Engineer

Softwarepaket

Gebündelter Code mit Beschreibung, bereit zur Installation

Was es ist

Ein Softwarepaket ist die Form, in der Software von ihren Autoren zu den Maschinen gelangt, auf denen sie läuft: Code als Archiv oder Verzeichnis, zusammen mit einem Manifest, das Name, Version und die zum Betrieb benötigten weiteren Bestandteile nennt. Die Mehrzahl lautet Softwarepakete; Systemdokumentationen sprechen oft ausdrücklich von Softwarepaketen. Jedes Ökosystem zieht die Grenzen anders. Für npm ist ein Paket eine Datei oder ein Verzeichnis, das durch eine package.json-Datei beschrieben wird. Die npm-Dokumentation unterscheidet es von einem Modul: allem, was Node.js mit require() oder import laden kann. Nur Module mit einer package.json sind Pakete. Python verwendet das Wort für zwei Dinge, die der Python Packaging User Guide auseinanderhält: Ein Importpaket ist ein Verzeichnis importierbarer Module. Ein Distributionspaket ist dagegen das versionierte Archiv, das ein Nutzer herunterlädt und installiert. In Rust bündelt ein Paket eine oder mehrere Crates mit einer Cargo.toml, die beschreibt, wie sie gebaut werden. Auf Debian ist die Einheit das Binärpaket, die vom System installierte und verwaltete .deb-Datei.

Ein Paket ist eine Verteilungseinheit und lässt sich am einfachsten von seinen Nachbarn abgrenzen. Eine Bibliothek ist Code; ein Paket ist die Form seiner Auslieferung. Eine Bibliothek kann Nutzer gleichzeitig als npm-Paket, Python-Wheel und Debian-Paket erreichen. Ein Paketmanager ist das Programm, das Manifeste liest, die darin aufgeführten Bestandteile abruft und alles in einer funktionsfähigen Reihenfolge installiert: npm, pip, Cargo und apt. Eine Registry oder ein Archiv hält Pakete zum Abruf bereit, etwa die npm-Registry, PyPI, crates.io oder ein Debian-Spiegelserver. Sprachbezogene Paketmanager installieren Code für ein einzelnes Projekt, meist in einem benachbarten Verzeichnis. Systempaketmanager installieren Software für die gesamte Maschine und halten fest, welches Paket jede Datei an welchen Ort gebracht hat.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Softwarepaket?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Struktur eines Systems wird zuerst auf dem Papier durchdacht, damit die Teile zusammenpassen und die Kosten feststehen, bevor die erste Zeile Code entsteht.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

Warum es wichtig ist

Fremder Code gelangt auf einem von drei Wegen in ein Projekt: über einen Paketmanager, als zusammen mit Lizenz und Prüfsumme ins Repository kopierter Code, sogenanntes Vendoring, oder durch einen Build aus dem Quellcode. Ein Paketmanager ist aus gutem Grund der übliche Weg. Er löst den gesamten Abhängigkeitsbaum auf, aktualisiert ihn mit einem Befehl und protokolliert, was er installiert hat. Der Preis ist Vertrauen: Das Projekt verlässt sich jetzt auf jedes Manifest in diesem Baum und jede zugehörige Registry. Vendoring tauscht dieses Vertrauen gegen Arbeit, denn der kopierte Code ändert sich nur, wenn jemand bewusst eine neue Version hineinkopiert. Selbst aus dem Quellcode zu bauen bietet die größte Kontrolle und verlangt die meiste Pflege.

Im Manifest zeigen sich die meisten Abwägungen. Es nennt gewöhnlich einen Versionsbereich statt einer einzelnen Version. Bei npm akzeptiert ein Caret-Bereich wie ^10.0.1 jede spätere Veröffentlichung mit derselben Hauptversion. Solche Bereiche lassen Korrekturen eintreffen, ohne dass jemand die Datei bearbeitet. Sie bedeuten aber auch, dass zwei Installationen im Abstand einer Woche unterschiedliche Bäume ergeben können. Eine Lockdatei schließt diese Lücke. npm schreibt mit package-lock.json den exakt aufgebauten Baum fest, damit spätere Installationen ihn reproduzieren. Cargo schreibt Cargo.lock mit der genauen Version jeder Abhängigkeit. Beide erwarten, dass die Datei eingecheckt wird. Das Manifest hält fest, was ein Projekt akzeptiert; die Lockdatei, was es erhalten hat.

