Engineer

Infrastructure immuable

❄ Remplacer les serveurs au lieu de les modifier

Ce que c’est

L’infrastructure immuable est un mode d’exploitation des serveurs où une machine n’est jamais modifiée une fois en service. Le Well-Architected Framework de HashiCorp la décrit comme une infrastructure qui, « une fois déployée, n’est jamais modifiée, seulement remplacée ». Nouvelle version d’application, bibliothèque mise à niveau, correctif de sécurité ou réglage modifié arrivent de la même façon : une nouvelle machine ou un nouveau conteneur, construit à partir d’une nouvelle image, remplace l’ancien. Celui-ci est ensuite arrêté et supprimé.

À l’opposé, l’infrastructure mutable met à jour sur place un serveur conservé longtemps. En juin 2013, Chad Fowler décrivait ce qu’il devient après des années de mises à niveau : corrigé encore et encore, portant des changements de configuration faits à la main en pleine panne et des tâches cron connues d’une seule personne, jusqu’à ce que l’équipe craigne autant de le modifier que de le remplacer. Son remède, nommé déploiements immuables, consistait à construire chaque système par automatisation, puis à ne plus y toucher. Une mise à niveau ou nouvelle version signifie alors construire un serveur ou une image neufs et jeter les anciens.

Deux types d’images rendent l’approche praticable. Une image de machine, selon la documentation Packer, est une unité statique contenant un système d’exploitation préconfiguré et les logiciels installés. Packer, outil communautaire, les construit : à partir d’une seule configuration source, il produit des images correspondantes pour plusieurs plateformes. Les images de conteneurs sont le second type. La documentation Docker est claire : une image créée ne peut plus être modifiée ; on ne peut qu’en créer une autre ou ajouter des changements au-dessus sous forme de nouvelles couches.

Le terme décrit un mode d'exploitation, pas un produit à installer. Il a aussi une frontière : les serveurs applicatifs peuvent être remplacés à chaque version, mais leurs données doivent vivre ailleurs, assez longtemps pour survivre à chaque remplacement. C'est à cette frontière que se concentre l'essentiel de la conception pratique.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que l'infrastructure immuable ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Toute la configuration décrite dans des fichiers, pour la recréer à l'identique sans qu'elle ne dépende de la mémoire d'une seule personne.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c'est important

La première raison est la dérive. Le cadre de HashiCorp la nomme directement : mises à jour manuelles et modifications sur place éloignent peu à peu les serveurs de leur configuration initiale. Deux serveurs partis du même état à un an d'intervalle ne finissent pas forcément identiques : ils ont reçu des séquences différentes de corrections, nouvelles tentatives et changements manuels. Si chaque serveur est une copie neuve d'une image, cette image le décrit entièrement et il ne reste rien à laisser dériver.

La deuxième raison est le retour arrière. Si une version se comporte mal, le cadre propose de redéployer l'artefact précédent, au lieu de réparer les serveurs en service. L'image précédente est un état connu ayant déjà fonctionné en production : sous pression, l'équipe n'a pas à reconstituer de mémoire et avec l'historique du shell l'état du serveur la veille.

La troisième est de limiter les interventions humaines en production. Le cadre rappelle que se connecter aux serveurs pour les mettre à jour manuellement introduit fautes de frappe et étapes oubliées, et qu'en infrastructure immuable personne n'a à s'y connecter à la main. Moins de personnes ont besoin d'un accès à une machine en service ; celles qui en disposent l'utilisent rarement.

La quatrième est la piste d'audit. Les auditeurs, rappelle la même page, demandent des preuves que la production n'a pas changé hors des procédures approuvées, alors que les changements manuels laissent des traces incomplètes. Si chaque changement est un nouvel artefact, le contrôle de versions et les journaux CI/CD montrent ce qui a changé, quand et qui l'a approuvé. Un correctif de sécurité suit le même chemin qu'une nouvelle fonctionnalité : un artefact neuf remplace celui en service, avec la même construction, les mêmes tests et les mêmes traces.

