Engineer

Terraform

outil qui planifie et applique le code d'infrastructure

Ce que c’est

Terraform est l’outil d’infrastructure en tant que code de HashiCorp. Vous décrivez les ressources cloud et sur site dans des fichiers texte, conservez ces fichiers sous gestion de versions, puis Terraform détermine les objets réels à créer, modifier ou détruire pour que l’infrastructure en fonctionnement corresponde aux fichiers. Le programme lui-même s’appelle `terraform`, en minuscules : c’est ainsi que son nom apparaît dans chaque commande et chaque script.

Le langage est déclaratif. La documentation de HashiCorp le décrit comme une manière de « décrire un objectif visé plutôt que les étapes pour l’atteindre ». Une configuration indique qu’une machine d’une certaine taille existe dans un certain réseau ; elle n’énumère pas les appels d’API qui permettraient de la créer. Les fichiers portent l’extension `.tf`, et il existe aussi une variante fondée sur JSON avec l’extension `.tf.json`. La syntaxe native est définie à partir d’une syntaxe appelée HCL et comporte deux éléments principaux : un bloc, qui contient d’autres contenus, et un argument, qui attribue une valeur à un nom.

Trois idées structurent le reste de l’outil.

Les fournisseurs sont des plugins. La configuration indique à Terraform lesquels installer, et `terraform init` les trouve, les télécharge et les installe. Selon HashiCorp, ces fournisseurs permettent à Terraform de travailler avec pratiquement toute plateforme ou tout service disposant d’une API accessible ; ceux qui sont publics figurent dans le Terraform Registry. Un fournisseur définit les types de ressources de sa plateforme : le même programme central peut ainsi gérer une zone DNS, une machine virtuelle et une règle de supervision chez trois prestataires différents.

Les modules sont des répertoires. Un module est un ensemble de fichiers `.tf` et `.tf.json` réunis dans un même répertoire ; le module racine est le répertoire de travail depuis lequel Terraform est lancé.

L'état sert de mémoire. Terraform conserve une trace de l'infrastructure réelle dans un fichier d'état, que son introduction qualifie de source de vérité de l'environnement. Cet état relie chaque ressource de la configuration à l'objet réel qu'elle gère, avec ses métadonnées. Par défaut, il s'agit d'un fichier local nommé `terraform.tfstate`, dont la version précédente est conservée à côté sous le nom `terraform.tfstate.backup`.

La licence a changé. Le 10 août 2023, HashiCorp a annoncé le passage de la licence de son code source, pour toutes les futures versions de ses produits, de Mozilla Public License 2.0 à Business Source License 1.1, tandis que ses API, ses SDK et presque toutes ses autres bibliothèques restent sous MPL 2.0. OpenTofu se présente comme un fork de Terraform, lancé par Gruntwork, Spacelift, Harness, Env0, Scalr et d'autres, et développé sous l'égide de la Linux Foundation. Son dépôt est publié sous MPL 2.0, et son site le présente comme un remplacement de Terraform qui conserve les méthodes de travail et les configurations existantes.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que Terraform ? » 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

HashiCorp décrit la démarche centrale en trois étapes. Écrire : vous définissez les ressources, qui peuvent relever de plusieurs fournisseurs et services. Planifier : Terraform détermine ce qu'il créerait, modifierait ou détruirait et le présente dans un plan d'exécution. Appliquer : une fois le plan approuvé, il effectue ces changements en les ordonnant selon les dépendances entre ressources, qu'il déduit d'un graphe de ressources.

L'étape intermédiaire concentre l'essentiel de la valeur. Le plan est un aperçu qu'une personne autre que l'auteur de la modification peut lire avant toute action. La personne chargée de la revue voit davantage que le diff des fichiers : elle voit les conséquences, deux ressources créées, une ressource mise à jour et, parfois, une ressource détruite que personne ne voulait toucher. Cette dernière ligne coûte peu à repérer dans une pull request, mais très cher à découvrir après coup.

