Engineer

Programpaket

kod paketerad med en beskrivning, klar att installera

Vad det är

Ett programpaket är den form som programvara färdas i från dem som skriver den till de maskiner som kör den: koden, som ett arkiv eller en katalog, tillsammans med ett manifest som anger namnet, versionen och vad mer som behövs för att den ska fungera. I vardagligt tal säger de flesta bara paket. Varje ekosystem drar gränserna på sitt eget ställe. För npm är ett paket en fil eller katalog som beskrivs av en package.json-fil, och dokumentationen för npm skiljer det från en modul, vilket är allt som Node.js kan läsa in med require() eller import; bara moduler som har en package.json är paket. Python använder ordet för två saker, och Python Packaging User Guide håller isär dem: ett importpaket är en katalog med moduler som koden importerar, medan ett distributionspaket är det versionerade arkiv som en användare laddar ned och installerar. I Rust samlar ett paket en eller flera crates med en Cargo.toml som talar om hur de ska byggas. På Debian är enheten det binära paketet, den .deb-fil som systemet installerar och håller reda på.

Ett paket är en enhet för distribution, och det är lättast att definiera mot sina grannar. Ett bibliotek är kod; ett paket är sättet som koden levereras på, och ett och samma bibliotek kan nå användarna som ett npm-paket, ett Python-wheel och ett Debianpaket samtidigt. En pakethanterare är programmet som läser manifesten, hämtar det de listar och installerar alltihop i en ordning som fungerar: npm, pip, Cargo och apt. Ett paketregister eller paketförråd är platsen där paketen väntar på att hämtas, till exempel npm-registret, PyPI, crates.io eller en Debianspegel. Pakethanterare för ett programspråk installerar kod för ett enda projekt, oftast i en katalog bredvid det. Pakethanterare för systemet installerar programvara för hela maskinen och noterar vilket paket som lade varje fil var.

Använd begreppet i ett beslut

Använd ”Vad är ett programpaket?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Systemets form utarbetad på papper först, så att delarna passar ihop och kostnaden är känd innan någon skriver kod.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. En infrastrukturplan för 14 avdelningar — 14 forskningsavdelningar under en och samma infrastrukturplan

En infrastrukturplan för 14 avdelningar Hör av er →

Varför det spelar roll

Kod från tredje part når ett projekt på ett av tre sätt: genom en pakethanterare, kopierad in i projektets repo tillsammans med sin licens och en kontrollsumma, vilket kallas vendoring, eller byggd från källkod. Pakethanteraren är det vanliga valet, och med goda skäl. Den löser beroendena i hela trädet, uppdaterar det med ett enda kommando och noterar vad den installerade. Priset är förtroende: projektet förlitar sig nu på varje manifest i trädet och varje register det kom ifrån. Vendoring byter det förtroendet mot arbete, eftersom den kopierade koden bara ändras när någon med avsikt kopierar in en ny version. Att bygga från källkod ger mest kontroll och kräver mest underhåll.

Det är i paketmanifestet som de flesta avvägningarna syns. Oftast anger det ett intervall av versioner snarare än en enda; i npm godtar ett caret-intervall som ^10.0.1 varje senare utgåva med samma huvudversion. Intervall låter rättningar komma in utan att någon redigerar filen, och de innebär också att två installationer med en veckas mellanrum kan ge olika träd. En låsfil sluter det glappet. npm skriver package-lock.json för att beskriva det exakta träd den byggde, så att senare installationer återskapar det, Cargo skriver Cargo.lock med den exakta versionen av varje beroende, och båda förväntar sig att filen checkas in. Manifestet anger vad ett projekt är villigt att godta; låsfilen anger vad det fick.

Manifesten säger också hur starkt ett paket behöver ett annat. Debian Policy Manual definierar en stege. Depends är ovillkorligt: paketet konfigureras inte förrän allt det beror på har konfigurerats. Recommends är starkt men inte ovillkorligt, för paket som finns bredvid det här i alla installationer utom de ovanliga. Suggests anger paket som gör det mer användbart, medan en installation utan dem är fullt rimlig. En installation som utelämnar ett rekommenderat paket är giltig, och programvara i den kan ändå behöva det paketet för någon av sina uppgifter.

Varje verktyg som resonerar om paket litar på deras metadata. En licenskontroll läser den deklarerade licensen, ofta ett SPDX-uttryck som MIT OR Apache-2.0, där OR erbjuder ett val och AND kräver varje licens på en gång. En sårbarhetsskanner matchar deklarerade namn och versioner mot publicerade säkerhetsmeddelanden. När en deklaration är fel blir varje verktyg som läser den fel på samma sätt, och därför förtjänar licensfilerna i ett levererat paket en egen genomläsning.

