Reproducible and Customizable Deployments with GNU Guix

Ludovic Courtès

FOSDEM 2016
The difficulty of keeping software environments under control.
#1. Upgrades are hard.
Distribution Upgrade of all the files:

WARNING

Following the upgrade instructions found in the release notes is the best way to ensure that your system upgrades from one major Debian release to another (e.g. from lenny to squeeze) without breakage!

These instructions will tell you to do a dist-upgrade (instead of upgrade) in the case of apt-get or full-upgrade (instead of safe-upgrade in the case of aptitude) at least once. So you would have to type something like

```
# aptitude full-upgrade
```
4.1. Preparing for the upgrade
   4.1.1. Back up any data or configuration information
   4.1.2. Inform users in advance
   4.1.3. Prepare for downtime on services
   4.1.4. Prepare for recovery
   4.1.5. Prepare a safe environment for the upgrade

4.2. Checking system status
   4.2.1. Review actions pending in package manager
   4.2.2. Disabling APT pinning
   4.2.3. Checking packages status
   4.2.4. The proposed-updates section
   4.2.5. Unofficial sources

4.3. Preparing sources for APT
   4.3.1. Adding APT Internet sources
   4.3.2. Adding APT sources for a local mirror
   4.3.3. Adding APT sources from optical media

4.4. Upgrading packages
   4.4.1. Recording the session
   4.4.2. Updating the package list
   4.4.3. Make sure you have sufficient space for the upgrade
   4.4.4. Minimal system upgrade
   4.4.5. Upgrading the system

4.5. Possible issues during upgrade
   4.5.1. Dist-upgrade fails with “Could not perform immediate configuration”
   4.5.2. Expected removals
   4.5.3. Conflicts or Pre-Depends loops
   4.5.4. File conflicts
   4.5.5. Configuration changes
   4.5.6. Change of session to console
#2. Stateful system management is intractable.
```bash
$DISTRO

down
apt-get update

state 1_a

$DISTRO

down
apt-get update

state 1_b
```
apt-get update
state 1
apt-get install foo
state 2
apt-get update
state 1
apt-get remove bar
state 2
$DISTRO$

$DISTRO$

apt-get update

state 1a

apt-get install foo

state 2a

apt-get remove bar

state 3a

apt-get update

state 1b

apt-get remove bar

state 2b

apt-get install foo

state 3b
$DISTRO$

apt-get update

state 1

apt-get install foo

state 2

apt-get remove bar

state 3

apt-get update

state 1

apt-get remove bar

state 2

apt-get install foo

state 3

= ?
#3. It’s worse than this.
Application-level package managers

- Anaconda - a package manager for Python
- Assembly - a partially compiled code library for use in Common Language Infrastructure (CLI) deployment, versioning and security.
- Biicode - a file-focused dependency manager for C/C++ languages and platforms (PC, Raspberry Pi, Arduino).
- Bower - a package manager for the web.
- UPT - a fork of Bower that aims to be a universal package manager, for multiple environments and unlimited kind of package
- Cabal - a programming library and package manager for Haskell
- Cargo - a package manager for Rust (programming language)
- CocoaPods - Dependency Manager for Objective-C and RubyMotion projects
- Composer - Dependency Manager for PHP
- CPAN - a programming library and package manager for Perl
- CRAN - a programming library and package manager for R
- CTAN - a package manager for TeX
- DUB - a package manager for D
As of `npm@2.6.1`, the `npm update` will only inspect top-level packages. Prior versions of `npm` would also recursively inspect all dependencies. To get the old behavior, use `npm --depth 9999 update`, but be warned that simultaneous asynchronous update of all packages, including `npm` itself and packages that `npm` depends on, often causes problems up to and including the uninstallation of `npm` itself.

To restore a missing `npm`, use the command:

```
curl -L https://npmjs.com/install.sh | sh
```
Giving up?
Giving up?

→ “app bundles” (Docker images)
“Debian and other distributions are going to be that thing you run docker on, little more.”

