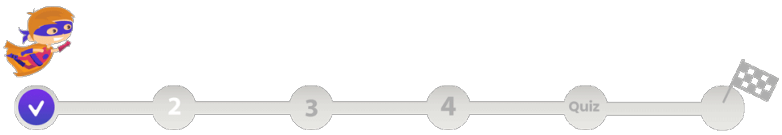
2 - Infrastructure as code



2. Infrastructure as code

Dans la philosophie DevOps, il existe une histoire pour raconter la **nécessité de basculer** vers une **Infrastructure as Code** (IaC).

Dans un Datacenter très, très lointain, un ancien groupe tout puissant plus connu sous le nom de « sysadmins » avait pour l'habitude de **déployer manuellement** toute infrastructure informatique. Chaque serveur, chaque information de table de routage, chaque configuration de base de données ou de load balancer étaient gérés à la main par leurs soins. C'était un âge où régnait la peur, la peur de l'interruption de service, la peur d'une erreur de configuration, la peur des mises en production, qui ont commencé à être interdites le vendredi. Mais la plus grande crainte des dirigeants venait du fait que tout le savoir était dans la tête de ces « sysadmins ».
Qu'arriverait-il de l'entreprise si ces individus décidaient de basculer vers le côté obscur ? De prendre des vacances voire même de quitter l'entreprise ? Toute la production était alors paralysée.

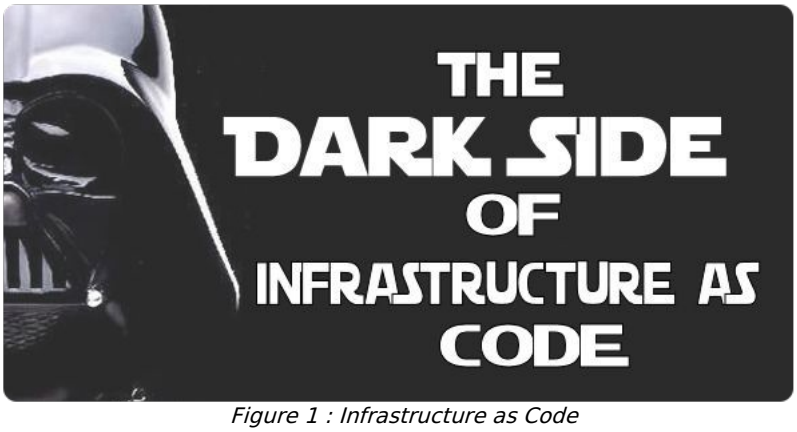


Figure 1 : Infrastructure as Code

C'est alors qu'un groupe de rebelle associé au mouvement DevOps est arrivé avec une solution miracle : l'**Infrastructure as Code.**

2.1 Pourquoi utiliser l'Infrastructure as Code (IaC)

Plutôt que de configurer manuellement un serveur en utilisant une interface utilisateur Web ou un service sécurisé d'accès à distance tel que SSH, l'**idée** derrière le **IaC** est d'**écrire du code pour définir, provisionner** et **gérer votre infrastructure**.

Cela présente un **certain nombre d'avantages** :

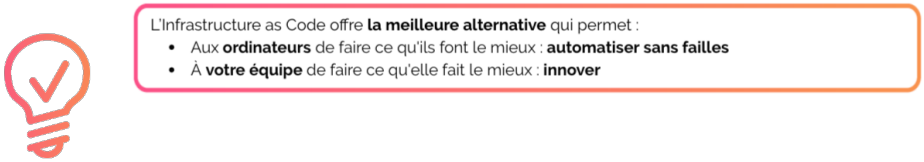
- Vous pouvez **automatiser** l'ensemble de votre processus de provisionnement et de déploiement, ce qui le rend beaucoup **plus rapide et fiable** que n'importe quel processus manuel.
- Vous pouvez représenter l'état de votre infrastructure dans **des fichiers sources** que n'importe qui peut lire plutôt que la tête d'un administrateur système.
- Vous pouvez stocker ces fichiers sources dans **un système de contrôle de version,** ce qui signifie que l'historique complet de votre infrastructure est maintenant versionné, vous pouvez plus facilement déboguer les problèmes en comparant deux versions et si dans un cas extrême, revenir aux anciennes versions.
- Vous pouvez **valider chaque changement d'infrastructure par le biais de revues de code et de tests automatisés.**
- Vous pouvez **créer des briques d'infrastructure réutilisables, documentées et éprouvées** qui facilitent la

mise à l'échelle et l'évolution de votre infrastructure.

2.2 Augmenter le bonheur de votre équipe

Déployer du code est une tâche répétitive et fastidieuse. Un ordinateur peut faire ce genre de choses rapidement et de façon fiable, mais un être humain sera lent et sujet aux erreurs d'inattention. Ce type de travail est généralement déprécié par vos équipes, car il n'implique aucune créativité, aucun défi et aucune reconnaissance. Vous pourriez déployer du code parfaitement pendant des mois, sans que personne ne s'en aperçoive, mais le jour où il y aura un problème, tout le monde vous tombera dessus.

Cela a tendance à créer un environnement stressant et désagréable.



