registre <https://registry.terraform.io/>, et `version` est une contrainte sur la version que nous demandons.

Ensuite, exécutez `terraform init` dans la console. La sortie devrait ressembler à ceci :

```
$ terraform init
```

```
Initializing the backend...
```

```
Initializing provider plugins...
```

```
- Finding kreuzwerker/docker versions matching "~> 3.0.2"...  
- Installing kreuzwerker/docker v3.0.2...  
- Installed kreuzwerker/docker v3.0.2 (self-signed, key ID BD080C4571C6104C)
```

```
[...]
```

```
Terraform has been successfully initialized!
```

Il a téléchargé le provider Docker requis et créé un fichier de verrouillage contenant la version et les empreintes (hashes) de notre provider.

Pour voir cela plus concrètement, explorez le dossier `.terraform` (avec `ls` et `cd`, ou bien avec `tree -a`).

Le provider `docker` peut désormais être appelé avec

```
provider "docker" {}
```

1.2 Créer une image redis avec Terraform

Sous le bloc Terraform, copiez ce qui suit afin de créer une nouvelle ressource gérée avec le type `docker_image` et le nom `redis` :

```
resource "docker_image" "redis" {  
  name = "docker.io/redis:6.0.5"  
}
```

La documentation des ressources `docker_image` est disponible à l'adresse <https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs/resources/image>. Dans la section Schéma, seul l'argument `name` est obligatoire, mais de nombreux autres arguments facultatifs sont configurables. Les attributs en lecture seule ne sont pas configurables dans les fichiers `.tf` ; leur valeur est déterminée par le provider lui-même.

Désormais, lançons `terraform plan` :

```
$ terraform plan
```

```
Terraform used the selected providers to generate the following execution  
plan. Resource actions are indicated with the following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# docker_image.redis will be created  
+ resource "docker_image" "redis" {  
  + id          = (known after apply)  
  + image_id    = (known after apply)  
  + name        = "docker.io/redis:6.0.5"  
  + repo_digest = (known after apply)  
}
```

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

Le symbole + indique que la ressource va être créée. Les commentaires sont préfixés par #.

Nous constatons ici que les attributs définis par le provider seront finalement intégrés à l'état de la ressource, mais leur valeur est inconnue au moment de la planification. Ce n'est pas toujours le cas : les attributs du fournisseur peuvent être absents de l'état ou avoir une valeur connue au moment de la planification.

Désormais, lançons `terraform apply`. Un plan de déploiement apparaît d'abord, avec un paragraphe supplémentaire qui invite à la confirmation.

Une fois accepté, le plan va s'exécuter.

```
docker_image.redis: Creating...
docker_image.redis: Still creating... [10s elapsed]
docker_image.redis: Creation complete after 14s [id=sha256:2355926154447ec75b25666ff5df14d1ab54f8bb4abf
```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Vérifions désormais l'état de notre infrastructure.

```
$ terraform show
# docker_image.redis:
resource "docker_image" "redis" {
  id          = "sha256:23559...docker.io/redis:6.0.5"
  image_id    = "sha256:23559..."
  name        = "docker.io/redis:6.0.5"
  repo_digest = "redis@sha256:800f25..."
}
```

Notons que certains attributs ont été calculé par le provider. Vérifions maintenant que notre image a bien été créée sur notre instance locale de Docker :

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
redis         6.0.5     800f2587bf33   5 years ago    145MB
```

1.3 Créer un conteneur Docker exécutant notre image

La référence est disponible ici : <https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs/resources/container>.

Ajoutez le bloc resource suivant dans `main.tf` :

```
resource "docker_container" "db" {
  name = "redis_db"
  image = docker_image.redis.image_id
}
```

Ici, `name` et `image` sont requis. La valeur de `image` fait référence à la ressource que nous avons déjà déclarée, de type `docker_image` avec le nom `redis`, et récupère son attribut `image_id`.

Exécutez `terraform plan` puis `terraform apply`, puis vérifiez avec :

```
$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS        NAMES
5b3910075b61   235592615444        "docker-entrypoint.s..."  5 seconds ago  Up 4 seconds  6379/tcp    redi
```

1.4 Inspecter l'état Terraform

Vous pouvez consulter directement le fichier JSON `terraform.tfstate` ou utiliser `terraform show`, mais une manière plus idiomatique est d'utiliser la commande `state`.

Exécuter `terraform state` seul affichera l'aide et l'ensemble des sous-commandes disponibles. Listons toutes les ressources connues dans l'état :

```
$ terraform state list
docker_container.db
docker_image.redis
```

Une commande utile est aussi `terraform state show`, qui affiche tout l'état d'une ressource donnée :