— Jos Poortvliet, ownCloud developer

http://lwn.net/Articles/670566/
It’s also that thing you run inside Docker!
<table>
<thead>
<tr>
<th>Docker Images</th>
<th>Manage Images</th>
<th>Filter</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Image</th>
<th>Size</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruby:latest</td>
<td>722 mb</td>
<td>17</td>
</tr>
<tr>
<td>python:latest</td>
<td>689 mb</td>
<td>13</td>
</tr>
<tr>
<td>golang:latest</td>
<td>725 mb</td>
<td>14</td>
</tr>
</tbody>
</table>

```
ADD file:e5a3d20748c5d3dd5fa11542d5fa4ef8b72a0bb78ce09f6da18855f3160 125 mb

CMD "/bin/bash"
0 bytes
```

```
RUN apt-get update && apt-get install -y --no-install-recommends ca-certificates curl wget && rm -rf /var/lib/apt/lists/*
44 mb

RUN apt-get update && apt-get install -y --no-install-recommends bazaar git mercurial openssh-client subversion procps
123 mb
```
Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities

Docker Hub is a central repository for Docker developers to pull and push container images. We performed a detailed study on Docker Hub images to understand how vulnerable they are to security threats. Surprisingly, we found that more than 30% of images in official repositories are highly susceptible to a variety of security attacks (e.g., Shellshock, Heartbleed, Poodle, etc.). For general images – images pushed by docker users, but not explicitly verified by any authority – this number jumps up to ~40% with a sampling error bound of 3%.
Functional package management.
gimp = f(gtk+, gcc, make, coreutils)

where f = ./configure && make && make install
\[ gimp = f(gtk+, gcc, make, coreutils) \]
\[ gtk+ = g(glib, gcc, make, coreutils) \]
gimp = f(gtk+, gcc, make, coreutils)
gtk+ = g(glib, gcc, make, coreutils)
gcc = h(make, coreutils, gcc_0)
...

gimp = $f$(gtk+, gcc, make, coreutils)
gtk+ = $g$(glib, gcc, make, coreutils)
gcc = $h$(make, coreutils, gcc$_0$)
...
the complete DAG is captured
$ guix build hello

isolated build: chroot, separate name spaces, etc.
$ guix build hello
/gnu/store/ h2g4sf72... -hello-2.10

hash of all the dependencies
$ guix build hello
/gnu/store/ h2g4sf72... -hello-2.10

$ guix gc --references /gnu/store/...-hello-2.10
/gnu/store/...-glibc-2.22
/gnu/store/...-gcc-4.9.3-lib
/gnu/store/...-hello-2.10
$ guix build hello
/gnu/store/ h2g4sf72... -hello-2.10

$ guix gc --references /gnu/store/...-hello-2.10
/gnu/store/...-glibc-2.22
/gnu/store/...-gcc-4.9.3-lib
/gnu/store/...-hello
(nearly) bit-identical for everyone
$ guix package -i gcc-toolchain coreutils sed grep
...

demo

$ eval 'guix package --search-paths'
...

$ guix package --manifest=my-software.scm
...
Want to get started hacking on GIMP?
Want to get started hacking on GIMP?

A simple matter of installing the deps, right?
$ guix environment --container gimp
...

$ guix environment --container gimp
   --ad-hoc git autoconf automake gdb
...

Whole-system deployment.
GuixSD: declarative OS config
Linux-libre
Linux-libre

initial RAM disk
Linux-libre

initial RAM disk

Guile
Linux-libre

initial RAM disk

PID 1: GNU Shepherd services...
Linux-libre

initial RAM disk

PID 1: GNU Shepherd
services...

