Engineer

Infrastructure comme code

Fichiers versionnés décrivant et pilotant l'infrastructure

Ce que c’est

L’infrastructure comme code consiste à décrire les supports d’un système — machines, réseaux, règles de pare-feu, répartiteurs de charge, enregistrements DNS — dans des fichiers texte, à les conserver sous contrôle de versions et à laisser un outil construire l’environnement réel à partir d’eux. Les fichiers sont la copie de référence. Un changement commence dans un fichier, passe la même revue que le code applicatif et n’atteint le système actif que lorsque l’outil l’applique. Le terme anglais s’écrit aussi avec des traits d’union, infrastructure-as-code, et s’abrège IaC.

La publication spéciale 800-204C du NIST, guide DevSecOps pour applications à microservices, décrit ce code comme celui qui provisionne et configure les ressources sur lesquelles une application est déployée, « de façon répétable et cohérente ». Le guide précise que ces définitions sont versionnées avec les mêmes outils que le code source applicatif. La répétabilité est l’objectif : un environnement produit depuis des fichiers peut être reproduit à côté du premier, après un sinistre ou dans un compte de test un mardi après-midi.

Deux grands styles de définition existent. Une définition déclarative indique le résultat final, par exemple un serveur d’une taille donnée dans un réseau donné avec un port ouvert ; l’outil décide quoi créer, modifier ou laisser intact. Une définition impérative énumère des étapes : créer ceci, rattacher cela, puis ouvrir le port. SP 800-204C classe l’infrastructure comme code dans le code déclaratif : l’équipe écrit ses exigences et un outil en fait une instance en service. HashiCorp décrit de même le langage Terraform comme la description d’un objectif souhaité plutôt que des étapes pour l’atteindre. La différence apparaît au second passage : une liste d’étapes décrit comment une opération a été effectuée une fois, une déclaration dit ce qui doit exister. C’est ce que l’outil doit savoir quand une partie existe déjà.

Il n’y a pas de langage unique. SP 800-204C note que, selon l’outil, le code emploie un langage général de script comme Python, JavaScript ou TypeScript, ou un langage de configuration propre, comme HCL de HashiCorp, compatible ou non avec un format standard tel que JSON. Le langage Terraform est du second type et est écrit dans des fichiers dont le nom se termine par `.tf`. Azure accepte les modèles Resource Manager en JSON ou dans le langage Bicep, plus court. Ansible écrit ses playbooks en YAML et se connecte aux machines par SSH avec les identifiants existants de leur système d’exploitation, sans y installer d’agent.

Il est utile de distinguer deux tâches portant ce nom. Le provisionnement crée les pièces : machine virtuelle, disque, réseau, pare-feu. La configuration façonne ce qui tourne dans une machine existante : paquets, comptes, fichiers de service. Les outils penchent vers l’une ou l’autre, avec une frontière souple. Dans les deux cas, la définition doit vivre dans un dépôt et l’outil doit aligner le système réel sur elle pour qu’il s’agisse d’infrastructure comme code.

Les outils diffèrent aussi dans la façon dont ils gardent la trace de ce qu’ils ont construit. Terraform conserve un fichier d’état reliant les ressources configurées aux objets réels qu’il a créés ; sa documentation décrit cet état comme source de vérité de l’environnement. Bicep n’en conserve pas : Azure enregistre lui-même le déploiement. L’emplacement de cette trace décide de ce qui arrive si deux personnes lancent l’outil en même temps ou si elle est perdue.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que l'infrastructure comme code ? » 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

Avant l’infrastructure comme code, chaque environnement était installé puis corrigé à la main. La préproduction recevait une correction absente de production, un port de test n’était jamais refermé et une machine sur trois recevait un paquet plus récent. SP 800-204C appelle cela la dérive : la

configuration présente ne donne plus l'état opérationnel prévu. Le guide nomme son entrée habituelle : un changement dans une console web ou en ligne de commande contournant le code. La dérive explique souvent qu'une version fonctionne en préproduction, mais échoue en production.

