Engineer

Paquet logiciel

Code et manifeste regroupés pour être installés

Ce que c’est

Un paquet logiciel est la forme sous laquelle un logiciel passe des personnes qui l’écrivent aux machines qui l’exécutent : le code, sous forme d’archive ou de répertoire, accompagné d’un manifeste donnant son nom, sa version et ce dont il a besoin pour fonctionner. Au pluriel, on parle de paquets, et la documentation système emploie souvent le terme paquet logiciel. Chaque écosystème en fixe les limites à sa façon. Pour npm, un paquet est un fichier ou un répertoire décrit par un fichier package.json. Sa documentation le distingue d’un module, c’est-à-dire de tout ce que Node.js peut charger avec require() ou import ; seuls les modules munis d’un package.json sont des paquets. Python emploie le mot dans deux sens, que le Python Packaging User Guide distingue : un paquet d’importation est un répertoire de modules que le code importe, tandis qu’un paquet de distribution est l’archive versionnée que l’utilisateur télécharge et installe. En Rust, un paquet rassemble une ou plusieurs crates avec un Cargo.toml indiquant comment les construire. Sur Debian, l’unité est le paquet binaire, le fichier .deb que le système installe et dont il conserve la trace.

Un paquet est une unité de distribution, plus facile à définir en le comparant aux notions voisines. Une bibliothèque est du code ; un paquet est la façon de le distribuer, et une même bibliothèque peut parvenir aux utilisateurs sous forme de paquet npm, de wheel Python et de paquet Debian en même temps. Un gestionnaire de paquets est le programme qui lit les manifestes, récupère ce qu’ils indiquent et installe le tout dans un ordre qui rend l’ensemble utilisable : npm, pip, Cargo ou apt. Un registre ou une archive est l’endroit où les paquets attendent d’être récupérés, comme le registre npm, PyPI, crates.io ou un miroir Debian. Les gestionnaires de paquets propres à un langage installent du code pour un projet, généralement dans un répertoire à côté de ses fichiers. Les gestionnaires de paquets système installent des logiciels pour toute la machine et enregistrent quel paquet a placé chaque fichier à quel endroit.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un paquet logiciel ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Définir d'abord le système sur le papier pour que ses éléments s'articulent et que son coût soit connu avant d'écrire du code.

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. Un plan d'infrastructure pour 14 départements — 14 départements de recherche sur un même plan d'infrastructure

Un plan d'infrastructure pour 14 départements Contactez-nous →

Pourquoi c'est important

Le code tiers arrive dans un projet de trois façons : par un gestionnaire de paquets, par copie dans le dépôt avec sa licence et une somme de contrôle, ce qu'on appelle le vendoring, ou par construction à partir des sources. Le gestionnaire de paquets est le choix habituel pour de bonnes raisons. Il résout tout l'arbre de dépendances, les met à jour en une commande et enregistre ce qu'il a installé. Le coût est la confiance : le projet dépend désormais de chaque manifeste de cet arbre et de chaque registre d'où il provient. Le vendoring remplace cette confiance par du travail, puisque le code copié ne change que lorsque quelqu'un en copie volontairement une nouvelle version dans le dépôt. Construire à partir des sources donne le plus de maîtrise et demande le plus d'entretien.

C'est dans le manifeste qu'apparaissent la plupart des compromis. Il indique généralement une plage de versions plutôt qu'une seule ; dans npm, une plage à accent circonflexe telle que ^10.0.1 accepte toute version ultérieure ayant le même numéro de version majeure. Les plages permettent de recevoir des correctifs sans modifier le fichier, mais deux installations à une semaine d'intervalle peuvent alors produire des arbres différents. Un fichier de verrouillage comble cet écart. npm écrit package-lock.json pour décrire l'arbre exact qu'il a construit afin que les installations suivantes le reproduisent ; Cargo écrit Cargo.lock avec la version exacte de chaque dépendance, et tous deux attendent que le fichier soit enregistré dans le dépôt. Le manifeste indique ce qu'un projet est prêt à accepter ; le fichier de verrouillage indique ce qu'il a reçu.

Les manifestes disent aussi à quel point un paquet a besoin d'un autre. Le Debian Policy Manual définit une gradation. Depends est absolu : le paquet n'est pas configuré tant que toutes ses dépendances ne l'ont pas été. Recommends exprime une exigence forte mais non absolue pour les paquets qui accompagnent celui-ci dans toutes les installations, sauf les cas inhabituels. Suggests désigne des paquets qui le rendent plus utile, mais qu'il est

parfaitement raisonnable de ne pas installer. Une installation qui omet un paquet recommandé est valide, et le logiciel qu'elle contient peut pourtant en avoir besoin pour l'une de ses tâches.

Tout outil qui raisonne sur les paquets se fie à leurs métadonnées. Un vérificateur de licences lit la licence déclarée, souvent une expression SPDX comme MIT OR Apache-2.0, où OR offre un choix et AND impose toutes les licences à la fois. Un scanner de vulnérabilités rapproche les noms et versions déclarés des avis publiés. Si une déclaration est erronée, chaque outil qui la lit se trompe de la même façon : les fichiers de licence contenus dans un paquet distribué méritent donc d'être lus séparément.

