Package Managers

Antioch & David

https://tinyurl.com/2019gpipackages

Types of Package Managers

System Specific

- Ensure that applications play nice with each other
- Apt (advanced package tool)
  - sudo apt install
  - apt show
  - apt list
  - sudo apt update
  - apt-cache search
  - sudo apt upgrade

More System Specific

- dnf (Red Hat)
  - Improved version of Yum
  - 15 years younger than apt
  - Simpler and more fully featured than apt
- Guix/nix
  - Similar release time as dnf
  - More feature we'll talk about later
- Flatpak and Snap
  - Red Hat and Canonical, respectively
  - Aim to fix fragmentation

App Stores

- Apps are standalone
  - Think snap/flatpak
- Duplicate dependencies D:
  - Lots of library code
  - Dynamic delivery

Language Specific

- RubyGems
  - Cryptographic signing
- Pip, pip3
  - Easy virtual environments
- Npm
  - Automated security audits
- Perl Package Manager

Demo: Pip on Andrew

Dependency Management

Motivation

- Antioch writes an awesome sorting function, AntiochSort().

Motivation

- Antioch writes an awesome sorting function, AntiochSort().
- I want to use his sorting function in my own project, HasheTree(), so I download his code and copy it into my project repository.

Some problems with this approach

- What if Antioch publishes a better version of AntiochSort? How do I know to update?

Some problems with this approach

- What if Antioch publishes a better version of AntiochSort? How do I know to update?
- What if Kimberly creates her own project, KimberTree, that also depends on AntiochSort? Now anyone who installs both HasheTree and KimberTree has duplicate copies of AntiochSort!

Better Approach

- Both HasheTree and KimberTree somehow **declare** a dependency on AntiochSort
- Some external system manages dependencies, and knows to install exactly one copy of AntiochSort that is available to both.
- This external system periodically checks for and installs updates to AntiochSort.

Historical Context

- The first approach was taken by early Linux systems, but it quickly got out of hand.
- Debian Linux developed **apt**, the advanced package tool, to try to solve the dependency management problem.

Problems with apt-like package managers

- Often not easy to install both AntiochSort 1.1 and AntiochSort 1.2 at the same time
  - Maybe KimberTree needs a new feature from 1.2, but HasheTree relies on undocumented behavior from 1.1

Problems with apt-like package managers

- What if the power goes out during an update?
  - System can be left in a partially-updated state that may not be recoverable

Solution: Purely Functional Package Managers with Atomic Updates

Guix

NixOS

Purely Functional Package Management

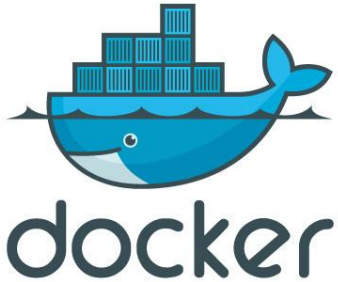
- All packages declare their exact set of dependencies.
- If anything changes, then the package must be rebuilt.
- A package is just a **function** of its source code and dependencies*.

*and package definition, which defines how to build the package and which options to enable

Implementation Strategy

- /gnu/store holds all packages, current and old
- /run/<timestamp> holds the set of packages that were installed at <timestamp>
  - these are symlinks -> /gnu/store
- /run/current holds the set of packages currently installed
  - this is a symlink to /run/<timestamp>

Package managers vs containers



A container is a single binary blob containing all of the dependencies of a certain program.

Containers also ignore most of the theoretical issues with package managers. As a result, they are unbelievably ugly in theory.

However, they are very common because writing container definitions is significantly easier than writing package definitions, and because using containers is really nice in practice.

I recommend using containers at internships and jobs, but do be aware that they have significant problems.

```
(define-public hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                  ".tar.gz"))
              (sha256
               (base32
                "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
    (build-system gnu-build-system)
    (synopsis "Hello, GNU world: An example GNU package")
    (description
     "GNU Hello prints the message \"Hello, world!\" and then exits.  It
serves as an example of standard GNU coding practices.  As such, it supports
command-line arguments, multiple languages, and so on.")
    (home-page "https://www.gnu.org/software/hello/")
    (license gpl3+)))
```