Så ser det ut i praktiken

En härdningsgranskning av en produktionsserver fann att dess jail för spärrning av intrångsförsök aldrig hade spärrat någon, och en av tre separata orsaker var ett förhållande mellan paket. Spärråtgärden anropade ett program för paketfiltrering som paketsystemet bara anger som rekommenderat i stället för krävt, och servern hade byggts utan det. Jailen startade, räknade misslyckade inloggningar och fattade sitt beslut varje gång, och föll sedan vid det sista steget, när spärren skulle tillämpas. Granskningen av tre SSH-skydd hittade felet genom att läsa av det körande systemet i stället för konfigurationsfilerna, och lämnade efter sig en regel: varje säkerhetsskydd backas upp av en automatisk kontroll, inte av en kommentar.

En Rust-kärna som länkas in i en levererad applikation för med sig hela sin beroendegraf in i produkten, här 644 crates, var och en med sin egen licensdeklaration. Ett beroende flera nivåer ned kan gå över till en copyleft-licens från en version till nästa, och det enda spåret skulle vara en ändrad rad i en låsfil som få läser. Licensgranskningen av 644 crates körs vid varje bygge: den prövar licensuttrycket för varje crate mot en tillåtelselista, godtar ett val mellan licenser när någon av dem är tillåten och en kombination bara när alla är det, och stoppar bygget vid allt som inte matchar. För att visa att kontrollen kan slå till bröts den med flit två gånger, en gång genom att licensuttrycket för en crate skrevs om och en gång genom att den fastlåsta kärnrevisionen flyttades utan att granskningen genererades om. Sidan anger begränsningen rakt ut: granskningen läser deklarerade metadata, som kan vara felaktiga, och den är ingen juridisk rådgivning.

Företagets egen git-forge installerades från leverantörens eget paketförråd som ett enda program som operativsystemet håller igång. Att lägga till en leverantörs paketförråd på en server är ett beslut i sig, eftersom rutinmässiga uppdateringar från och med då installerar vad som helst som den leverantören publicerar och signerar.

Själva webbplatsen behöver inte Node för att köras. Dess package.json är markerad som privat, vilket får npm att vägra publicera den, listar bara utvecklingsverktyg, med caret-intervall, och ligger bredvid en incheckad package-lock.json. Dess pyproject.toml konfigurerar en linter, en stavningskontroll och en typkontroll och har varken en tabell [project] eller [build-system], eftersom den inte beskriver något paket att bygga. Ikonerna är vendorerade, inkopierade med licensen från uppströmsprojektet och en SHA-256-kontrollsumma för varje fil.

Var det går fel

En funktion rapporterar att allt är friskt och gör ingenting

En tjänst startar, loggar att den körs och hoppar i tysthet över en av sina uppgifter varje gång. Ett program som den anropar hör till ett rekommenderat eller föreslaget paket, som installationen hade rätt att utelämna. När en funktion beror på ett externt program, kontrollera att det finns vid uppstart och stoppa med ett tydligt fel om det saknas, deklarera det som ett hårt krav i den egna driftsättningen och testa funktionen från början till slut på en server som är byggd som produktion.

Två maskiner installerar olika kod från samma manifest

Bygget går igenom på en bärbar dator och går sönder i CI, eller tvärtom, utan att projektets egen kod har ändrats. Manifestet anger versionsintervall, låsfilen checkades aldrig in, och ett beroende publicerade en ny utgåva mellan de två installationerna. Checka in låsfilen, installera från den i automatiseringen med kommandon som vägrar ändra den, som npm ci eller cargo build –locked, och uppdatera den som en egen, granskad ändring.

En licens ändras inuti ett välbekant beroende

En produkt som klarade en licensgranskning förra året levererar nu kod under villkor som ingen har godtagit, medan listan över beroenden ser likadan ut. Ett befintligt beroende bytte licens mellan två versioner, och en rutinmässig uppdatering drog in den nya versionen genom låsfilen. Kontrollera deklarerade licenser mot en uttrycklig tillåtelselista vid varje bygge, så att ändringen stoppar bygget samma dag som den kommer.

Läs mer

- npm Docs: About packages and modules — definitionen från npm själva av ett paket, och varför en modul inte alltid är ett.

- Debian Policy Manual: Declaring relationships between packages — Depends, Recommends, Suggests och de andra fälten, och vad en pakethanterare gör med vart och ett.
- Python Packaging User Guide: Writing your pyproject.toml — de tre tabellerna i pyproject.toml och de metadata som ett distributionspaket för Python deklarerar.
- The Cargo Book: Cargo.toml vs Cargo.lock — varför paketmanifestet beskriver vad ett projekt godtar, varför låsfilen anger exakta versioner och när den ska checkas in.
- SPDX Specification 2.3.1: License expressions — syntaxen som paket använder för att deklarera en licens, ett val mellan flera eller en uppsättning som gäller tillsammans.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ett programpaket?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/package/