La deuxième raison est l'étendue des plateformes prises en charge. Comme la connaissance de chaque plateforme réside dans son fournisseur, un langage, une démarche et des habitudes de revue communes couvrent tous les fournisseurs utilisés par une équipe. Une entreprise utilisant deux clouds et un hébergeur DNS distinct peut décrire les trois dans un dépôt et les examiner de la même manière. Bicep de Microsoft fait un autre compromis dans sa démarche Azure Resource Manager : il ne conserve pas son propre fichier d'état, car Azure stocke l'état du déploiement. Cela ne fait pas de Bicep un langage réservé à Azure : ses extensions prennent aussi en charge des ressources hors de Resource Manager, notamment Kubernetes et Microsoft Graph.

La troisième raison est l'état. Puisque Terraform enregistre quel objet réel appartient à quel bloc, il peut distinguer un objet qu'il gère d'un objet qu'il n'a jamais vu et comparer sa trace à ce qui fonctionne réellement. Cet enregistrement est aussi le prix de l'outil. Il doit être stocké à un endroit accessible à toute l'équipe, nécessite un verrou pour empêcher deux exécutions de l'écrire en même temps et peut contenir des secrets : son contrôle d'accès doit donc être aussi strict que celui de ces secrets.

D'autres coûts, plus modestes, s'ajoutent. HCL est un langage de plus à apprendre, et chaque fournisseur a ses propres types de ressources et particularités. Les équipes qui construisent un produit sur Terraform ont aussi une licence à lire : la FAQ de HashiCorp consacre une question à ce qui constitue une offre concurrente. OpenTofu propose la même démarche sous MPL 2.0.

À quoi cela ressemble en pratique

La page de HashiCorp consacrée à la syntaxe des ressources prend ce bloc comme exemple :

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

`resource` est le type de bloc et `aws_instance` le type de ressource, défini par le fournisseur AWS. `"web"` est le nom local : Terraform utilise cette étiquette pour suivre la ressource dans son fichier d'état, sans aucun effet sur les réglages de la machine réelle. Les deux lignes à l'intérieur du bloc sont des arguments.

Trois commandes font passer ce fichier du texte à une machine en fonctionnement :

```
terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

`terraform init` prépare le répertoire de travail : il installe les plugins fournisseurs demandés par la configuration et prépare l'emplacement de stockage de l'état. Il peut être exécuté plusieurs fois sans danger, et sa documentation demande de versionner le fichier de verrouillage des dépendances.

`terraform plan -out=tfplan` calcule le plan et l'écrit dans un fichier au format opaque. La planification n'applique pas les changements d'infrastructure proposés. `terraform apply tfplan` effectue ensuite les opérations de ce fichier sans demander de confirmation, car HashiCorp considère que transmettre un plan enregistré vaut approbation. Pour la même raison, l'option `-auto-approve` est ignorée lorsqu'un fichier de plan est fourni.

Dans une équipe, la répartition est naturelle. Une tâche CI calcule le plan sur la pull request et conserve le fichier, une personne lit le plan ainsi que le diff, puis, après la fusion, une seconde tâche applique ce même fichier : ce qui est exécuté est donc ce qui a été lu. La documentation de la commande apply ajoute une condition facile à oublier : s'assurer que personne ne peut modifier l'infrastructure en dehors de la démarche Terraform, sinon le plan décrit un monde qui a déjà changé.

Sur ce site, la page des technologies place Terraform à côté d'Ansible : tous deux écrivent ce que les machines doivent être, pour qu'un environnement puisse se lire plutôt que se reconstituer. Une réalisation applique l'idée avec un autre outil : l'infrastructure Azure a été provisionnée à partir du code avec Bicep. Les environnements de test, de préproduction et de production sont tous construits à partir des mêmes modèles ; comme ceux-ci acceptent des paramètres, ajouter ou modifier un environnement passe par une pull request. Rapprochée de cet article, la réalisation illustre le compromis évoqué plus haut. Bicep n'a pas de fichier d'état propre parce qu'Azure conserve cet état, tandis que Terraform garde le sien et, en contrepartie, fonctionne sur plusieurs clouds.

Où les problèmes surviennent

La plupart des accidents avec Terraform ont la même forme : la configuration est valide, l'exécution réussit et quelque chose de réel a disparu ou changé.

Renommer une ressource la détruit. Remplacer l'étiquette `"web"` par `"frontend"` donne à la ressource une nouvelle adresse. Le guide de refactoring de HashiCorp explique que, par défaut, Terraform interprète ce changement comme l'instruction de détruire l'objet existant et d'en créer un nouveau. Pour une machine jetable, cela coûte quelques minutes ; pour une base de données, cela coûte les données. La solution est un bloc `moved`, qui met à jour l'adresse sans rien détruire :

