Engineer

Configuration

les réglages externes au code qui décident de son exécution

Ce que c’est

La configuration est tout ce qu’un programme lit hors de son propre code source pour décider comment fonctionner à un endroit précis : le port sur lequel il écoute, la base à laquelle il se connecte, les comptes qu’il autorise et les fonctions qu’il active. À l’oral, le mot est presque toujours abrégé en config, et les fichiers qui la contiennent sont des fichiers de configuration. NIST, dans SP 800-128, son guide de gestion de configuration axée sur la sécurité, la définit comme « les conditions, paramètres et spécifications possibles permettant de décrire ou d’organiser un système d’information ou un composant de système ». Le mot à retenir est décrire : une configuration est une description, et le système peut y correspondre ou non.

Un programme tire rarement sa configuration d’un seul endroit. Des valeurs prédéfinies sont compilées dans le binaire, un fichier principal se trouve sur disque avec à côté un répertoire de fragments complémentaires, et des variables d’environnement et options de ligne de commande arrivent au démarrage. Le programme les fusionne en un ensemble de valeurs, selon une règle qui tranche lorsque deux sources définissent la même clé, et chaque programme a sa propre règle. Le serveur OpenSSH conserve la première valeur lue pour la plupart des mots-clés, alors que d’autres outils laissent un fichier ultérieur remplacer un précédent. Les mêmes fichiers lus dans un ordre différent produisent un système différent.

Les formats varient : lignes de type INI, TOML, YAML, JSON et formats propres à un programme. TOML 1.0.0 vise à être « un format minimal de fichier de configuration », tandis que YAML 1.2.2 est un langage de données général utilisé pour les fichiers de configuration. Chacun a ses règles pour transformer du texte en valeurs.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la configuration ? » 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 configuration permet à une même compilation de servir dans de nombreux endroits. Le même programme tourne sur un portable, un serveur de préproduction et en production, et sa configuration fait la différence. La méthode Twelve-Factor App, destinée aux services déployables, trace la frontière à cet endroit : la configuration est « tout ce qui est susceptible de varier entre les déploiements », par exemple les références aux services dont dépend l’application, les identifiants des services externes et le nom d’hôte de chaque déploiement. Elle exige une séparation stricte entre configuration et code et propose un test utile partout : savoir si « la base de code pourrait être publiée en open source à tout moment, sans compromettre aucun identifiant ».

Le même texte marque la limite opposée. Le câblage interne qui reste identique à chaque déploiement, comme la table de routes d’un framework web, sort de cette définition et se fait « de préférence dans le code ». Une valeur identique à chaque déploiement est du code déguisé en réglage, et un fichier ne lui donne qu’un autre endroit où être faux.

La configuration est aussi l’endroit où un système énonce sa sécurité : qui peut se connecter, quels ports répondent et ce qu’un pare-feu laisse passer. Un serveur n’est durci qu’à la hauteur des valeurs qu’il utilise réellement, et entre un fichier et ces valeurs se trouvent le parseur, les règles de priorité, les valeurs prédéfinies compilées, la version du programme et l’environnement. Chacun peut changer le résultat sans modifier le fichier. L’habitude fiable consiste donc à demander au programme ce qu’il utilise et à traiter cette réponse comme sa configuration.

À quoi cela ressemble en pratique

Ce site se configure dans un fichier TOML, hugo.toml, qui définit entre autres ses langues, taxonomies et montages. Une table de hugo.toml montre ce que quelques lignes peuvent décider :

```
[frontmatter]
  date = ["date", ":git", ":fileModTime"]
  lastmod = ["lastmod", ":git", ":fileModTime"]
```

Chaque liste donne un ordre de recherche : une date écrite dans la page, puis la date du dernier commit ayant touché le fichier, puis la date de modification du fichier. Une option près du début du fichier, `enableGitInfo = true` , permet à Hugo de lire l'historique Git. Le commentaire au-dessus de la table explique la première entrée : il faut citer explicitement dans la liste le nom du champ des métadonnées initiales, sinon un `lastmod` écrit dans une page est silencieusement ignoré alors que le site se construit normalement.

Cette table est la correction décrite dans la réparation d'un sitemap dont la plupart des pages partageaient un horodatage. Le sitemap avait indiqué aux moteurs de recherche que 172 des 176 pages du site avaient été modifiées pour la dernière fois au même instant : avec la date de modification du fichier comme source, chaque export du contenu, qui réécrit tous les fichiers, renouvelait tous leurs horodatages à la fois. La chaîne de repli a déplacé la source vers l'historique des versions. Au stade d'implémentation documenté, une nouvelle synchronisation après un mois de changements de contenu a touché huit fichiers sur 186.

