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 and maintainer-uploaded binaries (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 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 (what about the host?)

[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]

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%.

[6] http://www.banyanops.com/blog/analyzing-docker-hub/

Well that was pessimistic

Computers are hard. Let's just look at cat pictures, 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/

Meet GuixSD

GuixSD is the GNU/Linux distribution that uses Guix as its package manager.

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
- No single point of trust
- Unprivileged package management
- Atomic upgrades and roll backs
- 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.

Build results are immutable.

This maximizes build reproducibility.

Reproducible builds

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

Requires fixing issues in upstream build systems that are nondeterministic.

Why?

"With reproducible builds, multiple parties can redo this process independently and ensure they all get *exactly* the same result. We can thus gain confidence that a distributed binary code is indeed coming from a given source code." [8]

[8] `https://reproducible-builds.org/`

Use cases

Transparent

Guix is a source-based package manager, but will transparently download pre-built binaries from a trusted party, if available.

Otherwise, it will simply build from source.

Decentralized

In Guix, there is no central point of trust for receiving pre-built binaries (substitutes).

Decentralized

Guix provides `http://hydra.gnu.org`, but it is optional.

Users may authorize zero or more substitute servers, or even publish their own substitutes for others to use via `guix publish`.

Challenge authority

When builds are reproducible, users may challenge their substitute providers by building locally and comparing the results.

Unprivileged

Users can build and install software without root privileges.

Unprivileged

Each user may have one or more “profiles”, a union of many packages.

Use cases:

- Eva and Ben use different versions of Emacs
- Eva hacks on 2 Ruby projects that require different versions

Atomic

Package installation/removal and full-system updates are atomic operations, meaning that either the operation succeeds, or nothing happens.

Roll back

Any package transaction may be rolled back, likewise for full-system upgrades.

If a full-system update goes wrong, just boot into the previous working generation!

Coexistence

Each package has its own unique directory in the store that contains its build artifacts.

You can have every version of Ruby, Python, and Perl under the sun and that's OK!

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 programming 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, [9] so which did we choose?

[9] "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.

