Terraform sur Azure

Salah Gontara
2022-23

Sommaire

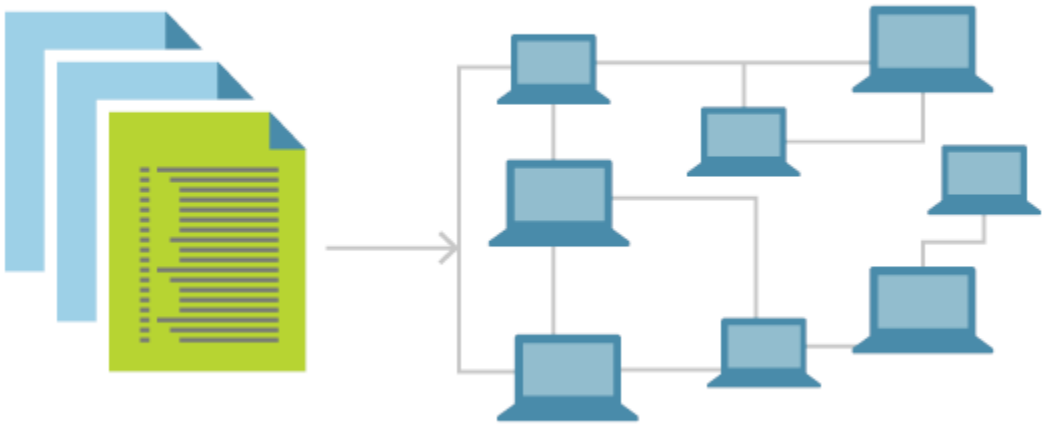
Qu’est-ce que Terraform?

```
resource "azurerm_virtual_machine" "testapp" {
  name                = "${var.prefix}-vm"
  location            = "${var.location}"
  resource_group_name = "${azurerm_resource_group.myresourcegroup.name}"
  vm_size             = "${var.vm_size}"
  network_interface_ids         = ["${azurerm_network_interface.testapp-nic.id}"]}
```

- Documentation exécutable
- Lisible par l’homme et la machine
- Facile à apprendre
- Tester, partager, réutiliser, automatiser
- Fonctionne sur tous les principaux fournisseurs de cloud

Infrastructure en tant que code (IaC)

- L’infrastructure en tant que code (IaC) est le processus de gestion et de provisionnement de l’infrastructure cloud avec des fichiers de définition lisibles par machine.
- Documentation exécutable.



IaC nous permet de

- Fournir un flux de travail codifié pour créer l'infrastructure
- Modifier et mettre à jour l'infrastructure existante
- Tester les modifications en toute sécurité à l'aide de **terraform plan** en mode d'exécution à sec
- Intégrer avec les flux de travail de code d'application (Git, Azure DevOps, outils CI/CD)
- Fournir des modules réutilisables pour faciliter le partage et la collaboration
- Appliquer la politique de sécurité et les normes organisationnelles
- Permettre la collaboration entre différentes équipes

Autres outils IaC

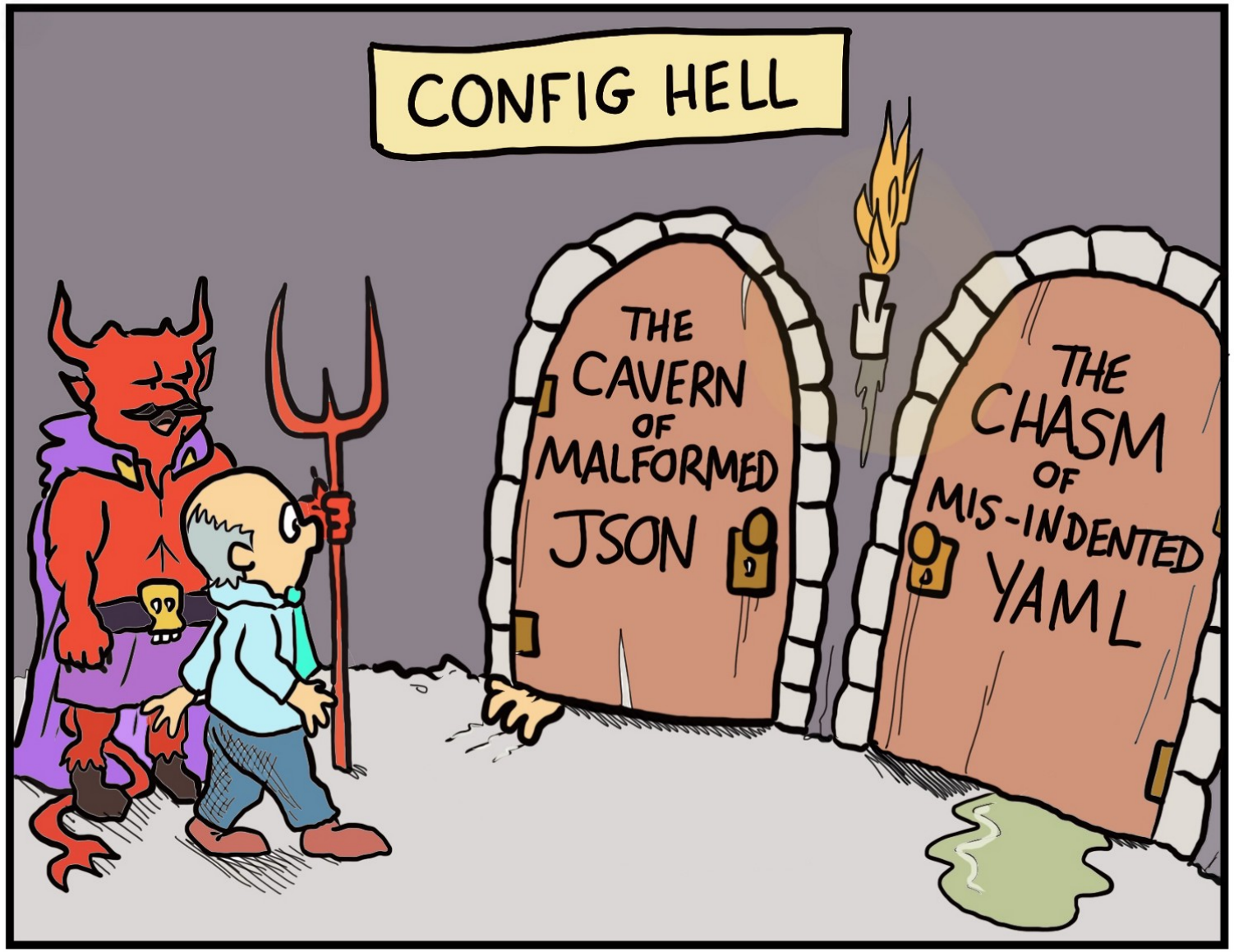
- Ces outils fonctionnent bien pour configurer le système d'exploitation et l'application.
- Ils ne sont pas spécialement conçus pour provisionner l'infrastructure cloud et les services de plateforme.