Sur un serveur, la même leçon apparaît dans le durcissement de SSH avec trois étapes de validation. Un rôle d'automatisation écrivait depuis des mois la configuration du démon SSH lorsqu'un rapport d'accès, en septembre 2026, a lu les réglages résolus du démon et en a trouvé deux qui différaient du fichier du rôle. Les réglages fournis par le système d'exploitation précédaient le fragment complémentaire et, pour ces deux réglages, la première occurrence l'emporte. La correction a donc été un préfixe de nom de fichier qui se classe avant celui du fournisseur. Autour de cette correction, le rôle valide désormais en trois étapes : contrôle de syntaxe du fichier candidat avant installation, contrôle de la configuration assemblée après, puis relecture de la sortie résolue avec des assertions sur 24 directives. Deux directives ont été écartées délibérément, car le démon ne les implémente plus.

Trois protections SSH contre la force brute qui n'ont jamais fonctionné montre l'échec sous l'autre angle. En août 2026, une revue a comparé la configuration de chaque protection au serveur en fonctionnement. Les trois étaient écrites, les trois figuraient dans chaque rapport et aucune n'avait rempli son rôle depuis l'installation du serveur. La configuration de la jail nommait encore le port 22, valeur abandonnée quand le durcissement initial avait déplacé le démon sur le port 1986. La ligne du limiteur de débit était correcte, mais l'ordre des règles du pare-feu l'annulait : une règle plus large placée avant interceptait le même trafic en premier. La configuration de l'action de bannissement appelait un binaire de filtrage de paquets que le gestionnaire de paquets recommande sans l'exiger, et sur cet hôte ce programme n'était jamais trouvé. Deux règles sont sorties de la revue : chaque contrôle de sécurité s'appuie sur une assertion qui le teste, jamais sur un commentaire qui le décrit, et un pare-feu se vérifie par la position de chaque règle, pas par sa seule présence dans la liste.

Où les problèmes surviennent

Chaque échec ci-dessous vient d'un fichier correct sur le papier mais incorrect dans ses effets ; chacun a un contrôle qui observe le comportement.

- La mauvaise source l'emporte. Le manuel de configuration du serveur OpenSSH dit : « Sauf indication contraire, la première valeur obtenue sera utilisée pour chaque mot-clé. » Une valeur du fournisseur lue avant le fragment complémentaire l'emporte donc sur lui, et un outil où la dernière valeur gagne inverse l'effet de la même disposition. Lisez la règle de priorité dans le manuel du programme, puis interrogez-le : `sshd -t` vérifie que la configuration est valide, et `sshd -T` affiche la configuration effective.
- Une clé TOML se retrouve dans la mauvaise table. Dans TOML 1.0.0, les paires sous un en-tête `[table]` lui appartiennent « jusqu'à l'en-tête suivant ou la fin du fichier », et l'indentation est ignorée. Une clé ajoutée en bas du fichier rejoint la dernière table même si elle est alignée à gauche. Gardez les tables à la fin et les clés simples au-dessus. Définir deux fois une clé ou une table est invalide : un doublon est une erreur à corriger, jamais un moyen de remplacer une valeur.
- Une valeur YAML sans guillemets change de type. Avec le schéma de base, recommandé par YAML 1.2.2 en l'absence de choix explicite, une valeur non citée qui correspond à l'un de ses motifs devient un nombre, un booléen ou null : `1.10` est le flottant 1,1, `0o17` l'entier 15, `0123` devient 123, `FALSE` est un booléen et `~` vaut null. Mettez entre guillemets chaque valeur qui doit rester du texte, en commençant par les chaînes de version et les identifiants à zéros initiaux. La spécification compte aussi les clés de mapping répétées parmi les causes de chargement incomplet : une clé écrite deux fois est donc un défaut.

- Un réglage paraît actif et ne fait rien. Une règle masquée par une précédente, une directive supprimée dans une nouvelle version, un port quitté depuis par le service : chacun reste dans le fichier et semble délibéré. La jail sur le port 22 et le limiteur sous une règle plus large paraissaient tous deux sains dans les rapports. Un contrôle qui lit chaque fichier isolément les accepte ; vérifiez donc l'ordre et la présence sur le système en fonctionnement.
- Un secret voyage avec le fichier. Un mot de passe dans un fichier de configuration va partout où ce fichier va : dépôt, sauvegardes, rapport de bug. Twelve-Factor garde la configuration dans des variables d'environnement notamment parce que, contrairement aux fichiers de configuration, « elles risquent peu d'être ajoutées accidentellement au dépôt de code ».

Pour aller plus loin

- Glossaire NIST CSRC : configuration — chaque définition du terme dans les publications de NIST, avec son document d'origine.
- TOML v1.0.0 — la spécification complète, assez courte pour se lire d'une traite.
- YAML Ain't Markup Language, version 1.2.2 — la spécification ; le chapitre 10 contient le schéma de base qui détermine ce que devient une valeur sans guillemets.
- The Twelve-Factor App : configuration — les raisons de garder les valeurs propres au déploiement hors du code, avec le test de publication en open source.
- sshd_config(5) — le manuel de configuration du serveur OpenSSH, sa règle de la première valeur, Include et Match.
- sshd(8) — le manuel du démon ; `-t` vérifie la configuration et `-T` affiche la configuration effective.



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 la configuration ? » 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/configuration/