Engineer

Programvarepakke

kode buntet med en beskrivelse, klar til å installeres

Hva det er

En programvarepakke er formen programvare reiser i fra dem som skriver den, til maskinene som kjører den: koden, som et arkiv eller en katalog, sammen med et manifest som oppgir navn, versjon og hva annet den trenger for å virke. I daglig tale heter den bare en pakke, med flertall pakker og pakkene, og på engelsk package. Hvert økosystem trekker grensene på sitt eget sted. For npm er en pakke en fil eller katalog som beskrives av en package.json-fil, og npm-dokumentasjonen skiller den fra en modul, som er alt Node.js kan laste med require() eller import; bare moduler som har en package.json, er pakker. Python bruker ordet om to ting, og Python Packaging User Guide holder dem atskilt: en importpakke (import package) er en katalog med moduler som koden importerer, mens en distribusjonspakke (distribution package) er det versjonerte arkivet en bruker laster ned og installerer. I Rust samler en pakke én eller flere crates med en Cargo.toml som sier hvordan de skal bygges. I Debian er enheten binærpakken, .deb-filen systemet installerer og holder rede på.

En pakke er en distribusjonsenhet, og den er lettest å definere mot naboene. Et bibliotek er kode; en pakke er måten den koden leveres på, og ett bibliotek kan nå brukerne som en npm-pakke, et Python-wheel og en Debian-pakke på samme tid. En pakkebehandler er programmet som leser manifester, henter det de lister opp, og installerer alt i en rekkefølge som virker: npm, pip, Cargo og apt. Et register eller en pakkebrønn er stedet der pakkene venter på å bli hentet, f.eks. npm-registeret, PyPI, crates.io eller et Debian-speil. Pakkebehandlere for et programmeringsspråk installerer kode for ett prosjekt, som regel i en katalog ved siden av det. Pakkebehandlere for systemet installerer programvare for hele maskinen og noterer hvilken pakke som la hver fil hvor.

Bruk begrepet i en beslutning

Bruk «Hva er en programvarepakke?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Formen på et system tegnet opp på papiret først, slik at delene passer sammen og kostnaden er kjent før noen skriver kode.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

Tredjepartskode kommer inn i et prosjekt på én av tre måter: via en pakkebehandler, kopiert inn i repoet med lisensen og en sjekksum, det som kalles vendoring, eller bygd fra kildekoden. En pakkebehandler er det vanlige valget, og med god grunn. Den løser avhengighetene i hele treet, oppdaterer dem med én kommando og noterer hva den installerte. Prisen er tillit: prosjektet stoler nå på hvert manifest i det treet og på hvert register det kom fra. Vendoring bytter den tilliten mot arbeid, fordi den kopierte koden bare endres når noen bevisst kopierer inn en ny versjon. Å bygge fra kildekoden gir mest kontroll og krever mest vedlikehold.

Det er i pakkemanifestet de fleste avveiningene viser seg. Det oppgir som regel et intervall av versjoner i stedet for én; i npm godtar et caret-intervall som ^10.0.1 enhver senere utgivelse med samme hovedversjon. Intervaller lar rettinger komme uten at noen redigerer filen, men de betyr også at to installasjoner med en ukes mellomrom kan gi ulike trær. En låsefil tetter det hullet. npm skriver package-lock.json for å beskrive det nøyaktige treet den bygde, så senere installasjoner gjenskaper det, Cargo skriver Cargo.lock med den nøyaktige versjonen av hver avhengighet, og begge forventer at filen sjekkes inn. Manifestet registrerer hva et prosjekt er villig til å godta; låsefilen registrerer hva det fikk.

Manifestene sier også hvor sterkt én pakke trenger en annen. Debian Policy Manual definerer en stige. Depends er absolutt: pakken konfigureres ikke før alt den er avhengig av, er konfigurert. Recommends er sterk, men ikke absolutt, og gjelder pakker som finnes sammen med denne i alle unntatt uvanlige installasjoner. Suggests nevner pakker som gjør den mer nyttig, mens det er helt rimelig å installere uten dem. En installasjon som utelater en anbefalt pakke, er gyldig, og programvaren i den kan likevel trenge den pakken til én av oppgavene sine.

Hvert verktøy som vurderer pakker, stoler på metadataene deres. En lisenssjekker leser den deklarerte lisensen, ofte et SPDX-uttrykk som MIT OR Apache-2.0, der OR gir et valg og AND krever alle lisensene samtidig. En sårbarhetsskanner sammenligner deklarerte navn og versjoner med publiserte sikkerhetsvarsler. Når en deklarasjon er feil, tar hvert verktøy som leser den, feil på samme måte, og derfor fortjener lisensfilene inne i en levert pakke en egen gjennomlesing.

