Engineer

パッケージ

コードに説明を添え、インストールできる形にまとめたもの

どのようなものか

パッケージは、ソフトウェアを作る人から実行するマシンへ届けるための形です。アーカイブやディレクトリーにしたコードに、名前、バージョン、動作に必要なものを記したマニフェストを添えます。英語の複数形はpackagesで、システムのドキュメントではソフトウェアパッケージとも呼びます。その範囲はエコシステムごとに違います。npmでは、package.jsonで記述されるファイルやディレクトリーがパッケージです。npmのドキュメントは、Node.jsがrequire()かimportで読み込めるものをモジュールと呼んで区別し、package.jsonを持つモジュールだけをパッケージとします。Pythonでは同じ語を二つの意味で使い、Python Packaging User Guideはそれを区別します。インポートパッケージはコードがインポートするモジュールのディレクトリーで、配布パッケージは利用者がダウンロードしてインストールする、バージョン付きのアーカイブです。Rustでは、一つ以上のcrateと、そのビルド方法を記したCargo.tomlをまとめます。Debianでは、システムがインストールして管理する.debファイル、バイナリーパッケージが単位です。

パッケージは配布の単位であり、近い概念と比べると分かりやすくなります。ライブラリーはコード、パッケージはその配り方です。一つのライブラリーを、npmパッケージ、Pythonのwheel、Debianパッケージとして同時に届けられます。パッケージマネージャーは、マニフェストを読み、そこに挙げたものを取得して、全体が動作するような順序ですべてインストールするプログラムです。npm、pip、Cargo、aptなどがあります。レジストリーやアーカイブは、取得を待つパッケージの置き場所です。npmレジストリー、PyPI、crates.io、Debianのミラーなどが該当します。言語用のパッケージマネージャーは、一つのプロジェクトのためにコードをインストールし、通常はその隣のディレクトリーに置きます。システム用のものはマシン全体にソフトウェアを導入し、どのパッケージがどのファイルをどこに置いたかを記録します。

概念を判断に生かす

「パッケージとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 まず全体像を設計し、各部分のつながりと費用を、コードを書き始める前に明らかにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 14部門を支える共通インフラ計画 — 14 共通インフラ計画の対象となった研究部門

14部門を支える共通インフラ計画 お問い合わせ →

なぜ重要なのか

第三者のコードをプロジェクトに取り込む方法は三つあります。パッケージマネージャーを使う、ライセンスとチェックサムを添えてリポジトリーにコピーするvendoringを行う、ソースからビルドする、という方法です。通常パッケージマネージャーが選ばれるのには理由があります。依存関係のツリー全体を解決し、一つのコマンドで更新し、インストールしたものを記録します。代償は信頼です。プロジェクトは、そのツリーのすべてのマニフェストと、取得元のすべてのレジストリーに依存します。vendoringは、その信頼を作業に置き換えます。誰かが意図して新しい版をコピーしたときだけ、取り込んだコードが変わるためです。ソースからのビルドは最も細かく制御できますが、保守の手間も最大になります。

その得失の多くがマニフェストに表れます。通常は一つの版ではなく、バージョンの範囲を指定します。npmの^10.0.1のようなキャレット付きの範囲は、メジャーバージョンが同じ、それ以降のリリースを認めます。ファイルを編集せず修正を受け取れる一方、一週間離れた二回のインストールで、異なる依存関係ツリーができることもあります。この差を埋めるのがロックファイルです。npmは、後で同じツリーを再現できるよう、構築した正確なツリーをpackage-lock.jsonに書きます。Cargoは、全依存関係の正確なバージョンをCargo.lockに書き、どちらもファイルをバージョン管理にコミットすることを前提とします。マニフェストは受け入れられるもの、ロックファイルは実際に受け取ったものを記録します。

マニフェストは、あるパッケージが別のものをどの程度必要とするかも示します。Debian Policy Manualは、その段階を定義します。Dependsは絶対的で、依存先のすべてが設定されるまで、そのパッケージも設定されません。Recommendsは強い依存ですが絶対ではなく、特殊な場合を除き、どのインストールでも一緒に使うパッケージを指します。Suggestsは、あると便利になるパッケージを挙げ、なくてもインストールとしては十分妥当です。推奨パッケージを省いたインストールは有効でも、中のソフトウェアが、ある仕事にそれを必要とする場合があります。

パッケージを分析するツールは、どれもメタデータを信頼します。ライセンス検査は宣言されたライセンスを読み、MIT OR Apache-2.0のようなSPDX式を使うこともよくあります。ORは選択を認め、ANDはすべてのライセンスを同時に満たすことを求めます。脆弱性スキャナーは、宣言された名前と版を、公開された勧告と照合します。宣言が間違うと、それを読むツールも同じように間違います。配布パッケージ内のライセンスファイルを、別途読む価値があるのはそのためです。

