Functional Package and Configuration Management with GNU Guix

David Thompson

Wednesday, January 20th, 2016

About me

- GNU project volunteer
- GNU Guile user and contributor since 2012
- GNU Guix contributor since 2013
- Day job: Ruby + JavaScript web development / “DevOps”

Overview

- Problems with application packaging and deployment
- Intro to functional package and configuration management
- Towards the future
- How you can help

User autonomy and control

It is becoming increasingly difficult to have control over your own computing:

- GNU/Linux package managers not meeting user needs
- Self-hosting web applications requires too much time and effort
- Growing number of projects recommend installation via `curl | sudo bash` [1] or otherwise avoid using system package managers
- Users unable to verify that a given binary corresponds to the source code

[1] `http://curlpipesh.tumblr.com/`

User autonomy and control

"Debian and other distributions are going to be that thing you run Docker on, little more." [2]

[2] "ownCloud and distribution packaging"
http://lwn.net/Articles/670566/

User autonomy and control

This is very bad for desktop users and system administrators alike. We must regain control!

What's wrong with Apt/Yum/Pacman/etc.?

- Global state (`/usr`) that prevents multiple versions of a package from coexisting
- Non-atomic installation, removal, upgrade of software
- No way to roll back
- Nondeterminstic package builds (though this is changing!)
- Reliance on pre-built binaries provided by a single point of trust
- Requires superuser privileges

The problem is bigger

Proliferation of language-specific package managers and binary bundles that complicate secure system maintenance.

Web applications

Web applications are particularly painful.

Web applications

It's common for today's web applications to require two or more package managers to get all dependencies.

Web applications

Integrating a web application packaged only for a language-specific manager into a system package manager proves difficult. NodeJS is particularly frightening. [3]

[3] "Let's Package jQuery: A Javascript Packaging Dystopian Novella" http://dustycloud.org/blog/javascript-packaging-dystopia/

Web applications

There's a growing number of popular web applications (Hadoop, Chef Server, Cloudera, etc.) that no one knows how to build from source! [4]

[4] "Your big data toolchain is a big security risk!"
http://www.vitavonni.de/blog/201504/2015042601-big-data-toolchains-are-a-security-risk.html

Deployment

How do we automate application deployment without going crazy?

Chef/Puppet/Ansible/etc. are pretty good, right?

Building on top of mainstream package managers and distros yields an unstable foundation.

Problems with configuration management software

- Imperative config management makes is overly-complex and brittle (idempotence is hard)
- More reliable builds require spawning new machines and building from scratch each time (sledgehammer)
- Made primarily for developers for server maintenance, but all users could benefit

Docker?

Surely Docker addresses these issues?

Docker?

I'm afraid not.

Problems with Docker

- Still imperative (though resulting images are immutable)
- Dockerfile DSL is not expressive
- Promotes one disk image per application to cover up underlying package management mess [5]
- No provenance
- Image layering is an ineffective caching strategy
- Does not compose (containers are only one important use-case)

[5] "The sad state of sysadmin in the age of containers" http://www.vitavonni.de/blog/201503/2015031201-the-sad-state-of-sysadmin-in-the-age-of-containers.html

Problems with Docker

- Reliance on DockerHub binaries proves to be insecure [6]

[6] "Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities"
`http://www.banyanops.com/blog/analyzing-docker-hub/`

Well that was pessimistic

Computers are hard. Maybe we should just farm potatoes instead.

Meet GNU Guix

Guix is the functional package management tool for the GNU system.

It is based on the pioneering work of the Nix project. [7]

[7] http://nixos.org/nix/

What does “functional” mean?

“Functional” in this context means treating package builds as functions, in the mathematical sense.

```
emacs = f(gcc,make,coreutils,...)
```

Functional package management

Benefits:

- Build reproducibility
- Atomic upgrades and roll backs
- No single point of trust
- Unprivileged package management
- Multiple variants of the same software may coexist

Functional package management

The complete dependency graph is captured, precisely, down to the bootstrap binaries.

No SAT solver or other complex algorithm for dependency resolution.

Functional package management

To view package builds this way, Guix performs builds in an isolated container in which only the specified dependencies are accessible.

This maximizes build reproducibility.

Reproducible builds

Reproducible builds produce bit-identical binaries when performed multiple times under the same conditions.

Allows for independent verification that a given binary corresponds to its alleged source code.

Why?

WRITEME

Mention reproducible-builds.org

Demo

guix package

guix challenge

Hacking

Guix is made to be maximally hackable, taking inspiration from Emacs.

We seek to intentionally blur the line between user and developer.

Choice of language

Guix is rather special in its choice of implementation language.

Philosophy

It's better to extend an existing language for package recipes and configuration files rather than making a new, domain-specific one.

Embedded vs. External DSLs

Using an extensible programming language as a host has several advantages compared to external DSLs:

- No new parser, interpreter/compiler, editor tools, etc. to maintain
- Access to all available libraries of the host language
- Extensions to the host language can be used as a library by others

Not all general-purpose programming languages are suitable for embedding new languages, [8] so which did we choose?

[8] "How to be a good host: miniKanren as a case study"
`https://www.youtube.com/watch?v=b9C3r3dQnNY`

Guile Scheme

- GNU Guile is a Scheme implementation and the official extension language of the GNU project
- It's a great choice for EDSLs because of Scheme's hygienic macro system
- It's a great choice for Guix because purely functional programming is well-supported in Scheme

Guile goes with everything

Guix uses Guile for nearly everything:

- Initial RAM disk
- Init system (GNU Shepherd, formerly GNU dmd)
- Package recipes (including build scripts!)
- Command line tools
- Low-level POSIX/Linux utilities (such as `call-with-container`)

Guix as a library

- Guix is a big collection of Guile modules
- Packages are first-class Scheme objects
- Anyone can use Guix as a library to write new Guile programs that manipulate package recipes, create new user interfaces (like a web UI), etc.

Example package recipe

WRITEME

Demo

build package at the REPL in Emacs

Other user interfaces

Demo Emacs UI, web prototype

Importing packages from elsewhere

guix import

guix refresh

Demo

Import a package from PyPI

Development environments

guix environment

Full-system configuration

WRITEME

Example system configuration

WRITEME

Demo

guix system vm

Project status

WRITEME

Future

WRITEME

Join us!

We need interested hackers to help us:

- Add new packages
- Upgrade existing packages
- Write system services
- Improve the UI
- Add new tools
- Translate to new languages
- Maintain the web site
- Other stuff!

Join us!

We are currently collecting donations via the FSF to purchase new servers for our build farm!

`https://gnu.org/software/guix/donate/`

Join us!

Chat with us in the `#guix` channel on Freenode or on the `guix-devel@gnu.org` and `help-guix@gnu.org` mailing lists.

Thank you!

Visit `https://gnu.org/software/guix` for source code, documentation, past talks, etc.

Questions?

Legal