Engineer

Paquete de software

Código empaquetado con una descripción, listo para instalar

Qué es

Un paquete es la forma en que el software pasa de quienes lo escriben a las máquinas que lo ejecutan: el código, en un archivo o un directorio, acompañado de un manifiesto que indica su nombre, su versión y qué más necesita para funcionar. El plural es paquetes, y la documentación de sistemas suele hablar de paquetes de software. Cada ecosistema fija sus propios límites. Para npm, un paquete es un archivo o directorio descrito por un archivo package.json, y su documentación lo distingue de un módulo: cualquier cosa que Node.js pueda cargar con require() o import; solo los módulos que incluyen un package.json son paquetes. Python usa la palabra para dos cosas, y la Guía del usuario de Python Packaging las distingue: un paquete de importación es un directorio de módulos que el código importa, mientras que un paquete de distribución es el archivo con versión que una persona descarga e instala. En Rust, un paquete reúne uno o más crates con un Cargo.toml que indica cómo compilarlos. En Debian, la unidad es el paquete binario, el archivo .deb que el sistema instala y registra.

Un paquete es una unidad de distribución, y se entiende mejor al compararlo con los conceptos cercanos. Una biblioteca es código; un paquete es la forma de distribuirlo, y una misma biblioteca puede llegar a sus usuarios como paquete npm, wheel de Python y paquete Debian a la vez. Un gestor de paquetes es el programa que lee los manifiestos, descarga lo que enumeran y lo instala todo en un orden en el que todo pueda funcionar: npm, pip, Cargo y apt. Un registro o repositorio de paquetes es donde estos esperan a ser descargados, como el registro de npm, PyPI, crates.io o una réplica de Debian. Los gestores de paquetes de lenguajes instalan código para un proyecto, normalmente en un directorio junto a él. Los gestores de paquetes del sistema instalan software para toda la máquina y registran qué paquete colocó cada archivo en cada sitio.

Utilice el concepto en una decisión

Aplique «¿Qué es un paquete de software?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La estructura del sistema se define primero sobre el papel para que las piezas encajen y los costes se conozcan antes de escribir código.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Un plan de infraestructura para 14 departamentos — 14 departamentos de investigación con un único plan de infraestructura

Un plan de infraestructura para 14 departamentos Póngase en contacto →

Por qué importa

El código de terceros llega a un proyecto de una de estas tres maneras: mediante un gestor de paquetes, copiado al repositorio con su licencia y una suma de comprobación, lo que se llama vendoring, o compilado desde el código fuente. Un gestor de paquetes suele ser la opción elegida por buenas razones. Resuelve todo el árbol de dependencias, las actualiza con un solo comando y registra lo que instala. El coste es la confianza: el proyecto pasa a depender de cada manifiesto de ese árbol y de cada registro del que procede. El vendoring cambia esa confianza por trabajo, porque el código copiado solo cambia cuando alguien copia una versión nueva de forma deliberada. Compilar desde el código fuente ofrece el máximo control y exige el máximo mantenimiento.

En el manifiesto se aprecian la mayoría de las ventajas y las contrapartidas. Suele indicar un intervalo de versiones, no una sola; en npm, un intervalo con circunflejo como ^10.0.1 admite cualquier versión posterior con la misma versión principal. Los intervalos permiten recibir correcciones sin editar el archivo, y también hacen que dos instalaciones separadas por una semana puedan producir árboles distintos. Un archivo de bloqueo cierra esa brecha. npm escribe package-lock.json para describir el árbol exacto que construyó, de modo que las instalaciones posteriores lo reproduzcan; Cargo escribe Cargo.lock con la versión exacta de cada dependencia, y ambos esperan que el archivo se incluya en el control de versiones. El manifiesto registra lo que un proyecto está dispuesto a aceptar; el archivo de bloqueo registra lo que recibió.

Los manifiestos también indican hasta qué punto un paquete necesita a otro. El Manual de políticas de Debian define una escala. Depends es absoluto: el paquete no se configura hasta que se hayan configurado todas sus dependencias. Recommends es una relación fuerte, aunque no absoluta, para paquetes que acompañan a este en todas las instalaciones salvo las poco habituales. Suggests enumera paquetes que lo hacen más útil, aunque instalarlo

sin ellos es perfectamente razonable. Una instalación que omite un paquete recomendado es válida, y el software que contiene puede seguir necesitando ese paquete para alguna de sus tareas.

Toda herramienta que analiza paquetes confía en sus metadatos. Un comprobador de licencias lee la licencia declarada, a menudo una expresión SPDX como MIT OR Apache-2.0, donde OR ofrece una elección y AND exige todas las licencias a la vez. Un analizador de vulnerabilidades coteja los nombres y las versiones declarados con los avisos publicados. Si una declaración es incorrecta, todas las herramientas que la leen se equivocan del mismo modo; por eso los archivos de licencia incluidos en un paquete distribuido merecen una lectura por separado.