L'infrastructure comme code retire la copie de référence du système actif. L'équipe modifie la définition, le système suit. Personne ne se connecte pour changer une règle de pare-feu : on modifie le fichier qui décrit le pare-feu et l'outil exécute le changement. Trois conséquences en découlent.

La première est la revue et l'historique. Un changement réseau devient un diff dans une pull request qu'une seconde personne lit avant toute action. Des mois après, le journal des commits dit qui a ouvert un port et pourquoi, plutôt que la mémoire de quelqu'un. Annuler le changement consiste à revenir sur la modification puis à relancer l'outil.

La deuxième est la répétabilité. Une définition bien écrite est idempotente : elle ne change rien dans un environnement déjà conforme et ramène un environnement ayant dérivé à l'état souhaité. Deux méthodes existent : ajuster ce qui est là ou le jeter et reconstruire depuis la définition. Prendre la seconde comme règle donne l'infrastructure immuable.

La troisième est le faible coût des environnements. SP 800-204C décrit les outils comme automatisant toute la vie de l'infrastructure, du provisionnement au retrait ; certaines ressources vivent trop peu pour mériter un entretien. L'équipe peut créer plusieurs environnements de test à la demande et les détruire ensuite. Le code lui-même peut être vérifié avant toute action : le guide demande une recherche de vulnérabilités dans chaque partie avant sa fusion. Une branche peut recevoir sa copie éphémère de l'agencement de production ; la personne chargée de la revue peut voir un aperçu avant d'approuver.

Les coûts méritent d'être dits. Modifier une règle de pare-feu par pull request est plus lent que dans une console web, ce qui pèse pendant une panne. Chaque outil a son langage et son modèle d'état à apprendre. Un fichier capable de construire un environnement peut aussi le supprimer : le levier porte aussi loin dans les deux sens. L'aperçu compte donc autant que l'exécution suivante.

À quoi cela ressemble en pratique

Voici le parcours d'un changement, de l'édition au système actif, dans une configuration où une tâche CI exécute l'outil et publie sa simulation sur la pull request :

```
Jour 1  09:10  Éditer la définition réseau : autoriser TCP 443 depuis le bureau
        09:12  Ouvrir une pull request ; CI simule l'exécution (le plan)
        09:14  Plan publié dans la pull request : une règle ajoutée, aucune ôtée
        10:30  Un relecteur lit le plan et le diff, puis approuve
        10:31  Fusion ; CI applique le plan revu
        10:33  Second passage de la même définition : aucun changement
Jour 40        Quelqu'un demande pourquoi 443 est ouvert ; le journal des
               commits indique qui et pourquoi
```

Le plan est la partie qui justifie sa place. Le diff montre ce que l'auteur voulait changer ; le plan montre ce que l'outil va faire, y compris ce que l'auteur n'avait pas prévu, comme la destruction d'une ressource. Terraform peut enregistrer un plan dans un fichier avec `-out` ; `terraform apply` recevant ce fichier en exécute les opérations. Ce qui tourne à 10:31 est donc ce qui a été approuvé à 10:30. Le second passage sans changement à 10:33 vérifie que définition et système concordent désormais.

Sur ce site, la page de prestation d'infrastructure comme code décrit le provisionnement reproductible de serveurs à partir d'une configuration versionnée, plutôt que de dépendre de la mémoire de quelqu'un.

Y parvenir de manière sûre est surtout une question d'ordre, comme le montre une réalisation de septembre 2026. Elle ajoute un second serveur chez un autre fournisseur. Le premier avait été configuré à la main puis protégé : pendant un temps, un hôte neuf avec sa politique initiale de mots de passe était ouvert à Internet. Cette fois, il a été créé depuis le dépôt et la protection était là avant la machine. Le play du pare-feu passe avant celui du serveur. Chez le second fournisseur, les étiquettes sont attribuées avant le pare-feu cloud : les règles visant des étiquettes ne peuvent s'y rattacher tant qu'elles n'existent pas. Relancer ne nuit pas, car le rôle de création reprend une machine existante. C'est le seul rôle qui coûte de l'argent : sa documentation indique région, type, caractéristiques et prix mensuel en euros. La réalisation expose aussi les limites d'une simulation. Les deux pare-feu sont configurés par appels aux API REST des

fournisseurs depuis une tâche HTTP générique sans prise en charge du mode de vérification : la simulation la saute. L'essai réel consiste à exécuter le play du pare-feu seul, sans coût et de façon réversible.