```
terraform state show docker_container.db
```

N'hésitez pas à abuser de ces commandes d'information !

2 Déployer une application à deux conteneurs

Nous allons déployer une simple application qui enregistre des « guests » (chaînes provenant d'une zone de texte) dans une base de données, et affiche la liste de tous les invités. Pour la suite du TP, nous utiliserons le port 8080. Veillez à bien fermer toutes les applications susceptibles de l'utiliser.

- Sur Ubuntu ou MacOS

```
sudo lsof -i :8080
```

- Sur Windows, avec Powershell :

```
Get-Process -Id (Get-NetTCPConnection -LocalPort 8080).OwningProcess
```

2.1 Comment cela fonctionne

En plus du conteneur Redis (base de données), nous avons un serveur web frontal **gb-frontend** qui doit connaître l'adresse de la base.

Il le fait en lisant la variable d'environnement `GET_HOSTS_FROM`, qui précise la méthode d'accès : sa valeur peut être `dns` ou `env`.

Nous utiliserons la méthode `dns` : le frontend accèdera à la base via les noms de domaine codés en dur **redis-leader** et **redis-follower**. Il y a deux noms car Redis peut être distribué sur plusieurs nœuds organisés avec un leader et des followers. Ici, nous n'aurons qu'un seul conteneur qui sera à la fois leader et follower.

Ainsi, nous devons faire en sorte que les noms de domaine **redis-leader** et **redis-follower** pointent vers notre conteneur de base de données. Pour cela, nous allons configurer notre frontend en ajoutant une entrée dans `/etc/hosts`. Heureusement, la ressource `docker_container` possède un bloc `host {}` qui fait exactement cela. Nous fournissons un nom d'hôte et une IP. Rappelez-vous qu'un bloc imbriqué crée un tableau : nous pouvons donc définir plusieurs blocs `host`.

Note : Exécutez `terraform state show docker_container.db` et cherchez un attribut qui ressemble à l'IP du conteneur. Nous ferons référence à cette IP par le chemin de l'attribut.

Enfin, nous devons exposer le port 80 du conteneur sur notre port local 8080 afin d'y accéder via le navigateur avec `localhost:8080`.

2.2 En pratique

Téléchargez l'archive **gb-frontend.zip** du code source du frontend et extrayez-la dans votre répertoire courant. Ajoutez ce qui suit dans votre `main.tf` :

```
resource "docker_image" "guestbook-frontend" {
  name = "gb-frontend"
  build {
    context = "../gb-frontend/"
  }
}

resource "docker_container" "frontend" {
  name = "guestbook_frontend"
  image = docker_image.guestbook-frontend.image_id
  ports {
    internal = "80"
    external = "8080"
  }
}
```

```

}
env = [
  "GET_HOSTS_FROM=dns"
]
host {
  host = "redis-leader"
  ip = docker_container.db.network_data[0].ip_address
}
host {
  host = "redis-follower"
  ip = docker_container.db.network_data[0].ip_address
}
}

```

Important : Assurez-vous de bien comprendre ce code. Posez des questions si ce n'est pas le cas.

Faites un plan et appliquez si tout vous paraît correct.

Accédez ensuite à l'application et testez-la : écrivez quelque chose et cliquez sur *Submit*. La chaîne devrait s'afficher en dessous.

3 Variable de sortie

Il serait pratique que notre module racine (ici simplement `main.tf`) nous montre directement le point d'accès de l'application une fois déployée.

Pour cela, ajoutez le bloc `output` suivant à votre configuration :

```

output "app_endpoint" {
  value = "http://localhost:${docker_container.frontend.ports[0].external}"
  description = "The URL endpoint to access the Guestbook application"
}

```

Notez l'interpolation de chaîne avec le marqueur `${ }`. L'expression à l'intérieur est interprétée par Terraform, ici nous récupérons le port externe de notre conteneur frontend.

Planifiez et appliquez à nouveau.

Le résultat de l'application comporte maintenant une section supplémentaire :

Outputs:

```
app_endpoint = "http://localhost:8080"
```

Nous pouvons aussi afficher les variables de sortie avec :

```
terraform output
```

4 Nettoyage

Nous avons terminé. Supprimons notre application avec toutes ses ressources associées :

```
terraform destroy
```

Note : Cette commande est en quelque sorte l'inverse de `apply`. Toutes les ressources gérées dans le fichier de configuration (c'est-à-dire les blocs `resource` dans les fichiers `.tf`) sont détruites et supprimées de l'état.

Vérifiez avec :

```

docker state list
docker container ls -a
docker image ls -a

```