Slik ser det ut i praksis

En herdingsgjennomgang av en produksjonsserver fant at jailen for utestenging av inntrengere aldri hadde stengt ute noen, og én av tre separate årsaker var et forhold mellom pakker. Utestengingshandlingen kalte et pakkefilterprogram som pakkesystemet oppgir som anbefalt i stedet for påkrevd, og serveren var bygd uten det. Jailen startet, talte mislykkede innlogginger og kom frem til en avgjørelse hver gang, og feilet så i det siste steget, der utestengingen skulle tre i kraft. Gjennomgangen av tre SSH-vern fant det ved å undersøke det kjørende systemet i stedet for konfigurasjonsfilene, og etterlot en regel om at hver sikkerhetskontroll bakkes opp av en automatisk kontroll, ikke av en kommentar.

En Rust-kjerne som lenkes inn i en levert app, tar med seg hele avhengighetsgrafen sin inn i produktet, her 644 crates, hver med sin egen lisensdeklarasjon. En avhengighet flere nivåer ned kan gå over til en copyleft-lisens fra én versjon til den neste, og det eneste sporet vil være en endret linje i en låsefil få leser. Lisensrevisjonen av 644 crates kjører ved hvert bygg: den vurderer lisensuttrykket til hver crate mot en tillatelsesliste, godtar et valg mellom lisenser når én av dem er tillatt, og en kombinasjon bare når alle er det, og stopper bygget ved alt som ikke har treff. For å vise at kontrollen kan slå ut, ble den brutt med vilje to ganger, én gang ved å skrive om lisensuttrykket til en crate og én gang ved å flytte den låste kjernerevisjonen uten å generere revisjonen på nytt. Siden sier rett ut hvor grensen går: revisjonen leser deklarerte metadata, som kan være feil, og den er ikke juridisk rådgivning.

Selskapets egen git-forge ble installert fra leverandørens pakkebrønn som ett enkelt program som operativsystemet holder i gang. Å legge til en leverandørs pakkebrønn på en server er en beslutning i seg selv, fordi rutineoppdateringer fra da av installerer det leverandøren publiserer og signerer.

Dette nettstedet trenger ikke Node for å kjøre. Filen package.json er merket som privat, noe som får npm til å nekte å publisere den, lister bare utviklingsverktøy, med caret-intervaller, og ligger ved siden av en innsjekket package-lock.json. Filen pyproject.toml setter opp en linter, en stavekontroll og en typesjekker og har ingen `[project]` - eller `[build-system]` -tabell, fordi den ikke beskriver noen pakke som skal bygges. Ikonene er vendoret: kopiert inn med lisensen fra oppstrøms og en SHA-256-sjekksum for hver fil.

Hvor det går galt

En funksjon melder at alt er i orden, og gjør ingenting

En tjeneste starter, logger at den kjører, og hopper i det stille over én av oppgavene sine hver gang. Et program den kaller, hører til en anbefalt eller foreslått pakke, som installasjonen hadde full rett til å utelate. Når en funksjon er avhengig av et eksternt program, sjekk at programmet finnes ved oppstart, og stopp med en tydelig feilmelding hvis det mangler. Deklarer det som et absolutt krav i deres egen produksjonssetting, og test funksjonen fra ende til ende på en server bygd som produksjon.

To maskiner installerer ulik kode fra samme manifest

Bygget går gjennom på en bærbar maskin og feiler i kontinuerlig integrasjon, eller omvendt, uten at prosjektets egen kode er endret. Manifestet oppgir versjonsintervaller, låsefilen ble aldri sjekket inn, og en avhengighet publiserte en ny utgivelse mellom de to installasjonene. Sjekk inn låsefilen, installer fra den i automatiseringen med kommandoer som nekter å endre den, f.eks. `npm ci` eller `cargo build --locked`, og oppdater den som en egen endring som gjennomgås.

En lisens endres inne i en kjent avhengighet

Et produkt som besto en lisensgjennomgang i fjor, leveres nå med kode under vilkår ingen har godtatt, mens listen over avhengigheter ser lik ut. En eksisterende avhengighet endret lisens mellom to versjoner, og en rutineoppdatering dro den nye versjonen inn via låsefilen. Sjekk deklarerte lisenser mot en eksplisitt tillatelsesliste ved hvert bygg, så endringen stopper bygget samme dag som den kommer.

Les mer

- npm Docs: About packages and modules — npms egen definisjon av en pakke, og hvorfor en modul ikke alltid er det.

- Debian Policy Manual: Declaring relationships between packages — Depends, Recommends, Suggests og de andre feltene, og hva en pakkebehandler gjør med hvert av dem.
- Python Packaging User Guide: Writing your pyproject.toml — de tre tabellene i pyproject.toml og metadataene en distribusjonspakke for Python deklarerer.
- The Cargo Book: Cargo.toml vs Cargo.lock — hvorfor manifestfilen beskriver hva et prosjekt godtar, mens låsefilen registrerer nøyaktige versjoner, og når den bør sjekkes inn.
- SPDX Specification 2.3.1: License expressions — syntaksen pakker bruker for å deklarere én lisens, et valg mellom flere eller et sett som gjelder samtidig.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en programvarepakke?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/package/