Outils de provisionnement cloud natifs

- Chaque cloud possède son propre outil de provisionnement basé sur YAML ou JSON.
- Terraform peut être utilisé par tous les principaux fournisseurs de cloud et hyperviseurs de machines virtuelles.

FaaS and Furious by Forrest Brazeal

A CLOUD GURU



"Come on, make up your mind -
or it's back to the Sinkhole of Nested XML."

Terraform vs. JSON

- Le code Terraform (HCL) est facile à apprendre et facile à lire.
- Il est également 50 à 70% plus compact qu'une configuration JSON équivalente.

ARM JSON:

```
"name": "[concat(parameters('PilotServerName'), '-vm')]",
```

Terraform:

```
name = "${var.PilotServerName}-vm"
```

Pourquoi choisir Terraform sur Azure ?

- Pouvoir prendre en charge de l’infrastructure multi-cloud et hybride
- Migrer à partir d’autres fournisseurs de cloud
- Augmenter la vitesse de provisionnement
- Améliorer l’efficacité
- Réduire les risques

Qu’est-ce que Terraform?

- Terraform est un outil de provisionnement open source.
- Il est livré comme un seul binaire qui est écrit en Go.
- Terraform est multiplateforme et peut fonctionner sous Linux, Windows ou MacOS.

HashiCorp
Terraform

Les commandes Terraform

- Terraform est un outil en ligne de commande.
- Les commandes Terraform sont soit saisies manuellement, soit exécutées automatiquement à partir d'un script.
- Les commandes sont les mêmes que vous soyez sous Linux, Windows ou MacOS.

```
# Commandes de base
terraform version
terraform help
terraform init
terraform plan
terraform apply
terraform destroy
```

Help sur Terraform

```
$ terraform help
Usage: terraform [-version] [-help] <command> [args]
...
Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    graph              Create a visual graph of Terraform resources
```

Code Terraform

- Le code Terraform est basé sur la boîte à outils HCL2.
- HCL signifie HashiCorp Configuration Language.
- Le code Terraform, ou simplement terraform, est un langage déclaratif pour provisionner l'infrastructure sur n'importe quel cloud ou plate-forme.

```
resource "azurerm_virtual_network" "vnet" {
  name                = "${var.prefix}-vnet"
  location            = "${azurerm_resource_group.myresourcegroup.location}"
  address_space       = ["${var.address_space}"]
  resource_group_name = "${azurerm_resource_group.myresourcegroup.name}"
}
```

Les espaces de travail

- Un espace de travail terraform est simplement un dossier ou un répertoire qui contient le code terraform.
- Les fichiers Terraform se terminent toujours par une extension *.tf ou *.tfvars.
- La plupart des espaces de travail terraform contiennent au moins trois fichiers :
  - **main.tf** - La plupart de votre code fonctionnel ira ici.
  - **variables.tf** - Ce fichier est destiné au stockage des variables.
  - **outputs.tf** - Définit ce qui est affiché à la fin d'une terraform.

