PACKAGE MANAGEMENT

Marina Seijo Gómez UO288559

Pedro Castro Montes UO288120

Javier Monteserín Rdgz. UO288524

Introducción

En el capítulo 489 del podcast Software Engineer Radio (SER) Sam Boyer, un ingeniero autodidacto, escritor del articulo *"So You Want to Write a Package Manager"* y actualmente empleado de Grafana Labs, es entrevistado por Robert Blumen. El objetivo de este artículo es recopilar y resumir el contenido del episodio.

¿Qué es un paquete?

Dar una definición precisa de lo que constituye un "paquete" se presenta en el podcast como una tarea complicada y frustrante. Esto deriva de que las personas tienen distintos conceptos de este término, normalmente influenciados por diversos factores como su área de trabajo o el lenguaje de programación que más usan, según explica Sam Boyer. Sin embargo, se puede definir como: una colección con un nombre y un contenido que tiene algún tipo de límite lógico que lo distingue/separa de otros paquetes y sus respectivos contenidos.

El uso de paquetes durante el desarrollo software se trata de una práctica muy habitual. Pero realmente ¿qué aportan a nuestros proyectos?

El beneficio principal y más evidente es la organización del código. De la mano de este también va la colaboración, entre los miembros de un equipo de desarrollo, y el mantenimiento, influido también por la división del código en partes más pequeñas y manejables. Otro factor es la reutilización ya que facilita el uso de un mismo código en distintos proyectos.

¿Qué es y para qué sirve un gestor de paquetes?

Los gestores de paquetes se tratan de herramientas que facilitan la instalación, actualización y gestión de paquetes en proyectos software. Podríamos decir que se tratan de herramientas de automatización, sin embargo, Sam Boyer explica que su valor no reside en esto sino en la uniformidad. En el podcast se define uniformidad como: la forma en la que se crea un sistema tras la ejecución de un comando. Pero realmente, ¿qué factores aportan uniformidad? Tras una investigación, personalmente, considero que los factores que más influyen son: El uso de estándares de instalación y actualización garantizando la consistencia en la gestión de dependencias junto con la gestión centralizada de dependencias facilitando la resolución de problemas y evitando incompatibilidades.

¿Cómo acceden los gestores a los paquetes?

Los gestores de paquetes implementan distintos modelos que les permiten acceder a los paquetes solicitados por el usuario. Principalmente hay 4 modelos:

1. Modelo central único en el que existe un único repositorio al que accede el gestor.
2. Modelo de varios repositorios, estos pueden ser especificados. El gestor de paquetes implementa un algoritmo de búsqueda entre los distintos repositorios.

3. Modelo distribuido. Se basa en la distribución de los datos en bloque en una red de nodos de forma ligada y cifrada.
4. Modelo por nombre.

Dependencias

El uso de paquetes y/o bibliotecas provoca dependencias. Siendo posible que un paquete dependa de 1 o más paquetes, que a su vez dependen de otros. Realizar manualmente la instalación o referenciación de estas sería una tarea ardua, sin embargo, puede ser delegada a un gestor de paquetes. Esas herramientas realizan un cálculo recursivo para calcular eficazmente todas las dependencias del sistema, dando como resultado un grafo.

Dependency Hell

Coloquialmente conocido como "El infierno de las dependencias" se trata de un problema provocado por una gran cantidad de dependencias enlazadas entre sí. Este problema puede surgir por varios motivos. (1) En sistemas donde se necesiten múltiples librerías, pero de los que se aprovechan pocas utilidades, dando lugar a mucha información redundante. (2) En sistemas en los que se producen largas cadenas de dependencias provocadas por ejemplo porque: el sistema depende de una librería A que a su vez depende de B que depende de C etc.….

Selección de versiones

La correcta selección de versiones es una tarea primordial. Los gestores de paquetes ofrecen a los usuarios distintas opciones. La primera es especificar la versión o bien un rango de versiones. Otra opción es no especificarla. En este contexto el gestor realizará la instalación de la última versión disponible. Estas restricciones permiten evitar la actualización automática de los paquetes provocada por una actualización recursiva de aquellos que dependen de estos.

Conflictos entre versiones

En el podcast, Sam Boyer pone el siguiente ejemplo: si tanto A como B dependen de un mismo paquete C pero requieren distintas versiones, por ejemplo, la 1.14 y 1.16, respectivamente, ¿Cómo lo resuelve el gestor de paquetes? Hay distintas formas para solucionar este tipo de conflictos. En aquellos lenguajes en lo que se siga "The Highlander rule", es decir, en los que solo se permita una instancia del mismo paquete, su respuesta depende de la política de gestión de versiones. Las políticas más comunes son: (1) uso de la versión más reciente, (2) la más antigua o (3) una intermedia que cumplan los requisitos de ambas dependencias. En aquellos lenguajes en los que no se siga la regla ya mencionada simplemente se tendría una instancia por cada versión solicitada.

Ciclos

Ante los ciclos no hay una clara solución ya que su aparición en grandes sistemas, o incluso en pequeños sistemas, es inevitable. Se trata esta la razón por la que la gran mayoría de los gestores de paquetes soportan los ciclos. El verdadero desafío ante este problema no es solucionarlo sino la construcción de un sistema lógico sensato aun teniendo dependencias cíclicas.