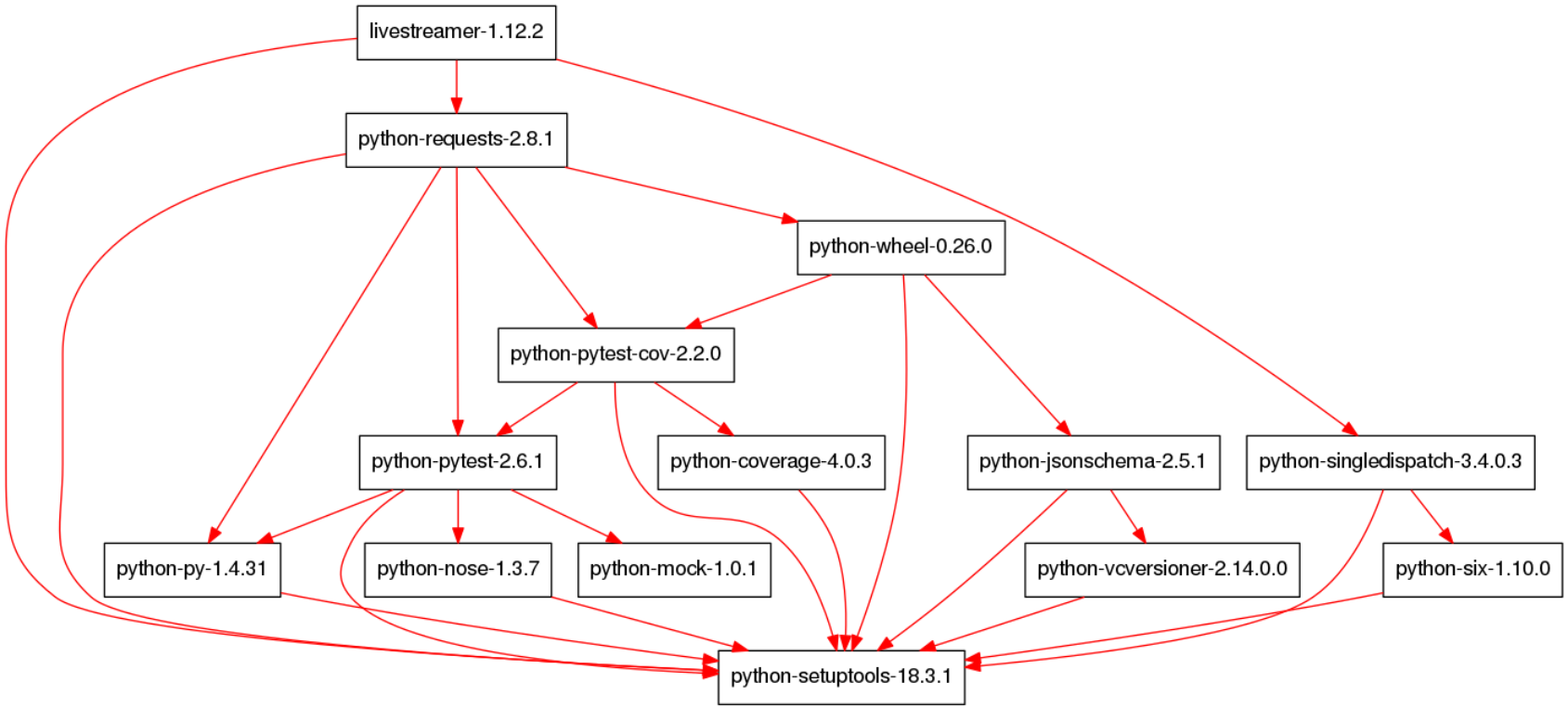
Packages are first-class Scheme objects.

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

```
(define-public livestreamer
  (package
    (name "livestreamer")
    (version "1.12.2")
    (source (origin
              (method url-fetch)
              (uri (string-append
                    "https://github.com/chrippa/livestreamer/archive/v"
                    version ".tar.gz"))
              (file-name (string-append "livestreamer-" version ".tar.gz"))
              (sha256
               (base32
                "1fp3d3z2grb1ls97smjkraazpxnvajda2d1g1378s6gzmda2jvjd"))))
    (build-system python-build-system)
    (arguments
     '(#:tests? #f)) ; tests rely on external web servers
    (native-inputs
     `(("python-setuptools" ,python-setuptools)))
    (propagated-inputs
     `(("python-requests" ,python-requests)
       ("python-singledispatch" ,python-singledispatch)))
    (synopsis "Internet video stream viewer")
    (description "Livestreamer is a command-line utility that extracts streams
from various services and pipes them into a video playing application.")
    (home-page "http://livestreamer.io/")
    (license license:bsd-2)))
```

Dependency graph



Demo!

Emacs + Geiser

Other user interfaces

Demo Emacs UI, web prototype

Importing packages

The `guix import` tool that can automatically generate code snippets for packages found in foreign systems.

Supported systems include: PyPI, RubyGems, CPAN, Hackage, ELPA, and CRAN.

Auto-updating

The `guix refresh` tool can automatically find the latest release of certain software.

For example, Python packages can be updated by querying PyPI for information on the latest release.

Demo!

```
guix import
```

Reproducible development environments

Getting the dependencies needed to create development environments can be tough.

Many languages invent their own solution, but this is a general problem.

Reproducible development environments

Guix has a tool for this: `guix environment`

Think of it like a language-agnostic version of Python's `virtualenv`.

Reproducible development environments

Environments can be purified via standard environment variables or, for better isolation, Linux containers.

This allows developers to have confidence that potential contributors will be able to build their software.

Demo!

```
guix environment
```

Full-system configuration

The Guix System Distribution supports a consistent whole-system configuration mechanism.

