L'enfer du packaging

Rencontres Python Lyon, le 18 mai 2022

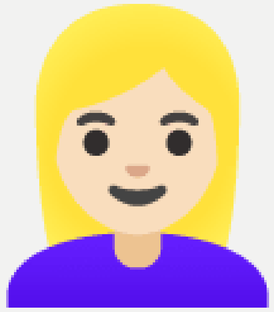
Témoignages

Écrire du code

J’adore écrire du code en Python 🐍. C’est simple, lisible, efficace. Je trouve en toute objectivité que c’est le meilleur langage de programmation du monde 🌍.

– Houssem, jeune développeur

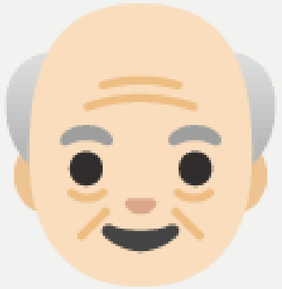
Écrire des tests



J’ai pleine confiance dans les outils 🧰 que j’utilise pour mes tests 👍. Pytest me fournit des fonctionnalités très appréciables pour mettre en place mes suites de tests, pierres angulaires de mon intégration continue 🦾.

– *Ada, directrice de projets informatiques*

Faire de la documentation



J’ai acquis une formidable expérience dans l’écriture de documentation 📔. Python, grâce à certains de ses paquets comme Sphinx, me donne de quoi faire de superbes pages qui aident nos utilisateurs à prendre du plaisir avec nos logiciels 💻.

– *Jean-Michel, architecte logiciel cheveronné*

Faire des paquets

Il y a 113 ans, j'ai commencé à lire la documentation officielle sur les paquets. J'ai survolé en quelques dizaines d'années le code de `distutils` avant de me rendre compte qu'il allait être déprécié 😭. J'ai étudié 2 398 solutions différentes sur StackOverflow qui m'ont proposé 731 outils pour créer mes paquets. L'an prochain, je me mets à Rust ⚙️.

– Gérard, ancien magicien de la gestion de projets

Trouver des solutions simples

Votre planète est étrange 🪐. Vous avez un langage de programmation qui prône la simplicité, cependant vos outils de packaging sont incroyablement complexes. C’est nul, non ? Vous voulez aller sur Mars, commencez par avoir des outils corrects pour vos paquets 📦.

– *Wendie, constructrice de zorglubs sur la planète OL69*

Faire des conférences

Salut 👋🏾 ! J’aime bien développer du code libre au sein de CourtBouillon, j’aime bien accompagner des gens sympas avec Stella. J’aime bien les navigateurs, les encodages, les dates, manger 🍲 et boire 🥂.

Et j’aime bien faire des paquets 🎁.

– *Guillaume, présentateur bavard*

Juste une mise au point

De quoi parle-t-on ?

Dans cette présentation, on parle :

- de l’architecture générale des paquets Python,
- de la création de paquets,
- de l’installation de paquets,
- de certaines spécifications associées à ces sujets,
- de certains outils associés à ces sujets.

C’est déjà bien assez long comme ça ⌚.

De quoi ne parle-t-on pas tout de suite ?

On parlera peut-être plus tard, autour d'un verre 🥛 :

- de ce qu'il faut utiliser pour les environnements virtuels,
- de ce qu'il faut utiliser pour gérer ses dépendances,
- de TOML, de reStructuredText, de Python 2,
- d'intégration continue, de reproductibilité, de graphes de dépendances,
- de HTML, de CSS, de SVG, de PDF,
- de l'état politique de la gauche, de la saison de l'Olympique Lyonnais ⚽.

De quoi ne parlera-t-on jamais ?

S’il vous plaît, arrêtez-moi si je me lance sur :

- sur certains trolls classiques  ,
- sur certains créateurs de certains projets,
- sur les mainteneurs de certains paquets de certaines distributions Linux  .

Quiz

Pourquoi en est-on là ?

A. L’équipe de développement de Python est stupide et incompétente.
B. Les personnes qui créent les outils de packaging sont sadiques.
C. Le problème est plus compliqué qu’il n’y paraît.
D. La réponse D.

(Vous pouvez répondre en tapotant sur un terminal imaginaire dans votre main, des caméras télépathiques à base de NFT mettront automagiquement vos réponses sur la Blockchain™.)

Réponse au quiz

Pourquoi en est-on là ?

L'équipe de développement de Python est stupide et incompétente.
Les personnes qui créent les outils de packaging sont sadiques.

C. Le problème est plus compliqué qu'il n'y paraît.
La réponse D.

C’est un problème facile à résoudre

On veut juste partager et installer du code.