```
moved {
  from = aws_instance.web
  to   = aws_instance.frontend
}
```

Conservez le bloc après son utilisation. HashiCorp recommande fortement de garder les anciens blocs `moved` et précise qu'en retirer un constitue une rupture de compatibilité.

Deux personnes appliquent des changements avec deux copies d'un état local. Par défaut, l'état de chaque espace de travail est un fichier local. Cela convient à une personne sur un portable. Dès qu'un collègue lance Terraform depuis une autre copie, les deux fichiers d'état ne s'accordent plus sur ce qui existe et chaque exécution agit selon sa propre version. La solution consiste à conserver l'état dans HCP Terraform ou dans un stockage distant avec

verrouillage et contrôle d'accès, comme le recommande HashiCorp. Terraform verrouille alors automatiquement l'état pour toute opération susceptible de l'écrire et ne continue pas s'il ne peut pas obtenir le verrou. Toutes les options de stockage ne prennent pas en charge le verrouillage : vérifiez-le avant de choisir et ne touchez pas à `-lock=false` ; l'option existe, mais la documentation déconseille son utilisation.

Un verrou bloqué est forcé alors que quelqu'un travaille encore. Lorsque le déverrouillage automatique échoue, le verrou reste en place et la commande `force-unlock` le libère avec l'identifiant affiché par Terraform. La documentation du verrouillage appelle à la plus grande prudence : forcer un verrou détenu par quelqu'un d'autre peut laisser deux processus écrire le même état, et la commande ne vise que votre propre verrou après l'échec du déverrouillage automatique. Avant de forcer quoi que ce soit, il faut déterminer à qui appartient le verrou et si l'exécution correspondante est encore active.

Le fichier d'état divulgue des secrets. HashiCorp avertit que des secrets peuvent se retrouver dans l'état et qu'un stockage sans contrôle d'accès sûr peut les exposer. Un fichier d'état dans un bucket lisible par tous, ou copié dans un artefact de build, est une fuite d'identifiants. Gardez-le hors du dépôt, donnez-lui le même contrôle d'accès qu'aux secrets qu'il contient et ne le modifiez jamais à la main ; la commande `terraform state` existe pour les corrections nécessaires.

L'approbation devient une formalité. L'option `-auto-approve` évite l'approbation interactive du plan ; transmettre un plan enregistré l'évite aussi, puisque le fichier vaut approbation. La documentation apply avertit que cela couvre également les changements destructeurs, dont la suppression de ressources. Une tâche CI qui applique à chaque fusion sans plan relu a supprimé le seul moment où une personne aurait pu prévoir la destruction. La solution est la démarche avec plan enregistré décrite plus haut, avec une revue humaine obligatoire entre la tâche de planification et celle d'application. Lorsque `-auto-approve` est utilisé en automatisation, HashiCorp recommande lui-même de calculer d'abord un plan et d'en examiner les changements.

Pour aller plus loin

- Qu'est-ce que Terraform ? — l'introduction de HashiCorp : écriture, planification, application, fournisseurs et état.
- Commande terraform apply — plans enregistrés, `-auto-approve` et exécution automatisée de la commande.

- Verrouillage de l'état — verrous automatiques, stockages sans cette fonction et avertissement sur `force-unlock`.
- Refactoring de modules — pourquoi changer d'adresse signifie détruire et recréer, et le bloc `moved`.
- HashiCorp adopte Business Source License — l'annonce d'août 2023 du passage de MPL 2.0 à BSL 1.1.
- OpenTofu — le fork de Terraform maintenu sous l'égide de la Linux Foundation.

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 Terraform ? » 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/terraform/