Manifeste sagen außerdem, wie dringend ein Paket ein anderes braucht. Das Debian Policy Manual definiert dafür eine Abstufung. Depends ist zwingend: Das Paket wird erst konfiguriert, wenn alles, wovon es abhängt, konfiguriert wurde. Recommends ist eine starke, aber keine zwingende Beziehung für Pakete, die

außer in ungewöhnlichen Fällen zu jeder Installation dazugehören. Suggests nennt Pakete, die es nützlicher machen, wobei eine Installation ohne sie völlig vernünftig ist. Eine Installation, die ein empfohlenes Paket auslässt, ist gültig. Die darin enthaltene Software kann dieses Paket trotzdem für eine ihrer Aufgaben brauchen.

Jedes Werkzeug, das Pakete beurteilt, vertraut ihren Metadaten. Eine Lizenzprüfung liest die deklarierte Lizenz, häufig einen SPDX-Ausdruck wie MIT OR Apache-2.0. OR bietet dabei eine Wahl, AND verlangt alle Lizenzen zugleich. Ein Schwachstellenscanner gleicht deklarierte Namen und Versionen mit veröffentlichten Sicherheitshinweisen ab. Ist eine Angabe falsch, liegen alle sie lesenden Werkzeuge auf dieselbe Weise falsch. Deshalb verdienen die Lizenzdateien innerhalb eines ausgelieferten Pakets eine eigene Prüfung.

Wie es in der Praxis aussieht

Die Härtungsprüfung eines Produktionsservers fand heraus, dass dessen Jail zum Sperren von Angreifern noch nie jemanden gesperrt hatte. Eine von drei getrennten Ursachen war eine Paketbeziehung. Die Sperraktion rief ein Paketfilterprogramm auf, das die Paketierung als empfohlen statt zwingend erforderlich aufführt. Der Server war ohne dieses Programm aufgebaut worden. Das Jail startete, zählte fehlgeschlagene Anmeldungen und traf jedes Mal seine Entscheidung. Erst beim letzten Schritt, dem Durchsetzen der Sperre, scheiterte es. Die Prüfung dreier SSH-Schutzmaßnahmen fand dies durch Untersuchung des laufenden Systems statt der Konfigurationsdateien. Sie hinterließ die Regel, jede Sicherheitskontrolle durch eine automatisierte Prüfung abzusichern, nicht durch einen Kommentar.

Ein in eine ausgelieferte Anwendung eingebundener Rust-Kern bringt seinen gesamten Abhängigkeitsgraphen mit ins Produkt: hier 644 Crates, jeweils mit eigener Lizenzdeklaration. Eine mehrere Ebenen tiefer liegende Abhängigkeit kann von einer Version zur nächsten zu einer Copyleft-Lizenz wechseln. Die einzige Spur wäre eine geänderte Zeile in einer Lockdatei, die nur wenige lesen. Das Lizenzaudit der 644 Crates läuft bei jedem Build. Es prüft den Lizenzausdruck jeder Crate gegen eine Positivliste, akzeptiert eine Lizenzauswahl, wenn irgendeine der Lizenzen erlaubt ist, eine Kombination dagegen nur, wenn alle erlaubt sind. Jeder nicht zugeordnete Ausdruck stoppt den Build. Um die Wirksamkeit der Prüfung zu zeigen, wurde sie zweimal gezielt zum Scheitern gebracht: einmal durch Umschreiben des Lizenzausdrucks einer Crate, einmal

durch Verschieben der festgeschriebenen Kernrevision ohne Neuerzeugung des Audits. Die Seite nennt die Grenze ausdrücklich: Das Audit liest deklarierte Metadaten, die falsch sein können, und ist keine Rechtsberatung.