All aspects of a system configuration are declared in a single place.

Advantages

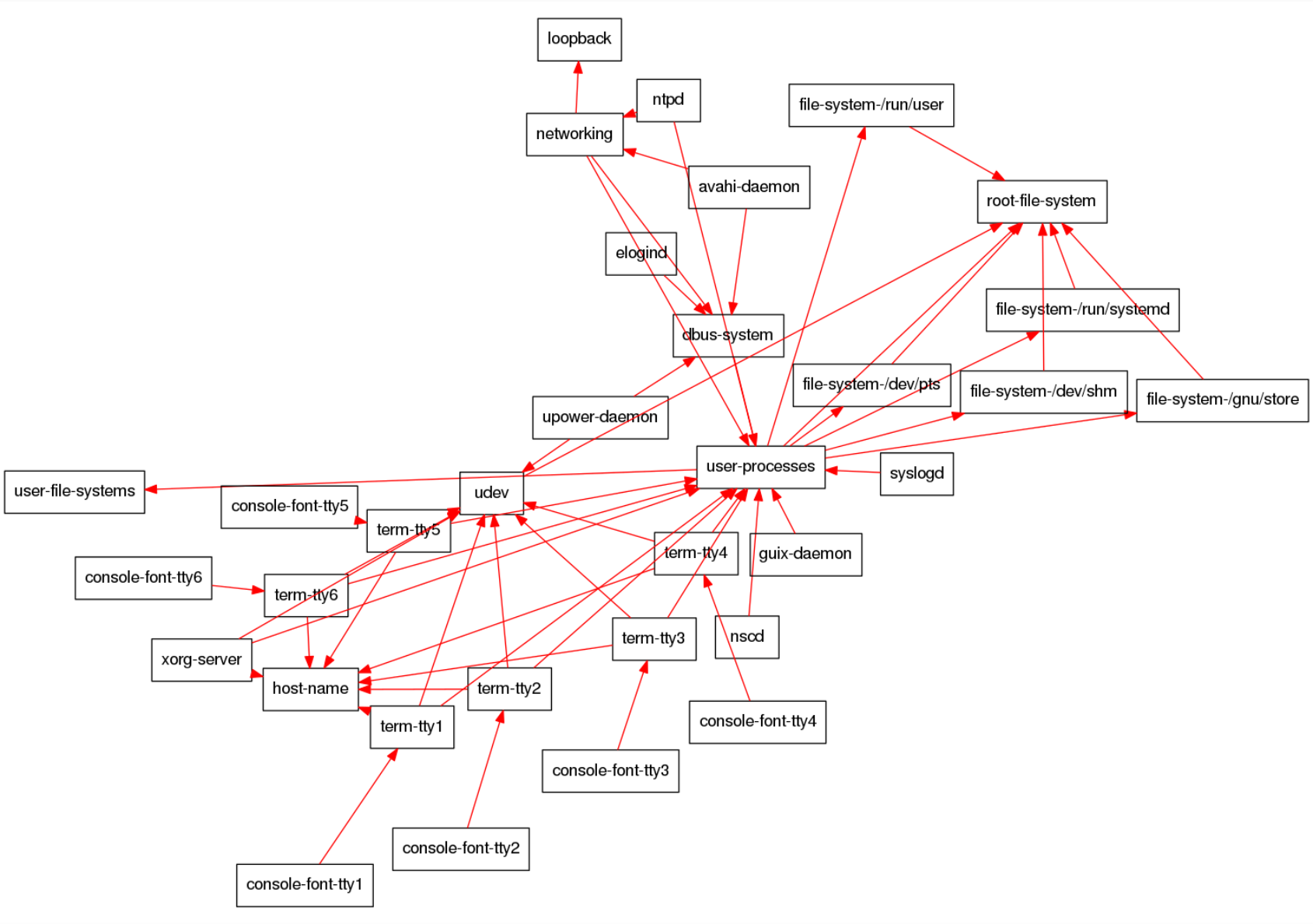
Easy to replicate configuration on different machines without resorting to additional tools layered on top.

