Package management

Hecho por:

Alberto Guerra Rodas
Ángel Macías Rodríguez
Pedro Limeres Granado
Sergio Truébano Robles

What is a package?
A Package is defined as a collection of elements with a name and a logical boundary within a larger context. The contents can include source code, compiled code or configuration files. Defining a package universally is challenging due to its abstract nature, and validity depends on the specific ecosystem, for example in GO, the packages are called modules which are a git tag attached to a commit, representing a snapshot of files in the git repository, including go files and metadata files like go.mod and go.sum.

Software Packages and Metadata
Metadata, in the context of software packages, comprises essential information such as the package name and version. It acts as a two-point coordinate system, allowing for reference and coordination among packages. Additionally, metadata often includes a list of dependencies, specifying other required packages and their compatible versions.

Package Manager
Package managers as automation tools that download and organize packages, ensuring uniformity for reproducible builds.
Popular language examples like Node (NPM or Yarn) and Rust (Cargo) illustrate the close integration of language and package manager. The evolving trend is that newer languages include package managers as part of their toolchain.
Package managers are usually developed in the same language as the primary programming language to ensure smooth execution. In the case of Python, practical considerations emphasize aligning the language and package manager to ensure compatibility across different platforms.

Package Transmission Mechanisms

There could be different models for package managers to retrieve packages:

- Central repository: a single repository with all the packages.
- Search algorithm: an algorithm selecting from multiple available repositories.
- Distributed model: distributed address resolution system, usually combined with blockchain.
- URL-based model: packages names are represented as URLs, so name resolution is made by means of DNS server.

In addition, package managers also deal with publishing operations of the packages, being sure they are validated before the release is made.

Validity process

Package managers should be able to ensure some way of allowing uniformity across all the users regarding packages' name resolution when having multiple repositories available. Moreover, users do not want to deal with verifying that the package they are installing is actually the right one, so package managers verify and authenticate packages' owners with cryptographic signatures and metadata.

Dependencies

The set of packages that are required from a given project are converted into a set of dependencies inside a metadata configuration file, or a set of commands that are run through a command-line interface. Packages may also reuse parts of its content with other packages, leading to dependency issues.

Cycles

In graph theory, a cycle is a path that starts from a given vertex and ends at the same vertex. When we apply this to package management, we conclude a graph exists when for example, one package A depends on another package B and this one depends on another package C depends on A as well.

When some problems related with version managers occur, version managers must decide what is the best thing to do in order to keep the program. The managers normally make different decisions depending on what language they are working with, when the language is dynamic, and there are no problems with type checking, the packages can usually be duplicated to achieve the needs in the version of another package. However, when the language has type checking and they are checked statically, the problem escalates. A solution for the problem could be taking the latest version accepted by all packages.

Dependency limbo vs Dependency hell

When the last thing explained happens it can lead into one of these things or even both. On the one side we have dependency limbo that means the programmer can get worried if this kind of problems occur, and may start worrying about the correctness of its code and whether or not it can compile, but with no proof that those problems will appear.

Dependency hell

The dependency hell expression is used to refer to the situation where a lot of packages depend on other packages and so on, leading to many circles. Then, when a package is

updated, it may cause errors in other packages that depend on him. To avoid the problem the version could be overridden, so it is declared the exact version to be used for a package. The main problem is that all the packages depending on that one must override too, and those that depend on him, which leads to a chain of overrides. Thus, it is easy that someone doesn't realize that an override is needed, so it wouldn't work. Besides, the version in the override may be changed, so the structure would be destroyed.

Lock file

The lock file saves all the dependencies and versions of a given moment in a single file with the intention of reproducing builds. It is necessary with version selection algorithms, but it could be omitted if the algorithm is stable, which means that given an input it always gives the same set of versions. Those versions declared on the lock file are said to be pinned because it is the exact one that is going to be used so even if it will not change or be updated, it's a constraint.

Package loading

In languages like the C languages, all libraries are typically incorporated to the point where you don't need them separately, leading to the failures to happen during development.
On the other hand, languages like Ruby or Python dynamically load libraries, leading to the failures to happen at runtime.