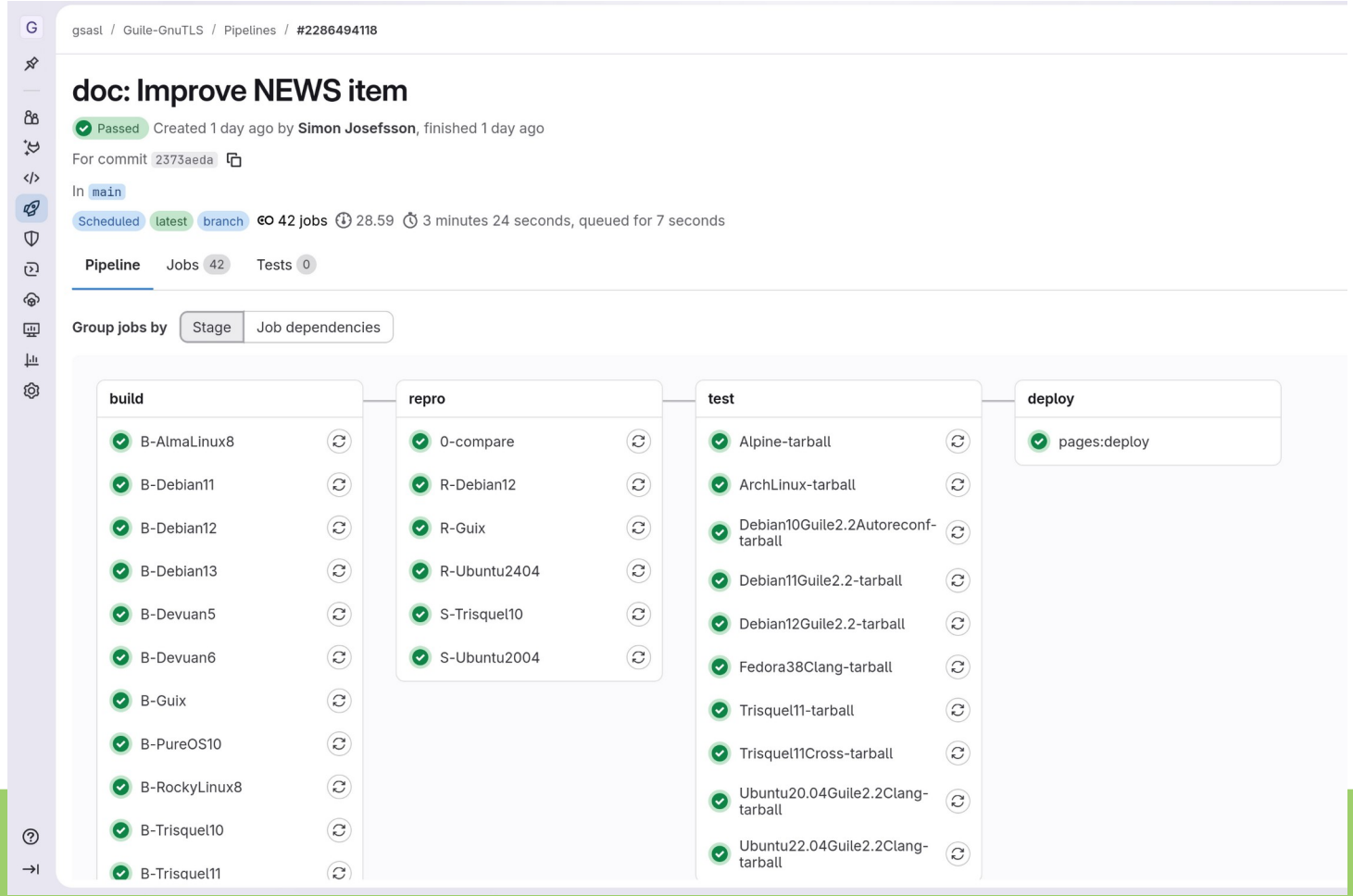
Guix Container Images

and what you can do with them

Simon Josefsson <simon@josefsson.org>
2026-01-31 FOSDEM'26 Belgium

B1D2BD1375BECB784CF4F8C4D73CF638C53C06BE

I like Continous compilation & testing



Inspiration

From: Ludovic Courtès
Subject: Building a Docker image for GitLab-CI
Date: Tue, 13 Feb 2024 11:31:28 +0100

Hello Guix!

Has anyone succeeded in building a Docker image suitable for use in GitLab-CI? I haven't. Here's what I tried.

Initially, I built an image with 'guix system image -t docker …' but that doesn't work because then the image's "entry point" is shepherd, but shepherd never returns. Thus, GitLab-CI would spawn the image and eventually time out.

So I tried this instead:

```
  guix pack guix bash-minimal coreutils-minimal grep net-base \
    --save-provenance -S /bin=bin -S /share=share -S /etc=etc \
    -f docker --max-layers=100
```

...