System upgrades are atomic and can be rolled back.

Example system configuration

```
(operating-system
  (host-name "izanagi")
  (timezone "America/New_York")
  (locale "en_US.UTF-8")
  (bootloader (grub-configuration (device "/dev/sda")))
  (file-systems (cons (file-system
                        (device "root")
                        (title 'label)
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (list (user-account
                (name "dave")
                (comment "David Thompson")
                (group "users")
                (supplementary-groups '("wheel" "netdev" "audio"
                                        "video" "cdrom"))
                (home-directory "/home/dave"))))
  (packages (cons* adwaita-icon-theme avahi dbus gnome-terminal
                   htop less man-db nss-certs openssh pulseaudio
                   wicd unzip rsync xfce
                   %base-packages))
  (services %desktop-services)
  (name-service-switch %mdns-host-lookup-nss))
```

Service graph



Demo!

```
guix system vm
```

Project status

- Full-featured package manager
- 3,000 packages, 4 platforms
- Guix System Distribution in beta
- Binaries at `http://hydra.gnu.org`
- tooling: auto-update, “linting”, etc.

Project status

In a Nutshell, GNU Guix...

... has had 10,244 commits made by 79 contributors
representing 164,213 lines of code

... is mostly written in Scheme
with a very well-commented source code

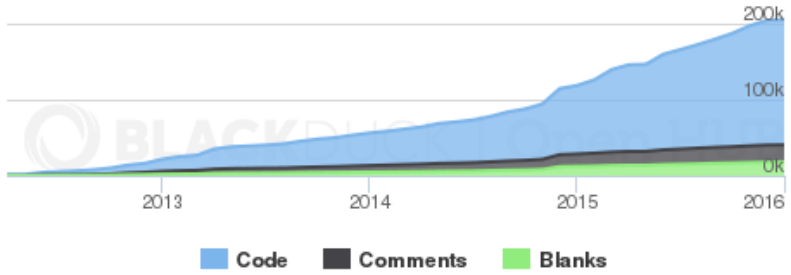
... has a codebase with a long source history
maintained by a very large development team
with increasing Y-O-Y commits

... took an estimated 43 years of effort (COCOMO model)
starting with its first commit in April, 2012
ending with its most recent commit 1 day ago

Languages

Scheme	85%	C++	8%
Emacs Lisp	5%	7 Other	2%

Lines of Code



Activity

30 Day Summary
Dec 18 2015 — Jan 17 2016

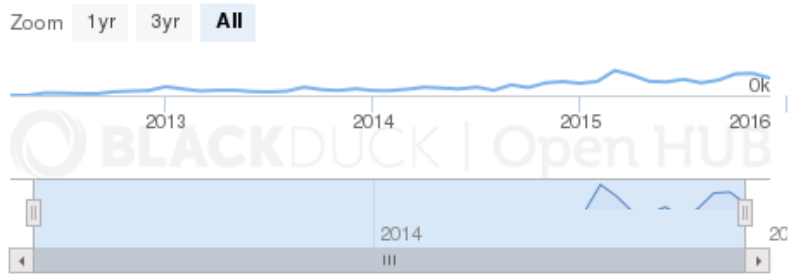
300 Commits
19 Contributors
including 2 new contributors