https://guix.gnu.org/blog/2018/a-packaging-tutorial-for-guix/

Security

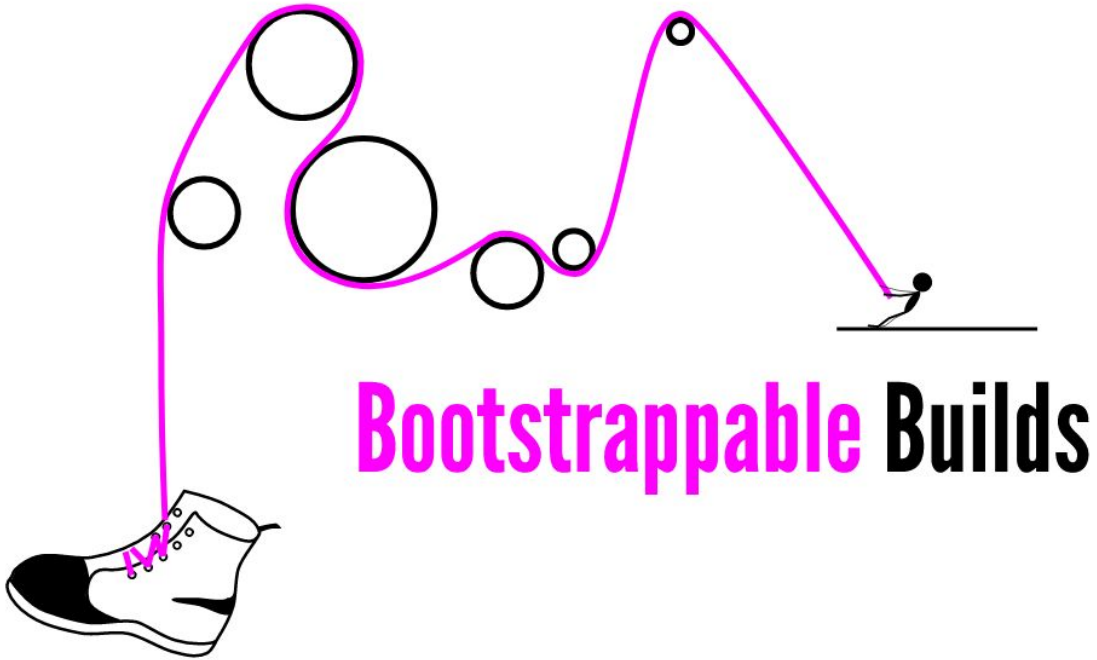
Reflections on Trusting Trust

Theoretical attack discovered by Ken Thompson: backdoor a compiler so that it discovers when it is compiling itself and reinserts the backdoor.

Implications: Bootstrapping & Auditing

- Carefully minimize the amount of binary code necessary to rebuild your package distribution from source.
- Manually audit any remaining binary code to check for backdoors.

https://bootstrappable.org/



Binary Package Distributions

- It is more efficient to compile code once on a central server and then distribute the resulting binary packages, then it is to distribute source packages that must be compiled on every machine.
- But what if someone tampers with your server or network connection?

Implications: Reproducible Builds

- If a package is bit-for-bit identical every time we compile it, then we can audit our servers for tampering by building locally and comparing.
- This is trickier than it sounds! Compilation is not always deterministic (especially if parallel) and many tools insert build timestamps.

https://reproducible-builds.org/

Security Vulnerabilities in Purely Functional Package Managers

- Almost all packages depend on a C standard library (usually glibc).
- Whenever the inputs of a package change, we must rebuild the package.
- So if a vulnerability is discovered in glibc, we have to rebuild everything!
  - This could takes days

Implications: Grafting

- Poke a hole in the purely functional abstraction
- We **know** that a security patch won't change the **interface** of glibc, so we tell the package manager that the old and new versions of glibc should be considered equivalent.
- We only have to rebuild glibc and then we substitute it into all other packages.

https://guix.gnu.org/manual/en/html_node/Security-Updates.html

Guix

left-pad

- A 17-line npm (javascript) package to pad strings to the left with whitespace
- A significant fraction of the entire npm repository depended on left-pad
- It was removed by its author, breaking many packages

Implications: Maintainer-controlled vs Author-controlled packages

Some package managers give software program authors control over packaging their own programs. This is most common with language-specific package managers, like pip for Python and npm for Node.js.

- When authors control their own packages, they are typically more up-to-date
- When maintainers control the packages, they are typically more stable