C’est quand même pas compliqué !

(Regardez ce que fait Rust.)

C’est un problème difficile à résoudre

On veut juste partager et installer du code.

On veut que les modules s’installent facilement, et qu’ils installent leurs dépendances. On veut une gestion de versions intelligente 🧠 mais simple, qui résolve les conflits toute seule 🤯, mais qu’on puisse forcer dans des cas particuliers.

On veut qu’ils soient partagés publiquement et accessibles sur un dépôt centralisé, mais qu’ils fonctionnent également avec du code gardé secret 🔒, sur des serveurs privés. Attention, ça marche avec le même installeur, avec les mêmes API, hein ! D’ailleurs, on peut avoir du code compilé dedans, parce que Python c’est trop lent 🐢. Et que ça fonctionne avec des versions différentes de Python, sur des OS différents. Sur du matériel différent aussi 🖥️, bien sûr. Forcément, on inclut des binaires pré-compilés, mais on veut aussi que ça se compile à la volée si besoin.

Je vous avais dit qu’il y avait des exécutables dedans ? Ils utilisent directement des bibliothèques binaires installées sur le système, peu importe le système d’exploitation ! Oui, même sur les architectures gros-boutistes ! D’ailleurs, les dépendances sont calculées à la volée sur la machine où le paquet est installé, c’est pratique… C’est comme quand je fais des jeux 🕹, j’ai des images 🖼️ et des sons 🎵 inclus, et les shaders des modèles 3D sont compilés à l’installation selon la carte graphique utilisée, et recompilés à la volée si la carte change. Bien sûr, selon la carte on installe les dépendances particulières nécessaires, c’est un besoin basique pour moi.

Ce qui serait vraiment bien, c’est que quand je crée mes wheels 🎡 je puisse faire de la cross-compilation distribuée. Les fichiers à inclure sont automatiquement ceux de mon gestionnaire de versions, d’ailleurs. L’envoi du paquet sur PyPI va chercher le mot de passe chiffré sur mon disque, gère tranquillou l’authentification à deux facteurs, et m’informe s’il y a un problème avec les droits que j’ai sur le serveur pour ce paquet. Ça me fait facilement une belle page de présentation du projet ✨, avec des liens normalisés pour accéder au code, aux tickets et à la documentation 📘. Attention, je veux pas faire du HTML non plus, faudrait faire un truc avec du markup, ou un truc comme ça. Mais avec un peu de style quand même ✏️. Et des images. Et des badges en SVG 🏷️ qui montrent le taux de couverture des tests 📈, et les versions de Python supportées.

C’est quand même pas compliqué.

Raconte-nous une histoire

Le problème est ancien

Entre la première version publique de Python et la première version publique de Rust sont apparus :

- Windows 3.1, Mac OS X et Linux ;
- Subversion, Mercurial et Git ;
- HTTP/1.0 et TLS,
- HTML, XML, CSS et JavaScript ;
- JPEG, PNG et SVG ;
- Netscape, Internet Explorer, Firefox et Chrome ;
- Google, Amazon et Facebook ;
- le Nokia 3310 et l'iPod.

Ça explique bien des choses…

On manquait cruellement de recul

Python n’a bien sûr pas intégré d’outils pour distribuer le code dès ses débuts :

- `distutils` est intégré dans la bibliothèque standard en 2000,
- le *Python Package Index* (PyPI) est mis en ligne en 2003, développé et maintenu par la communauté Python,
- `setuptools` est annoncé en 2004, puis fusionné avec `distribute` en 2013,
- la *Python Package Authority* (PyPA) est créée en 2011,
- un nombre indécent d’outils, de fichiers et de PEP ont été proposés concernant l’évolution du packaging (et je ne parle pas des gens qui ne font que râler).

Beaucoup d’outils

Vous avez déjà entendu parler de `easy_install`, `pip`, `poetry`, `distutils`, `setuptools`, `distribute`, `flit`, `wheel`, `twine`, et bien d’autres 🔧.

Beaucoup de fichiers

Vous avez déjà entendu parler de `setup.py`, `setup.cfg`, `MANIFEST.in`, `pyproject.toml`, `requirements.txt`, et bien d'autres 📄.

Beaucoup de problèmes

Le développement anarchique 🏴 a entraîné énormément de problèmes : pas de source d'informations sûre pour les personnes qui débutent, énormément de documentations contradictoires, beaucoup de forums obsolètes, beaucoup de code obsolète, des recettes inutiles, des dépôts illisibles 🤓...

En conclusion

C’est le bordel, hein ?