Rien de cela n'est gratuit. Le cadre note qu'une machine virtuelle démarre plus lentement et consomme plus de ressources qu'un conteneur, tandis que les conteneurs ont besoin d'un orchestrateur. Tout changement, même un réglage d'une ligne, impose une construction complète et un remplacement. Le processus de construction doit donc être automatique et assez rapide pour que personne ne soit tenté de le contourner. L'état doit quitter les machines remplaçables avant le premier remplacement, pas après.

À quoi cela ressemble en pratique

Un modèle Packer décrit la construction d'une image de machine. Celui-ci part de l'exemple de la documentation du constructeur Amazon EBS de Packer et ajoute un bloc `build` sous la forme indiquée dans la référence de ce bloc :

```
source "amazon-ebs" "web" {
  access_key    = var.aws_access_key
  secret_key    = var.aws_secret_key
  region        = "us-east-1"
  source_ami    = "ami-fce3c696"
  instance_type = "t2.micro"
  ssh_username  = "ubuntu"
  ami_name      = "web {{timestamp}}"
}

build {
  sources = ["source.amazon-ebs.web"]

  provisioner "shell" {
    scripts = fileset(".", "scripts/{install,secure}.sh")
  }
}
```

Le bloc `source` nomme le constructeur et l'image de base, `source_ami`, dont le volume racine est copié et provisionné sur une instance en cours d'exécution. Le bloc `build` indique quelles sources construire et comment les provisionner. Le provisionneur shell exécute les deux scripts dans cette machine avant d'en faire un artefact statique : une nouvelle image. Les déclarations des deux variables sont omises ici. Une commande exécute toutes les constructions prévues par le modèle et produit les artefacts :

```
packer build web.pkr.hcl
```

Le déploiement est l'autre moitié. Le cadre de HashiCorp rapproche les deux outils : Packer construit les images, Terraform les déploie, puis elles sont remplacées entièrement lorsqu'une mise à jour s'impose. Si une modification sur place est impossible, le comportement de Terraform sans réglage contraire, selon sa référence de cycle de vie, est de détruire l'objet existant puis de créer son remplaçant. Dans ce processus, les mises à jour ne nécessitent pas de se connecter à un serveur de production en service. Un changement des scripts produit une nouvelle image, celle-ci de nouvelles machines, et les anciennes sont retirées.

La même discipline s'applique à un seul hôte. La réalisation sur les conteneurs avec Podman et Quadlet sépare provisionnement et déploiement. Root prépare le chemin de déploiement une seule fois ; ensuite, les versions sont publiées par un utilisateur ordinaire sans privilèges particuliers, autorisé à exécuter exactement un script et rien d'autre. Une version est une image neuve suivie du redémarrage du service, sans shell root ni commandes libres. La réalisation sur CI/CD avec GitHub Actions va plus loin côté images : le frontend de production utilise une base distroless dépourvue de shell et de gestionnaire de paquets. Les workflows, pas une personne, font passer une version construite d'un environnement au suivant. Une telle image ne laisse presque rien à quoi se connecter et aucun outil pour installer quoi que ce soit : construire sa remplaçante est la seule manière raisonnable de la changer.

La réalisation sur les serveurs reconstructibles à partir de zéro illustre soigneusement l'autre modèle. Ansible amène un hôte actif à l'état décrit dans le dépôt ; chaque play est idempotent et le dépôt fournit la configuration permettant de reconstruire l'environnement déclaré. La réalisation expose le coût des changements sur place. Toute modification prend plus de temps qu'une édition directe sur le serveur, et une exécution arrêtée à mi-chemin laisse l'hôte

en pire état qu'une exécution refusant de démarrer. C'est pourquoi un play de vérifications préalables est désormais exécuté en amont. Un hôte à moitié modifié se rapproche du serveur fragile décrit par Fowler ; remplacer la machine est l'autre façon d'éviter ce risque.

Où les problèmes surviennent

Le modèle est simple à énoncer et facile à éroder. La plupart des échecs viennent d'une exception qui semblait inoffensive.

