dockercon 23

Reproducible builds with BuildKit for software supply chain security

Akihiro Suda

Software Engineer | NTT

NTT

Background

- Security assessment has been hard for Docker images, due to lack of verifiability in the software supply chain

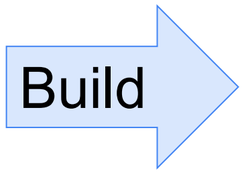
- Even when the source code (Dockerfile) is public, and the source code appears to be harmless, it is hard to prove that the image is actually buildable from the source code

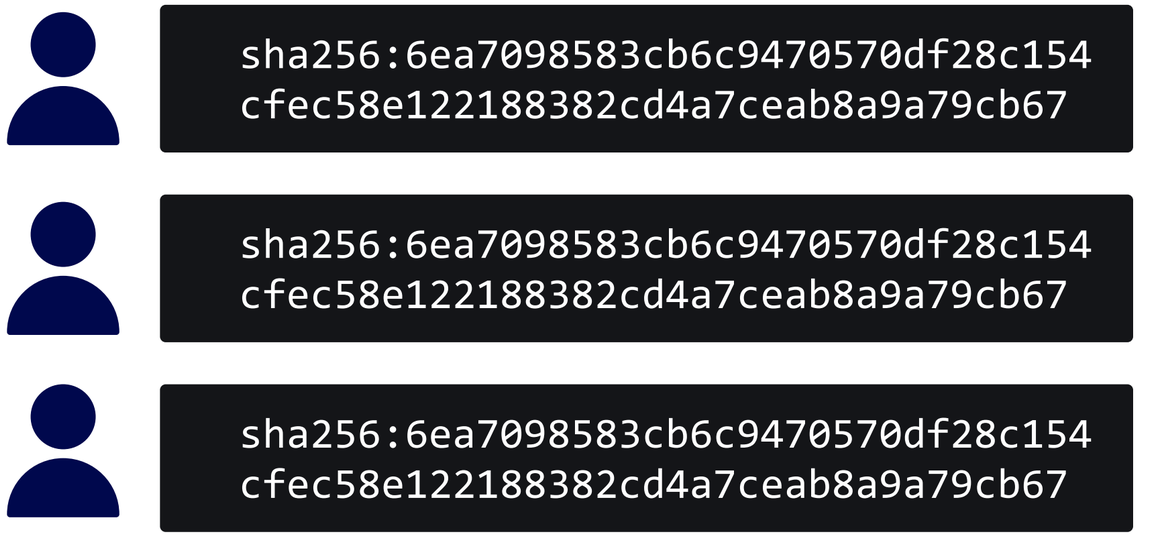
- **Reproducible builds help proving it**
  (But whether the source code is harmless is another topic)

What are Reproducible Builds?

- Same source, same binary
- Attestable by anybody
- Attestable at anytime

```
FROM debian
RUN  apt-get install -y gcc make ...
COPY . .
RUN  make
```





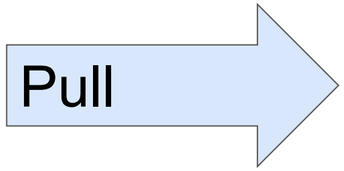
Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources

Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources

```
docker pull  some-image
```