Il nous faut un plan

Quiz

Que faut-il faire pour sortir de là ?

A. Casser la rétrocompatibilité et tout reconstruire de zéro.
B. Créer un nouveau standard qui couvre tous les besoins.
C. Fusionner les outils et les fichiers.
D. Construire une machine à remonter le temps.

Réponse au quiz

Que faut-il faire pour sortir de là ?

A. Casser la rétrocompatibilité et tout reconstruire de zéro.
B. Créer un nouveau standard qui couvre tous les besoins.
C. Fusionner les outils et les fichiers.
D. Construire une machine à remonter le temps.

Tous les besoins, vraiment ?

En réfléchissant bien, en fait, on ne va pas couvrir tous les besoins. Utiliser du code spécifique à chaque paquet pour faire des paquets, c'était une mauvaise idée 👎.

Mais c'est parfois nécessaire 🤔.

Établissons une méthode d’installation

1. On regarde comment construire la wheel du paquet.
2. On installe ce qu’il faut pour construire la wheel du paquet.
3. On récupère récursivement les dépendances du paquet.
4. On construit les wheels du paquet et de ses dépendances.
5. On installe les wheels.

Écrivons des PEP

PEP 427
: On définit un format commun, les wheels, qui s'installe très facilement, sans exécution de code spécifique.

PEP 517
: On établit une architecture de paquets sources qui est indépendante de l'outil utilisé pour construire les paquets.

PEP 518
: On spécifie un nouveau fichier, `pyproject.toml`, qui indique comment le paquet est construit.

PEP 621
: On unifie les métadonnées dans `pyproject.toml`.

PEP 660
: On gère les installations de paquets permettant l'édition.

Créons des outils de base

Adieu `distutils` 👋.

`setuptools` reste un standard dans les faits, mais son équipe de développement a rendu obsolète l'exécution directe de `setup.py`.

`wheel` existe depuis un moment pour apporter à `setuptools` une gestion des wheels. `build` est créé pour apporter une implémentation de référence pour la création de wheels.

L'installateur historique reste `pip`. `installer` est créé pour apporter une implémentation de référence pour l'installation de wheels.

L'implémentation de référence pour publier les paquets sur PyPI est `twine`.

Créons des outils avancés

`setuptools` continue de supporter `setup.py` pour la définition dynamique de métadonnées et l'exécution de code lors de la création de paquets.

D'autres outils sont créés pour créer des paquets : `flit` et `poetry` sont les plus connus. Ils gèrent simplement les cas simples, correspondant à la grande majorité des paquets.

Le résultat

On peut faire désormais faire quelque chose de simple.

Ce n’est pas dans le futur. C’est maintenant.

Les limites, les critiques

On sera pour longtemps obligés de conserver le fonctionnement précédent à cause de la rétrocompatibilité. Mais ces changements nous permettent tout de même d'avancer ⏩.

Nous avons un nouveau fichier, qui dépend d'un nouveau format.

Même avec toutes ces précautions, certaines choses seront cassées 🚧.

Il faudra faire évoluer tous les paquets, toutes les documentations, toutes les mentalités.

Et après ?

Beaucoup d'outils stockent ou souhaitent stocker leur configuration dans `pyproject.toml`. N'hésitez pas à les encourager et à les aider !

Tant que tout le monde suit les spécifications, il est très utile d'avoir des outils variés pour des utilisations variées.

Il est important d'informer tout le monde de ces changements. La communication est difficile 😁.

Il est également important d'accompagner celles et ceux qui subissent ces changements, et doivent adapter leurs outils à ces évolutions.

Et enfin…

J’ai encore beaucoup de choses à apprendre, mais j’ai vu une présentation qui m’a expliqué les bases du packaging. L’histoire est chaotique 🤯, mais c’est plus facile et plus marrant qu’avant ! Ça me donne confiance 😎, j’ai envie de faire des projets et de les partager 🎁.

– Vous !

Pour aller plus loin

Vous pouvez poser des questions !

Beaucoup d'informations ont été compilées dans une suite d'articles qui existe en français et en anglais. Vous y trouverez beaucoup d'informations et nombre de liens vers des PEP, des vieilles discussions sur des mailing-lists, des outils géniaux ou obscurs, des podcasts, des XKCD…

Pour celles et ceux qui aiment les vidéos, des conférences ont été faites à PyConFr 2017 et PyConUS 2021. Certaines choses ont évolué depuis, mais on y apprend beaucoup de choses toujours d'actualité.

Et si vous aimez ce sujet et ce que vous avez appris, n'hésitez pas à soutenir CourtBouillon 💜 !