Declaring v1.0

From: Simon Josefsson
Subject: Re: Building a Docker image for GitLab-CI
Date: Wed, 25 Dec 2024 21:38:14 +0100

All,

Here are some updates about Guix container images for GitLab pipelines or local podman usage. I'm declaring this v1.0.

tl;dr: https://gitlab.com/debdistutils/guix/container

Final images are built from a pure Guix container now. Everything is done on public shared GitLab runners in the pipeline, no container uploads. Stage0 creates Debian+Guix that builds a pure Guix stage1 which builds the final Stage2 images. The content of these images appears to be reproducible, but alas the docker images itself aren't: https://issues.guix.gnu.org/75090

...

One year of use

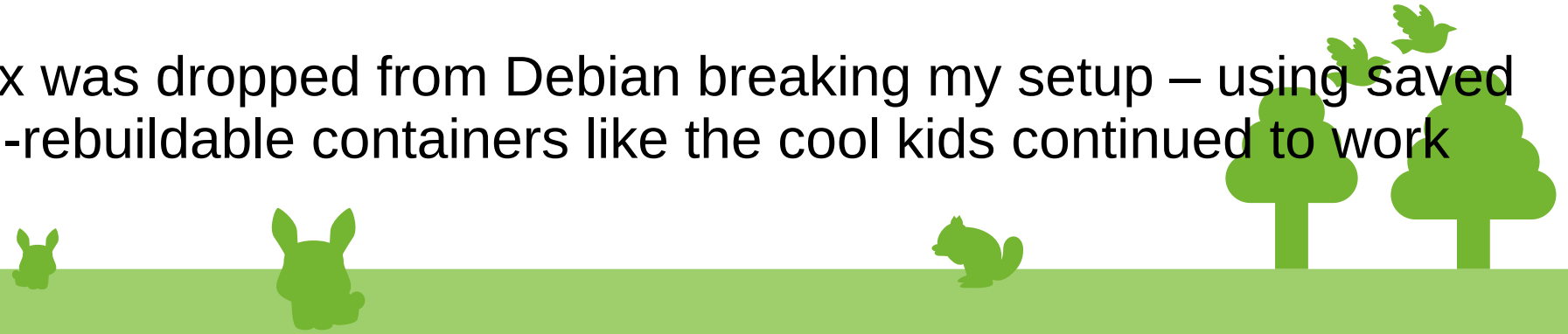
- During 2025, the Guix Container Images was integrated into GitLab CI/CD Pipelines for Libidn, Libidn2, OATH Toolkit, Libtasn1, GNU SASL, InetUtils, Libntlm, Guile-GnuTLS, etc

One year of use

- During 2025, the Guix Container Images was integrated into GitLab CI/CD Pipelines for Libidn, Libidn2, OATH Toolkit, Libtasn1, GNU SASL, InetUtils, Libntlm, Guile-GnuTLS, etc
- The ”**make dist**” tarballs used for releases are bit-by-bit identical and reproducible with tarballs built on GitLab.com from source

One year of use

- During 2025, the Guix Container Images was integrated into GitLab CI/CD Pipelines for Libidn, Libidn2, OATH Toolkit, Libtasn1, GNU SASL, InetUtils, Libntlm, Guile-GnuTLS, etc
- The ”**make dist**” tarballs used for releases are bit-by-bit identical and reproducible with tarballs built on GitLab.com from source
- Guix was dropped from Debian breaking my setup – using saved non-rebuildable containers like the cool kids continued to work



Initial GitLab Build Design

- Initial design overly complex:
  - Stage0: Install Guix in a Debian container, save container
  - Stage1: Build a pure Guix container using previous container
  - Stage2: Build another pure Guix container using previous container
- I thought I could get to reproducibility this way

New Design

- New design - https://gitlab.com/debdistutils/guix/container
- Realized the Debian+Guix containers had standalone use
  - https://hub.docker.com/r/jas4711/debian-with-guix
- For reproducibility testing, having two ”similar” containers with Guix helps
  - Trisquel and Ubuntu with Guix
  - https://hub.docker.com/r/jas4711/guix-on-dpkg
- Take upstream container and **./guix-install.sh && guix pull** and upload resulting container into a registry

New Design

- Stage1: Use Debian+Guix to create pure Guix container
  - pack=$(guix pack $GUIX_PACKS --save-provenance -S /bin=bin -S /share=share -f docker --image-tag=guix –max-layers=8)
  - skopeo --insecure-policy copy --additional-tag $CI_REGISTRY_IMAGE:$CI_JOB_NAME docker-archive://$pack docker://$CI_REGISTRY_IMAGE:$CI_JOB_NAME
- Stage2: Use Trisquel/Ubuntu container to reproduce it
- Test & Deploy to GitLab registry and Docker Hub
  - Amd64, arm64, ppc64el supported – riscv64 exists but not published (no QEMU builds, using real hardware)