Un correctif urgent par SSH ramène la dérive. Sous pression, quelqu'un se connecte à un serveur en service et modifie un fichier. Le correctif fonctionne jusqu'au remplacement de la machine au déploiement suivant, puis disparaît, avec toute trace de sa nécessité. Jusque-là, le serveur ne correspond plus à son image, précisément ce que l'approche vise à empêcher. Faites passer tout changement, même urgent, par la construction de l'image et gardez-la assez rapide pour être le chemin le plus court. HashiCorp exclut toute connexion à un serveur en service pour des modifications ou mises à jour manuelles.

Les données écrites sur la machine disparaissent avec elle. Docker explique qu'à la suppression d'un conteneur, les changements de fichiers effectués à l'intérieur sont aussi supprimés. Téléversements, fichiers générés et base locale sur une machine remplaçable disparaissent à la prochaine version. Gardez l'état hors de la partie remplacée. Les volumes Docker conservent les données au-delà de la vie d'un conteneur : un nouveau conteneur rattaché au même volume retrouve les fichiers. Pour les machines, un disque séparé ou un service de base de données extérieur à l'image remplit ce rôle.

Le remplacement interrompt le service. Sans réglage contraire, Terraform détruit l'objet avant de créer son remplaçant, laissant un intervalle sans rien en service. Le réglage `create_before_destroy` du bloc `lifecycle` d'une ressource inverse l'ordre :

```
resource "aws_instance" "web" {
  # ...
  lifecycle {
    create_before_destroy = true
  }
}
```

HashiCorp impose une activation explicite de ce réglage parce que deux copies d'un objet ne peuvent pas toujours coexister : beaucoup de types exigent un nom unique ou imposent d'autres limites. La référence demande de connaître les contraintes de chaque type avant de s'y fier.

Une reconstruction démarre discrètement avec une base plus récente. Au lieu d'un `source_ami` fixe, le constructeur Amazon EBS peut le tirer d'un `source_ami_filter`. Avec `most_recent` à true, le filtre choisit l'image correspondante la plus récente, ce que la documentation juge surtout utile pour construire quotidiennement une distribution. C'est commode pour rester à jour, mais deux constructions du même modèle à des jours différents partent alors de bases différentes : reconstruire une ancienne version ne la reproduit plus. Pour les images de versions, fixez `source_ami` à une image précise et faites-le évoluer délibérément dans un commit revu.

Des secrets sont intégrés à l'image. Le cadre de HashiCorp est ferme : les artefacts ne doivent jamais contenir de secrets tels que mots de passe de bases de données, clés API ou clés de chiffrement. Quiconque peut lire un artefact peut en extraire les secrets ; changer une donnée d'authentification force alors la reconstruction et le redéploiement de tous les artefacts. Récupérez les secrets au démarrage de la machine ou du conteneur. Le modèle de HashiCorp utilise Vault : la machine s'authentifie au démarrage et reçoit des identifiants temporaires qui expirent, tandis que leur rotation dans Vault ne reconstruit rien. Le modèle précédent suit la même règle pour ses propres clés, reçues comme variables et non écrites dans le fichier.

L'artefact précédent a disparu au moment nécessaire. Revenir en arrière consiste à le redéployer, à condition qu'il existe encore. Un nettoyage supprimant les anciennes images, ou un nom d'image réutilisé à chaque construction, transforme ce retour en reconstruction d'ancien code sur une base qui a peut-être évolué. Gardez les images des versions récentes ayant fonctionné en production, donnez à chacune un nom jamais réutilisé et réalisez un essai de retour arrière vers l'une d'elles avant le jour où il compte.

Pour aller plus loin

- Créer une infrastructure immuable — Well-Architected Framework sur la dérive, le retour arrière, les types d'images et les secrets.

- Jetez vos serveurs et brûlez votre code : infrastructure immuable et composants jetables — Article de Chad Fowler de juin 2013, lu dans une copie d'Internet Archive.
- Introduction à Packer — Ce que construit Packer et le contenu d'une image de machine.
- Constructeur Amazon EBS — Constructeur de l'exemple, avec `source_ami` et `source_ami_filter` .
- Référence du méta-argument lifecycle — Ordre de Terraform sans réglage contraire, détruire puis créer, et `create_before_destroy` .
- Conserver les données des conteneurs — Pourquoi les fichiers internes disparaissent avec le conteneur et comment les volumes les conservent.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que l'infrastructure immuable ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/immutable-infrastructure/