2.3 Gestion de configuration contre Orchestration

Maintenant que vous ne doutez plus de la nécessité d'utiliser l'infrastructure as code pour votre entreprise, l'une des tâches les plus compliquées reste de savoir **quel outil vous allez utiliser.**
La plupart des outils de gestion de configuration ou d'orchestration peuvent être utilisés pour gérer l'infrastructure sous forme de code. Ils sont majoritairement **open source**, soutenus par de **larges communautés de contributeurs** et travaillent avec de nombreux **fournisseurs de cloud.** Ils sont bien documentés, à la fois en termes de **documentation officielle et de ressources communautaires.**

Chef, Puppet, Ansible et SaltStack font partie des outils de « **gestion de configuration** », c'est-à-dire qu'ils sont conçus pour **installer et gérer des logiciels** sur des serveurs existants. **CloudFormation et Terraform** quant à eux sont des « **outils d'orchestration** », c'est-à-dire qu'ils sont conçus pour provisionner les serveurs eux-mêmes, laissant à d'autres outils le soin de configurer ces serveurs. Vous pouvez également utiliser des orchestrateurs de conteneurs tels que **Kubernetes, Rancher, Mesos** ou autres.

Ces deux catégories ont cependant tendance à se rejoindre. La plupart des outils de gestion de configuration intègre des mécanismes de provisionnement et la plupart des outils d'orchestration permettent de gérer des configurations.

Figure 2 : Logo de Terraform

Si vous utilisez **Docker**, la grande majorité de vos besoins de gestion de configuration est déjà prise en charge. Avec **Docker,** vous pouvez **créer des conteneurs** qui ont déjà installé et configuré tous les logiciels dont votre serveur a besoin. Une fois que vous avez une telle image, tout ce dont vous avez besoin est un serveur pour l'exécuter. Et si tout ce que vous avez à faire est de **provisionner un groupe de serveurs,** alors un **outil d'orchestration comme Terraform** sera généralement mieux **adapté** qu'un **outil de gestion de configuration.**

```
provider "digitalocean" {
  # You need to set this in your .bashrc
  # export DIGITALOCEAN_TOKEN="Your API TOKEN"
  #
}

resource "digitalocean_droplet" "mywebserver" {
  # Obtain your ssh_key id number via your account.
  ssh_keys           = [12345678]         # Key example
  image              = "${var.ubuntu}"
  region             = "${var.do_ams3}"
  size               = "s-1vcpu-1gb"
  private_networking = true
  backups            = true
  ipv6               = true
  name               = "mywebserver-ams3"

  provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/bin",
      "sudo apt-get update",
      "sudo apt-get -y install nginx",
    ]

    connection {
      type     = "ssh"
      private_key = "${file("~/.ssh/id_rsa")}"
      user     = "root"
      timeout  = "2m"
    }
  }
}

resource "digitalocean_domain" "mywebserver" {
  name       = "www.mywebserver.com"
  ip_address = "${digitalocean_droplet.mywebserver.ipv4_address}"
}

resource "digitalocean_record" "mywebserver" {
  domain = "${digitalocean_domain.mywebserver.name}"
  type   = "A"
  name   = "mywebserver"
  value  = "${digitalocean_droplet.mywebserver.ipv4_address}"
}
```

Figure 3 : Exemple de provisionnement de serveur avec Terraform

2.4 Infrastructure Mutable vs Infrastructure Immutable

Les **outils de gestion de configuration** tels que **Chef, Puppet, Ansible et SaltStack** sont configurés par défaut sur un paradigme d'infrastructure mutable (qui est sujet au changement).

Par **exemple**, si vous dites à Chef d'**installer une nouvelle version d'OpenSSL**, il lancera la mise à jour du logiciel sur vos serveurs existants. **Au fur et à mesure** que vous appliquez les mises à jour, vos serveurs vont construire un **historique unique des changements**.

Cela conduit souvent à un phénomène connu sous le nom de **dérive de configuration**, où chaque serveur devient légèrement différent de tous les autres, pouvant conduisant à des erreurs de configuration subtiles qui sont difficiles à diagnostiquer et presque impossible à reproduire.

Si vous utilisez un outil d'orchestration tel que **Terraform** pour déployer des images machine créées par Docker, alors chaque « changement » est en fait un déploiement d'un nouveau serveur (tout comme chaque « changement » d'une variable dans la programmation fonctionnelle renvoie en fait une nouvelle variable).

Par **exemple**, pour **déployer une nouvelle version d'OpenSSL**, vous devez **créer une nouvelle image** en utilisant Docker avec la nouvelle version d'OpenSSL déjà installée, **déployer cette image** sur un ensemble de serveurs totalement nouveaux, puis **déployer les anciens serveurs**.

Cette approche **réduit la probabilité de dérive de configuration**, permet de **savoir exactement quel logiciel est exécuté** sur un serveur et vous permet de **déployer de façon triviale n'importe quelle version** précédente du logiciel à tout moment.

Bien sûr, il est possible de forcer les outils de gestion de configuration à faire des déploiements immutables, mais ce n'est pas le fonctionnement natif pour ces outils, alors que c'est une façon naturelle d'utiliser les outils d'orchestration.

Glossaire

Bientôt disponible

Pour aller plus loin

Bientôt disponible

Pour voir la fiche complète et les documents attachés, rendez-vous sur
https://elearning.26academy.com/course/play/5aa2661754470b34d4e474f9