Guile

Guile
Linux-libre

initial RAM disk

PID 1: GNU Shepherd
services...

applications
Trustworthiness.
Debian’s dirtiest secret:
Binary packages built by developers are used in the archive

— Lucas Nussbaum, FOSDEM 2015
alice@foo$ guix package --install=emacs
The following package will be installed:
  emacs-24.5 /gnu/store/...-emacs-24.5

The following files will be downloaded:
/gnu/store/...-emacs-24.5
/gnu/store/...-libxpm-3.5.10
/gnu/store/...-libxext-1.3.1
/gnu/store/...-libxaw-1.0.11
alice@foo$ guix package --install=emacs
The following package will be installed:
  emacs-24.5 /gnu/store/...-emacs-24.5

The following files will be downloaded:
  /gnu/store/...-libxext-1.3.1
  /gnu/store/...-libxaw-1.0.11
The following derivations will be built:
  /gnu/store/...-emacs-24.5.drv
  /gnu/store/...-libxpm-3.5.10.drv
(define foo (package ...))
(define foo (package ...))

test

guix build foo
/gnu/store/...-foo-1.0
(define foo (package ...))

test

guix build foo
/gnu/store/...-foo-1.0

git push
git.sv.gnu.org
(define foo (package ...))

test

guix build foo
/gnu/store/...-foo-1.0

git push

user

hydra.gnu.org
build farm

pull

git.sv.gnu.org

pull
(define foo (package ...))

user

hydra.gnu.org
build farm

get binary

pull

git.sv.gnu.org

pull

git push

guix build foo
/gnu/store/...-foo-1.0

test
(define foo (package ...))

test

guix build foo
/gnu/store/...-foo-1.0

git push
git.sv.gnu.org

user

git pull
(define foo (package ...))

guix build foo
/gnu/store/...-foo-1.0

test

git push
git.sv.gnu.org

no "maintainer uploads"

no single point of trust

user
(define emacs (package ...)) /gnu/store/...-emacs-24.5
towards greater user control

1. Bit-reproducible builds

2. No single binary provider

3. Tools for users to challenge binaries
towards greater user control

1. Bit-reproducible builds
   ▶ we have isolated build environments!
   ▶ ... but we need builds to be deterministic
   ▶ http://reproducible-builds.org

2. No single binary provider

3. Tools for users to challenge binaries
towards greater user control

1. Bit-reproducible builds
   - we have isolated build environments!
   - ... but we need builds to be deterministic
   - http://reproducible-builds.org

2. No single binary provider
   - guix publish
   - P2P publishing over GNUnet? (GSoC 2015)

3. Tools for users to challenge binaries
towards greater user control

1. Bit-reproducible builds
   ▶ we have **isolated build environments**!
   ▶ ... but we need builds to be **deterministic**
   ▶ http://reproducible-builds.org

2. No single binary provider
   ▶ guix publish
   ▶ P2P publishing over GNUnet? (GSoC 2015)

3. Tools for users to challenge binaries

-gnu/store/...-openssl-1.0.2d contents differ:
  local hash: 0725122...
  http://hydra.gnu.org/...-openssl-1.0.2d: 0725122...
  http://guix.example.org/...-openssl-1.0.2d: 1zy4fma...

-gnu/store/...-git-2.5.0 contents differ:
  local hash: 00p3bmr...
  http://hydra.gnu.org/...-git-2.5.0: 069nb85...
  http://guix.example.org/...-git-2.5.0: 0mdqa9w...

-gnu/store/...-pius-2.1.1 contents differ:
  local hash: 0k4v3m9...
  http://hydra.gnu.org/...-pius-2.1.1: 0k4v3m9...
  http://guix.example.org/...-pius-2.1.1: 1cy25x1...
Status.
timeline

- Nov. 2012 — dubbed GNU
- Jan. 2013 — 0.1
- ... 
- Apr. 2014 — 0.6, signed binaries, guix system 
- July 2014 — 0.7, installable operating system 
- ... 
- 29 Jan. 2015 — 0.8.1, ARMv7 port
- ... 
- 5 Nov. 2015 — 0.9.0, new service framework, etc.
- Jan. 2016 — successful fundraiser for new build farm
status

- full-featured package manager
- 3,000+ packages, 4 platforms
- **Guix System Distribution**\(^{\beta}\)
- binaries at [http://hydra.gnu.org](http://hydra.gnu.org)
- tooling: auto-update, “linting”, etc.
- $\approx 25$ contributors per month
- ... and lots of friendly people!
- $\approx 400$ commits per month
- $200–500$ new packages per release
install the distribution

use it, report bugs, add packages

help with the **infrastructure** + admin

donate hardware/money

share your ideas!