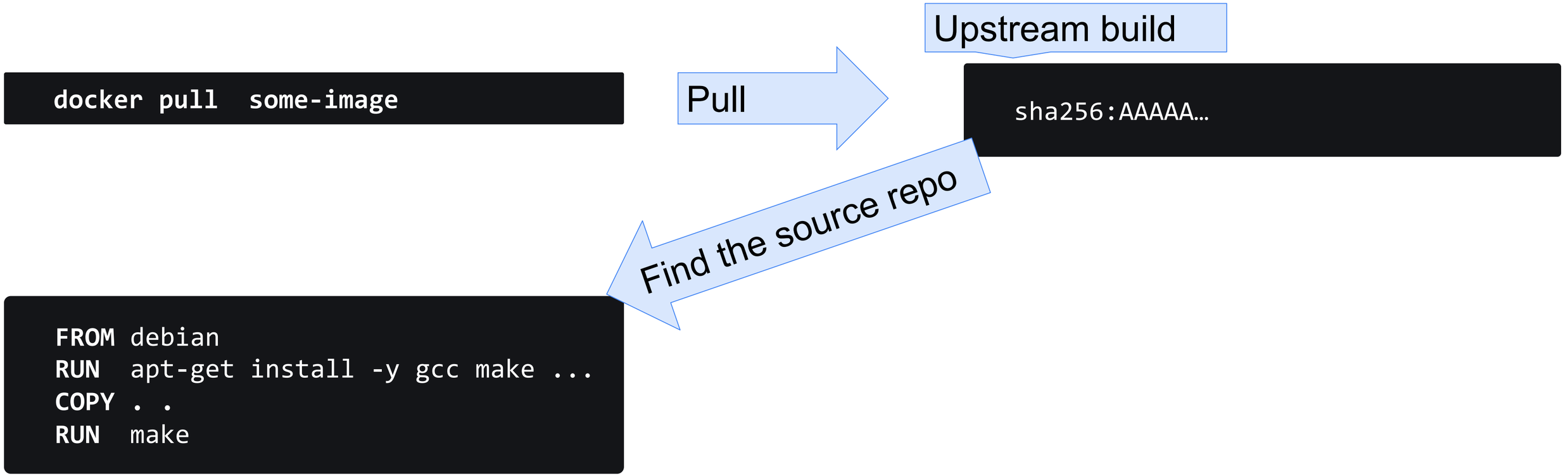


Upstream build

```
sha256:AAAAA…
```

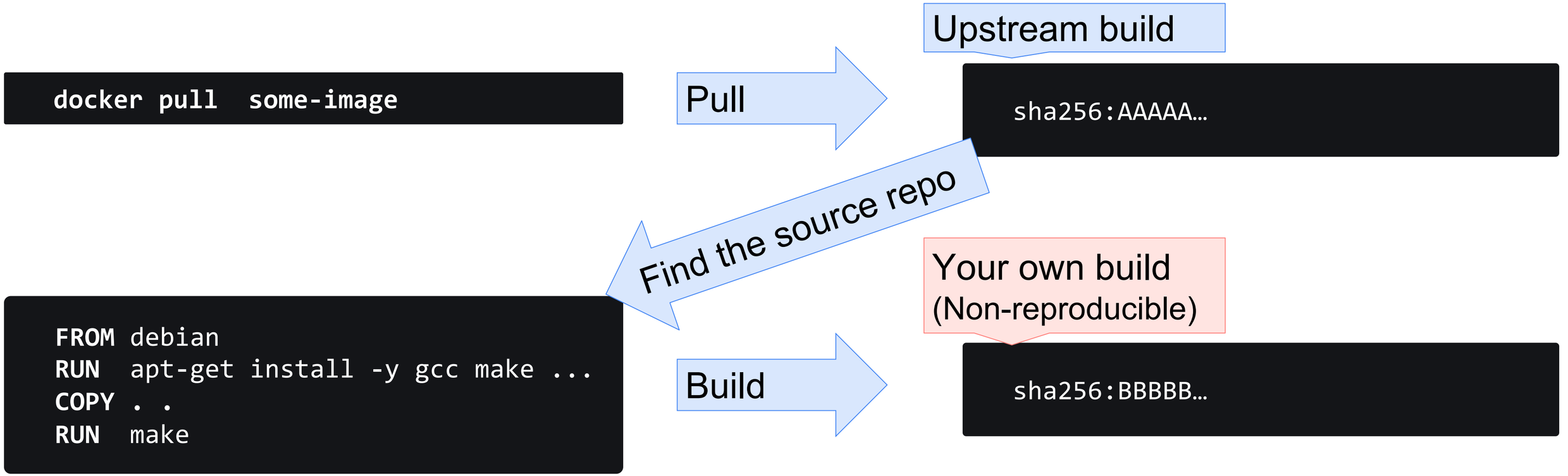
Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources



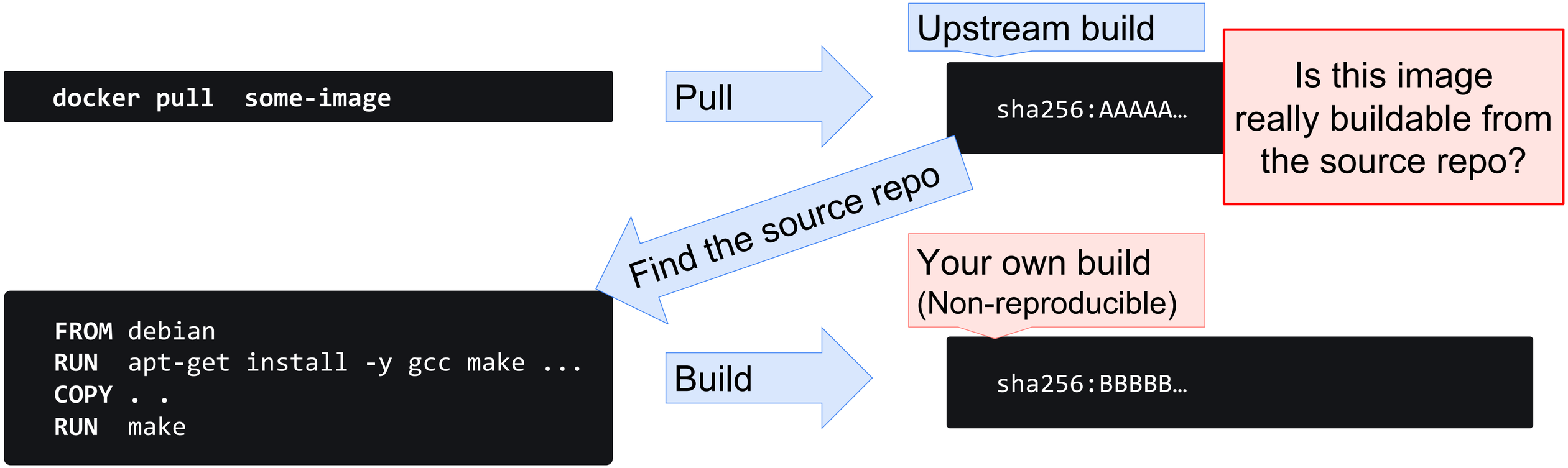
Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources



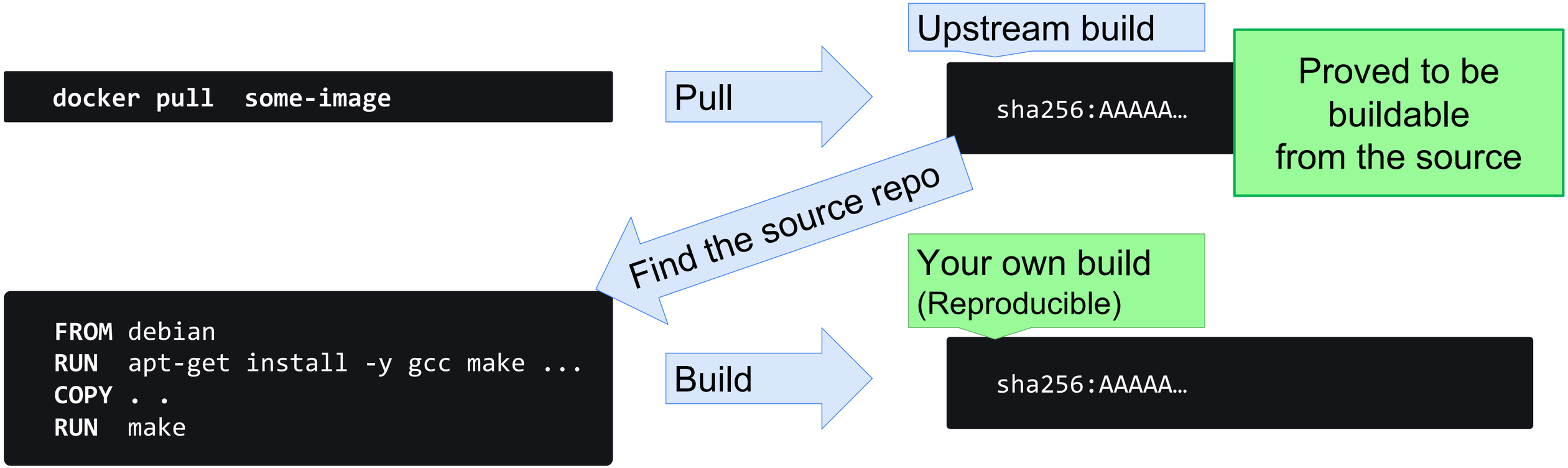
Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources



Why do we need reproducible builds?

Because non-reproducible builds cannot be proved to be buildable from harmless sources



Why do we need reproducible builds?

- Reproducibility per se doesn’t prove any harmlessness

- Non-reproducibility doesn’t prove any harmfulness, either

Why do we need reproducible builds?

- Reproducibility proves that the image is actually buildable from its source
- The source still has to be reviewed
- **The source may still be malicious**

- **But at least the image contains no secret code that you can never review**

Docker Hub images are actually reproducible?

Docker Hub images are actually reproducible?