Die eigene Git-Forge des Unternehmens wurde aus der Paketquelle ihres Herstellers installiert, als einzelnes Programm, das vom Betriebssystem am Laufen gehalten wird. Das Repository eines Herstellers auf einem Server hinzuzufügen ist eine eigenständige Entscheidung. Von diesem Zeitpunkt an installieren Routineaktualisierungen alles, was dieser Hersteller veröffentlicht und signiert.

Diese Website selbst benötigt im Betrieb kein Node. Ihre package.json ist als private gekennzeichnet, wodurch npm die Veröffentlichung verweigert. Sie enthält ausschließlich Entwicklungswerkzeuge mit Caret-Versionsbereichen und liegt neben einer eingecheckten package-lock.json. Die pyproject.toml konfiguriert einen Linter, eine Rechtschreibprüfung und eine Typprüfung. Sie hat keine Tabelle [project] oder [build-system], weil sie kein zu bauendes Paket beschreibt. Die Icons werden per Vendoring eingebunden: mit der ursprünglichen Lizenz und einer SHA-256-Prüfsumme pro Datei hineinkopiert.

Wo es schiefgeht

Eine Funktion meldet einen fehlerfreien Zustand und tut nichts

Ein Dienst startet, protokolliert seinen laufenden Zustand und überspringt bei jedem Durchgang still eine seiner Aufgaben. Ein von ihm aufgerufenes Programm gehört zu einem empfohlenen oder vorgeschlagenen Paket, das die Installation weglassen durfte. Wenn eine Funktion ein externes Programm benötigt, prüfen Sie dessen Vorhandensein beim Start und brechen Sie bei Fehlen mit einer klaren Fehlermeldung ab. Deklarieren Sie es in Ihrer eigenen Bereitstellung als zwingende Voraussetzung und testen Sie die Funktion durchgängig auf einem wie das Produktionssystem aufgebauten Server.

Zwei Maschinen installieren aus demselben Manifest unterschiedlichen Code

Der Build läuft auf dem Laptop erfolgreich durch und scheitert in der kontinuierlichen Integration oder umgekehrt, ohne Änderung am eigenen Projektcode. Das Manifest nennt Versionsbereiche, die Lockdatei wurde nie eingecheckt, und zwischen beiden Installationen erschien eine neue Version

einer Abhängigkeit. Checken Sie die Lockdatei ein. Installieren Sie in der Automatisierung daraus mit Befehlen, die sie nicht ändern dürfen, etwa npm ci oder cargo build –locked. Aktualisieren Sie sie als gesonderte, geprüfte Änderung.

Eine vertraute Abhängigkeit wechselt ihre Lizenz

Ein Produkt, das im letzten Jahr eine Lizenzprüfung bestand, liefert nun Code zu Bedingungen aus, denen niemand zugestimmt hat, obwohl seine Abhängigkeitsliste gleich aussieht. Eine bestehende Abhängigkeit wechselte zwischen Versionen ihre Lizenz, und eine Routineaktualisierung zog die neue Version über die Lockdatei ein. Prüfen Sie die deklarierten Lizenzen bei jedem Build gegen eine ausdrückliche Positivliste, damit der Wechsel den Build an dem Tag stoppt, an dem er eintrifft.

Weiterführende Lektüre

- npm-Dokumentation: Pakete und Module — npms eigene Definition eines Pakets und weshalb ein Modul nicht immer eines ist.
- Debian Policy Manual: Beziehungen zwischen Paketen deklarieren — Depends, Recommends, Suggests und die anderen Felder sowie der Umgang eines Paketmanagers mit jedem davon.
- Python Packaging User Guide: Die pyproject.toml schreiben — die drei Tabellen der pyproject.toml und die von einem Python-Distributionspaket deklarierten Metadaten.
- The Cargo Book: Cargo.toml und Cargo.lock — weshalb das Manifest beschreibt, was ein Projekt akzeptiert, die Lockdatei exakte Versionen festhält und wann sie eingecheckt werden sollte.
- SPDX-Spezifikation 2.3.1: Lizenzausdrücke — die Syntax, mit der Pakete eine Lizenz, eine Auswahl mehrerer oder eine gemeinsam geltende Kombination deklarieren.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Softwarepaket?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/package/