Terraform Init

```
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 1.35.0
...
provider.azurerm: version = "~> 1.35"

Terraform has been successfully initialized!
```

- Terraform récupère tous les fournisseurs et modules requis et les stocke dans le répertoire .terraform.
- Si vous ajoutez, modifiez ou mettez à jour vos modules ou fournisseurs, vous devrez exécuter init à nouveau.

Terraform Plan

```
$ terraform plan
An execution plan has been generated and is shown below.
Terraform will perform the following actions:
  # azurerm_resource_group.myresourcegroup will be created
  + resource "azurerm_resource_group" "myresourcegroup" {
      + id       = (known after apply)
      + location = "centralus"
      + name     = "bugsbunny-workshop"
      + tags     = (known after apply)
    }
Plan: 1 to add, 0 to change, 0 to destroy.
```

- Prévisualisez vos modifications avec terraform plan avant de les appliquer.

Où les variables sont-elles définies?

```
variable "prefix" {
  description = "préfixe de nom"
}

variable "location" {
  description = "Région de création."
  default     = "West Europe"
}
```

- Les variables terraform sont placées dans un fichier appelé variables.tf. Les variables peuvent avoir des paramètres par défaut.
- Si vous omettez la valeur par défaut, l'utilisateur sera invité à entrer une valeur.

Anatomie d’une ressource

- Chaque ressource terraform est structurée exactement de la même manière.
  - resource = Mot-clé de niveau supérieur
  - type = Type de ressource. Exemple : azurerm_virtual_machine.
  - name = Nom arbitraire pour faire référence à cette ressource. Utilisé en interne par terraform. Ce champ ne peut pas être une variable.

```
resource "type" "name" {
  parameter = "foo"
  parameter2 = "bar"
  list = ["one", "two", "three"]
}
```

Configuration du fournisseur Terraform

- Le programme de base terraform nécessite au moins un fournisseur pour construire quoi que ce soit.
- Vous pouvez configurer manuellement la ou les versions d'un fournisseur que vous souhaitez utiliser.
- Si vous omettez cette option, Terraform utilisera par défaut la dernière version disponible du fournisseur.

```
provider "azurerm" {
  version = "=1.35.0"
}
```

Terraform Apply

```
$ terraform apply
An execution plan has been generated and is shown below.

Terraform will perform the following actions:
  # azurerm_resource_group.myresourcegroup will be created
  + resource "azurerm_resource_group" "myresourcegroup" {
      + id       = (known after apply)
      + location = "centralus"
      + name     = "seanc-workshop"
      + tags     = (known after apply)
    }

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

“terraform apply” exécute un plan, puis si vous l’approuvez, il applique les modifications.

Terraform Destroy

```
$ terraform destroy
An execution plan has been generated and is shown below.

Terraform will perform the following actions:

  # azurerm_resource_group.myresourcegroup will be destroyed
  - resource "azurerm_resource_group" "myresourcegroup" {
      - id       = "/subscriptions/ -> null
      - location = "West Europe" -> null
      - name     = "seanc-workshop" -> null
      - tags     = {} -> null
    }

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

“terraform destroy” fait le contraire. Si vous approuvez, votre infrastructure est détruite.

Terrform format

- Terraform est livré avec un formateur / nettoyeur de code intégré.
- Il peut rendre toutes vos marges et l’indentation de la liste propres et bien rangées.

```
terraform fmt
```

Main.tf

- Le premier fichier s'appelle main.tf. C'est là que vous stockez normalement votre code terraform.
- Avec une infrastructure plus grande et plus complexe, vous pouvez diviser cela en plusieurs fichiers.

```
# This is the main.tf file.
resource "azurerm_resource_group" "tpazure" {
  name     = "${var.prefix}-vault-workshop"
  location = "${var.location}"
}

resource "azurerm_virtual_network" "vnet" {
  name                = "${var.prefix}-vnet"
  location            = "${azurerm_resource_group.tpazure.location}"
  address_space       = ["${var.address_space}"]
  resource_group_name = "${azurerm_resource_group.tpazure.name}"
}
...
```

Variables.tf

- Le deuxième fichier s'appelle variables.tf. C'est ici que vous définissez vos variables et éventuellement définir certaines valeurs par défaut.

```
variable "prefix" {
  description = "This prefix will be included in the name of most resources."
}

variable "location" {
  description = "The region where the virtual network is created."
  default     = "centralus"
}

variable "address_space" {
  description = "The address space that is used by the virtual network."
  default     = "10.0.0.0/16"
}
```

Outputs.tf

- Le fichier de sortie est l'endroit où vous configurez les messages ou les données que vous souhaitez afficher à la fin d'un terraform apply.

```
output "Vault_Server_URL" {
  value = "http://${azurerm_public_ip.vault-pip.fqdn}:8200"
}

output "MySQL_Server_FQDN" {
  value = "${azurerm_mysql_server.mysql.fqdn}"
}

output "catapp_url" {
  value = "http://${azurerm_public_ip.catapp-pip.fqdn}"
}
```

TP: Provisionnement et Configuration des VMs sur Azure

https://gitlab.com/eps_devops/terraform-intro