How to Use - interactively

```
$ podman run -it –entrypoint=/bin/sh docker.io/jas4711/guix:latest
sh-5.1# guix describe
  guix 2d4ed08
    repository URL: https://git.guix.gnu.org/guix.git
    branch: master
    commit: 2d4ed08662714ea46cfe0b41ca195d1ef845fd1b
sh-5.1# exit
```

```
# .gitlab-ci.yml
test-amd64-latest-wget-configure-make-libksba:
  image: registry.gitlab.com/debdistutils/guix/container:latest
  before_script:
  - groupadd --gid 0 root
  - useradd --uid 0 --gid root --shell /bin/sh --home-dir / --system root
  - cp -rL /gnu/store/*profile/etc/* /etc/
  - groupadd --system guixbuild
  - for i in $(seq -w 1 10); do useradd -g guixbuild -G guixbuild -d /var/empty
-s $(command -v nologin) -c "Guix build user $i" --system guixbuilder$i; done
  - export HOME=/
  - env LANG=C.UTF-8 guix-daemon --build-users-group=guixbuild &
  - guix archive --authorize < /share/guix/ci.guix.gnu.org.pub
  - guix archive --authorize < /share/guix/bordeaux.guix.gnu.org.pub
  - guix describe
  - guix install libgpg-error
  - GUIX_PROFILE="//.guix-profile"
  - . "$GUIX_PROFILE/etc/profile"
  script:
  - wget https://www.gnupg.org/ftp/gcrypt/libksba/libksba-1.6.7.tar.bz2
  - tar xfa libksba-1.6.7.tar.bz2
  - cd libksba-1.6.7
  - ./configure
  - make V=1
  - make check VERBOSE=t V=1
```

sendmail.mc deja vu

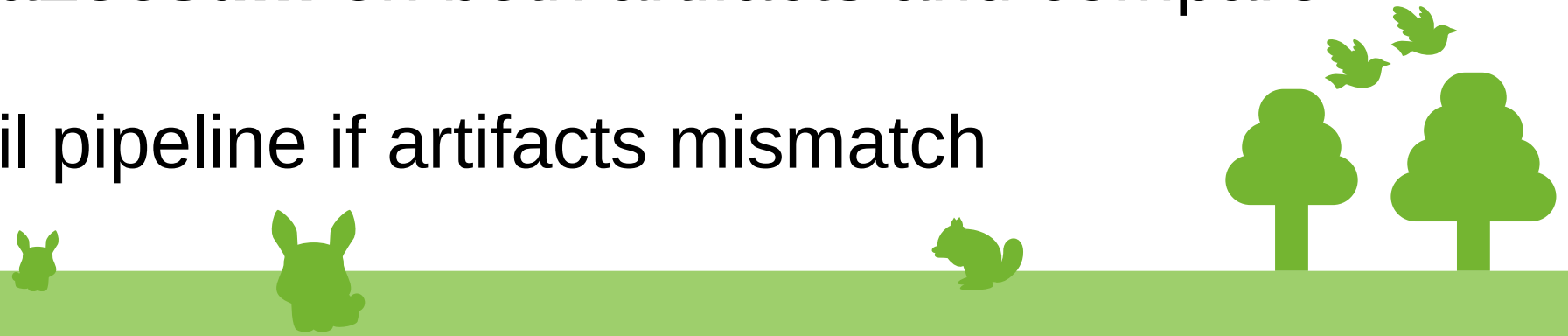
- groupadd --gid 0 root
- useradd --uid 0 --gid root --shell /bin/sh --home-dir / --system root
- cp -rL /gnu/store/*profile/etc/* /etc/
- groupadd --system guixbuild
- for i in $(seq -w 1 10); do useradd -g guixbuild -G guixbuild -d /var/empty -s $(command -v nologin) -c "Guix build user $i" --system guixbuilder$i; done
- export HOME=/
- env LANG=C.UTF-8 guix-daemon --build-users-group=guixbuild &
- guix archive --authorize < /share/guix/ci.guix.gnu.org.pub
- guix archive --authorize < /share/guix/bordeaux.guix.gnu.org.pub

Hide things or not?

- How come everything you do has already been done before?
- MetaCall Guix produce Guix containers on GitHub since 2019
- Uses a custom script as container entry-point:
  - https://github.com/metacall/guix/blob/master/scripts/entry-point.sh
- Good inspiration for my effort – would be useful to compare goals and design in detail

What to use Guix containers for? Reproducible tarballs!

- Define two GitLab CI/CD jobs that builds your project and run ’make dist’
- Define another GitLab CI/CD job that run **sha256sum** on both artifacts and compare
- Fail pipeline if artifacts mismatch



What to use Guix containers for? Reproduce tarballs!

- Just because tarballs were reproducible at release time does not mean they can be reproduced later on
- Normally this is not the case… timestamps with day or month
- Using the Guix time-machine inside a Guix container allows you to confirm reproducibility of old tarballs continously
- https://gitlab.com/debdistutils/verify-reproducible-releases
- Thank you Guix time machine!

Security vs Let me do what I want

- Guix's guix-daemon can be run root-less now!
- Except not on GitLab shared runners – no user namespaces
- Ironically the root-less guix-daemon requires use of –cap-add=CAP_SYS_ADMIN,CAP_NET_ADMIN and/or –disable-chroot and/or --security-opt seccomp=unconfined and/or privileged=true runners depending on platform
- Regression compared to Guix v1.4.0
- https://codeberg.org/guix/guix/issues/3917

Thank You!

Questions?