実際の活用例

本番サーバーの強化レビューでは、侵入を遮断するjailが誰一人として遮断していなかったことが分かり、独立した三つの原因の一つがパッケージ間の関係でした。遮断処理が呼ぶパケットフィルタープログラムは、パッケージの定義では必須ではなく推奨で、サーバーには導入されていませんでした。jailは起動し、ログイン失敗を数え、毎回遮断の判断に達しても、最後に遮断を適用する段階で失敗していました。SSHの三つの防御を調べたレビューは、設定ファイルではなく稼働中のシステムを調べて発見しました。そして、すべてのセキュリティー制御を、コメントではなく自動化したアサーションで裏付ける規則を残しました。

出荷したアプリケーションにRustのコアをリンクすると、依存関係のグラフ全体が製品に入ります。この例では644のcrateがあり、それぞれがライセンスを宣言します。何層も下の依存先が、次の版でコピーレフトのライセンスに変わっても、ほとんど読まれないロックファイルの一行だけが変化の痕跡になるかもしれません。644のcrateのライセンス監査は、ビルドのたびに動きます。各crateのライセンス式を許可リストに照らし、選択肢なら一つでも許可されれば通し、組み合わせならすべてが許可されたときだけ通して、合致しないものがあればビルドを止めます。検査が働くことを示すため、意図的に二回失敗させました。一回はcrateのライセンス式を書き換え、もう一回は監査を再生成せず、固定したコアのリビジョンを変えました。ページは限界も明記します。監査が読むのは間違うこともある宣言メタデータであり、法的助言ではありません。

自社のGitフォージは、OSが動かし続ける単一のプログラムとして、提供元自身のパッケージ配布元からインストールしました。提供元のリポジトリーをサーバーに加えること自体が、一つの判断です。それ以降、日常の更新で、その提供元が公開して署名したものをインストールするからです。

このサイト自身は、動作にNodeを必要としません。package.jsonはprivateにしてnpmに公開を拒否させ、キャレット付きのバージョン範囲で開発ツールだけを列挙し、コミット済みのpackage-lock.jsonと並べています。pyproject.tomlは、リンター、スペルチェッカー、型チェッカーを設定しますが、ビルドするパッケ

ージを記述しないため、[project]や[build-system]のテーブルもありません。アイコンはvendoringで取り込み、元のライセンスと、各ファイルのSHA-256チェックサムを添えてコピーしています。

問題が起きるところ

正常と報告する機能が、何もしていません

サービスは起動して稼働中とログに記録しながら、毎回、一つの仕事を黙って省きます。呼び出すプログラムが推奨や提案のパッケージに属し、インストール時に省いてよいものだったためです。外部プログラムに依存する機能では、起動時に存在を調べ、なければ明確なエラーで止めます。自分たちのデプロイ定義では必須と宣言し、本番と同様に構築したサーバーで、機能をエンドツーエンドで検査します。

同じマニフェストから、二台が違うコードをインストールします

プロジェクト自身のコードは変わらないのに、ノートパソコンではビルドが通り、継続的インテグレーションでは失敗する、またはその逆が起きます。マニフェストがバージョン範囲を指定し、ロックファイルをコミットしておらず、その二回の間に依存先が新しい版を公開したためです。ロックファイルをコミットし、自動化ではnpm ciやcargo build –lockedのように、ロックファイルの変更を拒むコマンドで、そこからインストールします。更新は別の変更としてレビューします。

使い慣れた依存先のライセンスが変わります

去年ライセンスレビューを通った製品が、依存先の一覧は同じに見えるのに、誰も同意していない条件でコードを出荷します。既存の依存先が版の間でライセンスを変え、通常の更新がロックファイルを通じて新しい版を取り込んだからです。ビルドのたびに、宣言されたライセンスを明示的な許可リストで調べ、変化したその日に、ビルドを止めます。

参考資料

- npmドキュメント：パッケージとモジュール — npm自身によるパッケージの定義と、モジュールが必ずしもパッケージではない理由です。
- Debian Policy Manual：パッケージ間の関係の宣言 — Depends、Recommends、Suggestsなどのフィールドと、それぞれに対するパッケージマネージャーの処理です。

- Python Packaging User Guide：pyproject.tomlの記述 — pyproject.tomlの三つのテーブルと、Pythonの配布パッケージが宣言するメタデータです。
- The Cargo Book：Cargo.tomlとCargo.lock — マニフェストが受け入れ可能なものを記述し、ロックファイルが正確な版を記録する理由と、コミットすべき時です。
- SPDX仕様2.3.1：ライセンス式 — 一つのライセンス、複数からの選択、同時に適用する組み合わせを宣言する構文です。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「パッケージとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/package/