12 Month Summary
Jan 17 2015 — Jan 17 2016

5290 Commits
Up + 2672 (102%) from previous 12 months

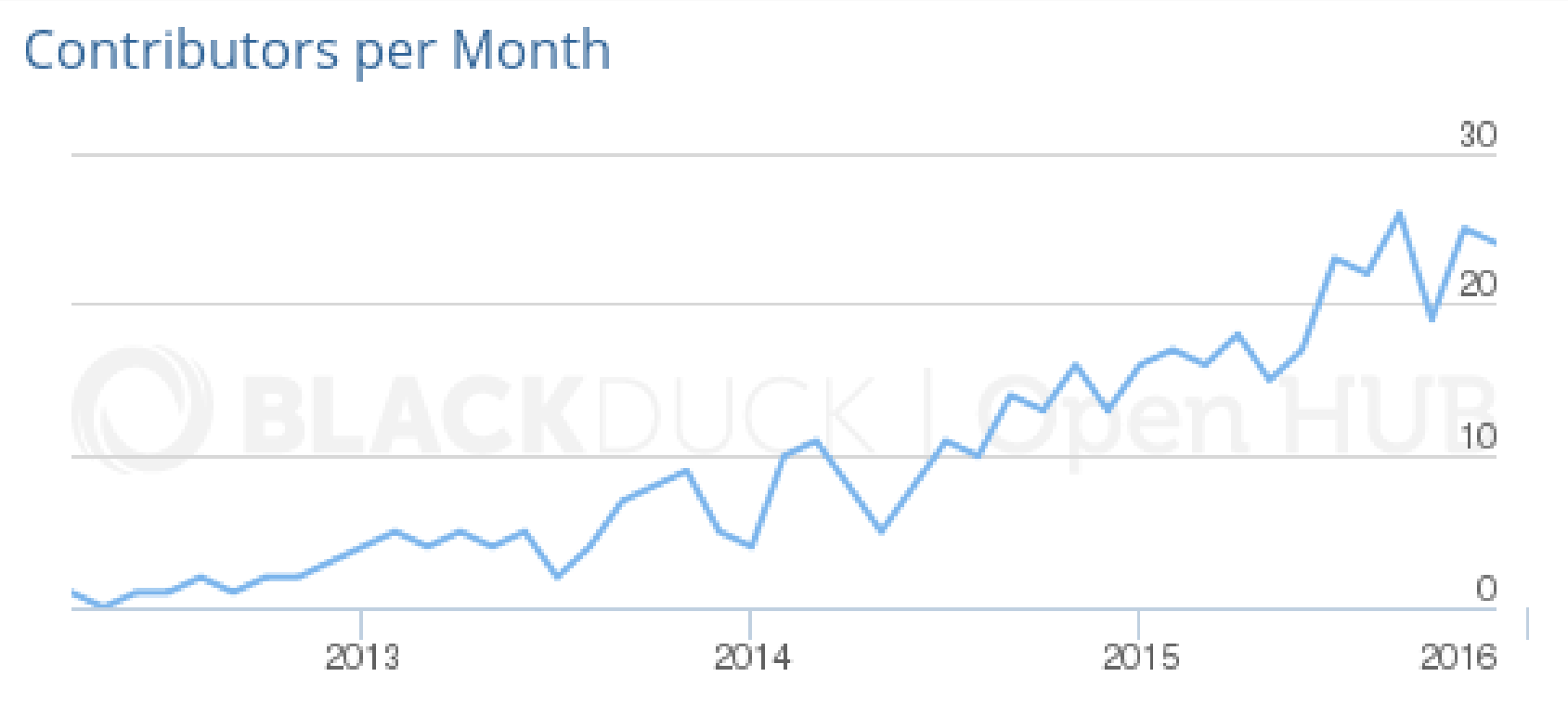
60 Contributors
Up + 24 (66%) from previous 12 months

Commits per Month



https://www.openhub.net/p/gnuguix

Project status



The people have spoken

Project status

≈200–500 new packages per release. More needed!

Future

I intend to focus on:

- A cluster deployment tool: `guix deploy`
- Improved support for GuixSD containers

Future

More generally:

- Stronger build farm
- More packages that are reproducible
- GNOME
- LVM
- Encrypted root

Join us!

- Use Guix on top of your existing distro
- Use the distribution
- Add new packages or upgrade existing ones
- Write system services
- Add new translations
- Tell us your ideas!

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