Cómo se aplica en la práctica

Una revisión de endurecimiento de un servidor de producción descubrió que su jail de bloqueo de intrusiones nunca había bloqueado a nadie, y una de las tres causas independientes era una relación entre paquetes. La acción de bloqueo llamaba a un programa de filtrado de paquetes que el empaquetado clasifica como recomendado, no obligatorio, y el servidor se había construido sin él. El jail arrancaba, contaba los inicios de sesión fallidos y tomaba su decisión cada vez, pero fallaba en el último paso: aplicar el bloqueo. La revisión de tres protecciones de SSH lo descubrió inspeccionando el sistema en ejecución, en vez de los archivos de configuración, y dejó una regla: cada control de seguridad debe estar respaldado por una aserción automatizada, no por un comentario.

Un núcleo de Rust enlazado con una aplicación distribuida incorpora al producto todo su grafo de dependencias: aquí, 644 crates, cada uno con su propia declaración de licencia. Una dependencia situada varios niveles por debajo puede pasar a una licencia copyleft de una versión a otra, y la única huella sería una línea modificada en un archivo de bloqueo que poca gente lee. La auditoría de licencias de 644 crates se ejecuta en cada compilación: evalúa la expresión de licencia de cada crate frente a una lista de licencias permitidas, admite una elección si al menos una licencia está permitida y una combinación solo si lo están todas, y detiene la compilación ante cualquier caso que no encaje. Para demostrar que el control podía activarse, se hizo fallar deliberadamente dos veces: una al reescribir la expresión de licencia de un crate y otra al cambiar la revisión fijada del núcleo sin regenerar la auditoría. La página expone el límite con claridad: la auditoría lee metadatos declarados, que pueden ser incorrectos, y no constituye asesoramiento jurídico.

La forja Git de la empresa se instaló desde la fuente de paquetes del propio proveedor como un único programa que el sistema operativo mantiene en ejecución. Añadir el repositorio de un proveedor a un servidor es una decisión por sí misma, porque, desde ese momento, las actualizaciones habituales instalan lo que ese proveedor publique y firme.

Este sitio no necesita Node para funcionar. Su package.json está marcado como private, lo que indica a npm que rechace su publicación, enumera únicamente herramientas de desarrollo con intervalos de versión con circunflejo y está junto a un package-lock.json incluido en el control de versiones. Su pyproject.toml configura un linter, un corrector ortográfico y un comprobador de tipos, y no contiene tablas [project] ni [build-system], porque no describe ningún paquete que generar. Los iconos se incorporan mediante vendoring: se copian con la licencia del proyecto de origen y una suma SHA-256 para cada archivo.

Dónde surgen los problemas

Una función aparece como operativa, pero no hace nada

Un servicio arranca, registra que está funcionando y omite silenciosamente una de sus tareas cada vez. Un programa al que llama pertenece a un paquete recomendado o sugerido, que la instalación podía omitir legítimamente. Si una función depende de un programa externo, compruebe su presencia al arrancar y deténgase con un error claro si falta, declárelo como requisito obligatorio de su propio despliegue y pruebe la función de extremo a extremo en un servidor construido como el de producción.

Dos máquinas instalan código distinto desde el mismo manifiesto

La compilación funciona en un portátil y falla en integración continua, o al revés, sin cambios en el código del proyecto. El manifiesto indica intervalos de versión, nunca se incluyó el archivo de bloqueo en el control de versiones y una dependencia publicó una versión entre ambas instalaciones. Incluya el archivo de bloqueo en el control de versiones, instale a partir de él en la automatización con comandos que se nieguen a modificarlo, como npm ci o cargo build –locked, y actualícelo mediante un cambio separado y revisado.

Una dependencia conocida cambia de licencia

Un producto que superó una revisión de licencias el año pasado distribuye ahora código con condiciones que nadie aceptó, aunque su lista de dependencias parezca la misma. Una dependencia existente cambió de licencia entre versiones, y una actualización habitual incorporó la nueva versión mediante el archivo de bloqueo. Compruebe las licencias declaradas frente a una lista explícita de licencias permitidas en cada compilación, para que el cambio la detenga el mismo día en que llegue.

Lecturas adicionales

- Documentación de npm: paquetes y módulos — la definición de paquete de npm y por qué un módulo no siempre lo es.
- Manual de políticas de Debian: declaración de relaciones entre paquetes — Depends, Recommends, Suggests y los demás campos, y qué hace un gestor de paquetes con cada uno.
- Guía del usuario de Python Packaging: escribir pyproject.toml — las tres tablas de pyproject.toml y los metadatos que declara un paquete de distribución de Python.
- El libro de Cargo: Cargo.toml frente a Cargo.lock — por qué el manifiesto describe lo que acepta un proyecto, el archivo de bloqueo registra versiones exactas y cuándo incluirlo en el control de versiones.
- Especificación SPDX 2.3.1: expresiones de licencia — la sintaxis con la que los paquetes declaran una licencia, una elección entre varias o un conjunto que se aplica a la vez.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un paquete de software?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/package/