À quoi cela ressemble en pratique

Une revue de durcissement d'un serveur de production a constaté que son mécanisme de bannissement d'intrusions, ou jail, n'avait jamais banni personne, et que l'une des trois causes distinctes relevées tenait à une relation entre paquets. L'action de bannissement appelait un programme de filtrage de paquets que les métadonnées du paquet déclarent comme recommandé plutôt qu'obligatoire, et le serveur avait été construit sans lui. Le jail démarrait, comptait les échecs de connexion et prenait sa décision à chaque fois, puis échouait à la dernière étape, celle du bannissement effectif. La revue de trois protections SSH l'a découvert en inspectant le système en cours d'exécution plutôt que les fichiers de configuration, et en a tiré une règle : chaque contrôle de sécurité est étayé par une assertion automatisée, pas par un commentaire.

Un cœur Rust lié à une application distribuée fait entrer tout son graphe de dépendances dans le produit : ici, 644 crates, chacune avec sa propre déclaration de licence. Une dépendance située plusieurs niveaux plus bas peut passer à une licence copyleft d'une version à la suivante, avec pour seule trace une ligne modifiée dans un fichier de verrouillage que peu de gens lisent. L'audit des licences de 644 crates s'exécute à chaque construction : il évalue l'expression de licence de chaque crate selon une liste d'autorisations, accepte un choix de licences si l'une d'elles est autorisée, et une combinaison seulement si elles le sont toutes, puis arrête la construction pour tout cas sans correspondance. Pour montrer que le contrôle pouvait se déclencher, deux échecs ont été provoqués volontairement : l'un en réécrivant l'expression de licence d'une crate, l'autre en déplaçant la révision épinglée du cœur sans régénérer l'audit. La page énonce clairement sa limite : l'audit lit des métadonnées déclarées, qui peuvent être erronées, et ne constitue pas un conseil juridique.

La forge git de l’entreprise a été installée depuis la source de paquets de son propre fournisseur sous la forme d’un programme unique que le système d’exploitation maintient en fonctionnement. Ajouter le dépôt d’un fournisseur à un serveur est une décision à part entière : dès lors, les mises à jour courantes installent tout ce que ce fournisseur publie et signe.

Ce site lui-même n’a pas besoin de Node pour être servi. Son package.json est marqué private, ce qui demande à npm de refuser sa publication, ne déclare que des outils de développement, avec des plages de versions à accent circonflexe, et se trouve à côté d’un package-lock.json enregistré dans le dépôt. Son pyproject.toml configure un outil d’analyse de code, un correcteur orthographique et un vérificateur de types, et n’a pas de table [project] ou [build-system], puisqu’il ne décrit aucun paquet à construire. Les icônes sont intégrées au dépôt, copiées avec la licence amont et une somme de contrôle SHA-256 pour chaque fichier.

Où les problèmes surviennent

Une fonctionnalité se dit opérationnelle et ne fait rien

Un service démarre, écrit dans les journaux qu’il fonctionne et saute discrètement l’une de ses tâches à chaque fois. Un programme qu’il appelle appartient à un paquet recommandé ou suggéré, que l’installation était en droit d’omettre. Quand une fonctionnalité dépend d’un programme externe, vérifiez sa présence au démarrage et interrompez le démarrage en signalant une erreur claire s’il manque. Déclarez-le comme une exigence obligatoire dans votre propre déploiement, puis testez la fonctionnalité de bout en bout sur un serveur préparé comme celui de production.

Deux machines installent un code différent depuis le même manifeste

La construction réussit sur un ordinateur portable et échoue en intégration continue, ou l’inverse, sans changement du code propre au projet. Le manifeste indique des plages de versions, le fichier de verrouillage n’a jamais été enregistré dans le dépôt, et une dépendance a publié une version entre les deux installations. Enregistrez le fichier de verrouillage dans le dépôt, effectuez les installations automatisées à partir de ce fichier avec des commandes qui refusent de le modifier, telles que npm ci ou cargo build –locked, et mettez-le à jour dans un changement distinct, soumis à revue.

Une licence change dans une dépendance connue

Un produit qui avait passé une revue de licences l'an dernier distribue maintenant du code sous des conditions que personne n'a acceptées, alors que sa liste de dépendances paraît identique. Une dépendance existante a changé de licence entre deux versions, et une mise à jour courante a introduit la nouvelle version par le fichier de verrouillage. Vérifiez les licences déclarées selon une liste explicite d'autorisations à chaque construction, afin que le changement arrête la construction le jour de son arrivée.

Pour aller plus loin

- Documentation npm : paquets et modules — la définition d'un paquet par npm et la raison pour laquelle un module n'en est pas toujours un.
- Debian Policy Manual : déclarer les relations entre paquets — Depends, Recommends, Suggests et les autres champs, ainsi que leur traitement par un gestionnaire de paquets.
- Python Packaging User Guide : écrire votre pyproject.toml — les trois tables de pyproject.toml et les métadonnées déclarées par un paquet de distribution Python.
- The Cargo Book : Cargo.toml et Cargo.lock — pourquoi le manifeste décrit ce qu'un projet accepte, le fichier de verrouillage enregistre les versions exactes, et quand l'enregistrer dans le dépôt.
- Spécification SPDX 2.3.1 : expressions de licence — la syntaxe employée par les paquets pour déclarer une licence, un choix parmi plusieurs ou un ensemble applicable conjointement.



Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un paquet logiciel ? » 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/package/