Où les problèmes surviennent

Les échecs du code d'infrastructure prennent souvent l'apparence d'une réussite. L'outil termine proprement, CI est au vert, et les fichiers divergent discrètement du réel.

Une correction manuelle pendant un incident est annulée au passage suivant. Quelqu'un ouvre un port dans la console à deux heures du matin, la panne finit et, une semaine après, l'exécution habituelle le referme : les fichiers n'en savaient rien. L'inverse arrive aussi : personne ne relance l'outil et les fichiers cessent de décrire le réel. Reportez la correction d'urgence dans le code le jour même et planifiez une comparaison entre fichiers et système actif. SP 800-204C la demande juste après chaque déploiement puis à intervalles pendant le fonctionnement. Le code de sortie détaillé de la commande plan de Terraform en fait une courte tâche nocturne :

```
terraform plan -detailed-exitcode
# exit 0: no changes, exit 1: error, exit 2: changes present
```

Sur une branche déjà appliquée, un code de sortie 2 indique que le réel s'est éloigné du code ; quelqu'un devrait chercher pourquoi.

Retirer une ressource du fichier ne la supprime pas toujours. Les outils interprètent différemment cette absence. Pour les modèles Azure Resource Manager, Microsoft précise dans sa documentation what-if qu'une ressource existante mais absente du modèle n'est supprimée qu'en mode complet. Dans tout autre mode, elle continue de tourner et de coûter. Apprenez la règle de retrait avant de vous y fier et lisez l'aperçu pour les suppressions attendues autant que redoutées.

La simulation ne voit pas tout. Un aperçu modélise le changement, sans le garantir. La documentation what-if de Microsoft énumère des propriétés annoncées supprimées alors qu'Azure remplit seulement une valeur prédéfinie au déploiement : c'est du bruit. Toute propriété définie par la fonction `reference` apparaît comme changeante à chaque passage. La réalisation

précédente montre l'autre faille : une tâche sans prise en charge du mode de vérification est sautée, donc l'aperçu n'en dit rien. Identifiez les sources de bruit connues de l'outil et traitez ce qu'il ne voit pas comme non testé jusqu'à un essai sur quelque chose de peu coûteux et réversible.

Les secrets finissent dans le dépôt ou l'état. Un mot de passe écrit dans une définition reste dans l'historique de chaque clone. Moins évident : Terraform avertit que l'état peut contenir des secrets et que, sans configuration de stockage contraire, il s'agit du fichier local `terraform.tfstate`. Gardez les secrets dans un magasin spécialisé et référencez-les par nom ; conservez l'état dans un stockage distant avec verrouillage et contrôle d'accès, comme le recommande HashiCorp. Ne l'éditez jamais à la main : la commande `terraform state` existe pour les modifications nécessaires.

Un script se déguise en définition. Un script shell dans le dépôt qui crée un serveur est versionné, mais pas idempotent : deux passages donnent deux serveurs ou une erreur à mi-chemin. Utilisez des ressources déclaratives ou des étapes vérifiant avant d'agir, comme le rôle précédent qui reprend une machine existante. Ansible énonce clairement le critère : si un système est déjà dans l'état décrit par un playbook, le relancer ne change rien.

Pour aller plus loin

- Mise en œuvre de DevSecOps pour une application à microservices avec maillage de services (NIST SP 800-204C) — Infrastructure comme code déclaratif et versionné, dérive et recherche de vulnérabilités.
- Documentation du langage Terraform — Langage déclaratif, ressources et blocs.
- Commande plan — Simulation, plans enregistrés et codes de sortie détaillés.
- État — Ce qu'enregistre Terraform, où et pourquoi c'est sensible.
- Introduction à Ansible — Automatisation sans agent par SSH, playbooks et état souhaité.
- Simulation what-if du déploiement de modèles — Aperçu d'un déploiement Azure, types de changements et bruit connu.

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 comme code ? » 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/infrastructure-as-code/