- No, mostly

Docker Hub images are actually reproducible?

```
$ docker pull golang:1.21.1-alpine@\
  sha256:96634e55b363cb93d39f78fb18aa64abc7f96d372c176660d7b8b6118939d97b

$ DOCKER_BUILDKIT=0 \
  docker build -t my-golang \
  "https://github.com/docker-library/golang.git#\
  585c8c1e705a7a458455f0629922a4f90628ce08:1.21/alpine3.18”

$ go install github.com/reproducible-containers/diffoci/cmd/diffoci@latest

$ diffoci diff docker://golang:1.21.1-alpine docker://my-golang
```

Docker Hub images are actually reproducible?

```
$ docker pull golang:1.21.1-alpine@\
  sha256:96634e55b363cb93d39f78                                    7b

$ DOCKER_BUILDKIT=0 \
  docker build -t my-golang \
  "https://github.com/docker-library/golang.git#\
  585c8c1e705a7a458455f0629922a4f90628ce08:1.21/alpine3.18”

$ go install github.com/reproducible-containers/diffoci/cmd/diffoci@latest

$ diffoci diff docker://golang:1.21.1-alpine docker://my-golang
```

DOCKER_BUILDKIT=0 with Docker 20.10.23 corresponds to the current Docker Hub image (Will change in the future)

Docker Hub images are actually reproducible?

```
$ docker pull golang:1.21.1-alpine@\
  sha256:96634e55b363cb93d39f78fb18aa64abc7f96d372c176660d7b8b6118939d97b

$ DOCKER_BUILDKIT=0 \
  docker build -t my-golang \
  "https://github.com/docker-library/gol
  585c8c1e705a7a458455f0629922a4f90628ce

$ go install github.com/reproducible-containers/diffoci/cmd/diffoci@latest

$ diffoci diff docker://golang:1.21.1-alpine docker://my-golang
```

DiffOCI: diff for Open Container Initiative (OCI) images
https://github.com/reproducible-containers/diffoci

Docker Hub images are actually reproducible?

```
$ diffoci diff docker://golang:1.21.1-alpine docker://my-golang
TYPE     NAME                                                INPUT-0                          INPUT-1
Desc     application/vnd.docker.distribution.manifest.v2+json   b25862...                        3c4eca0...
...
File     etc/ssl/certs/3e45d192.0                            2023-08-09 03:36:47 +0000 UTC    2023-09-21 08:35:31 +0000 UTC
...
(More than 14,000 lines)
...
File     go/                                                 2023-09-06 18:31:40 +0000 UTC    2023-09-21 08:35:45 +0000 UTC
```

Docker Hub images are actually reproducible?

The “--semantic” flag ignores ”boring” differences (timestamps, file ordering, etc.)

```
$ diffoci --semantic diff docker://golang:1.21.1-alpine docker://my-golang
TYPE     NAME                      INPUT-0                                                                  INPUT-1
Layer    ctx:/layers-1/layer       length mismatch (457 vs 454)
Layer    ctx:/layers-1/layer       name "usr/local/share/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "etc/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "usr/share/ca-certificates/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    eef110e...                                                               e9bfe18...
Layer    ctx:/layers-2/layer       length mismatch (13939 vs 13938)
Layer    ctx:/layers-2/layer       name "usr/local/go/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    60e22bb...                                                               67f2648...
Layer    ctx:/layers-3/layer       length mismatch (4 vs 3)
Layer    ctx:/layers-3/layer       name "go/.wh..wh..opq" only appears in input 0
```

Docker Hub images are actually reproducible?

The “--semantic” flag ignores ”boring” differences (timestamps, file ordering, etc.)

“.wh..wh..opq” (AUFS whiteouts) are missing due to the filesystem difference

```
$ diffoci --semantic diff docker
TYPE     NAME                      INPUT-0                                                    INPUT-1
Layer    ctx:/layers-1/layer       length mismatch (457 vs 454)
Layer    ctx:/layers-1/layer       name "usr/local/share/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "etc/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "usr/share/ca-certificates/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    eef110e...                                                 e9bfe18...
Layer    ctx:/layers-2/layer       length mismatch (13939 vs 13938)
Layer    ctx:/layers-2/layer       name "usr/local/go/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    60e22bb...                                                 67f2648...
Layer    ctx:/layers-3/layer       length mismatch (4 vs 3)
Layer    ctx:/layers-3/layer       name "go/.wh..wh..opq" only appears in input 0
```

Docker Hub images are actually reproducible?

The “--semantic” flag ignores ”boring” differences (timestamps, file ordering, etc.)

“.wh..wh..opq” (AUFS whiteouts) are missing due to the filesystem difference

```
$ diffoci --semantic diff docker
TYPE     NAME                      INPUT-0                                                                  INPUT-1
Layer    ctx:/layers-1/layer       length mismatch (457 vs 454)
Layer    ctx:/layers-1/layer       name "usr/local/share/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "etc/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "usr/share/ca-certificates/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    eef110e...                                                               e9bfe18...
Layer    ctx:/layers-2/layer       length mismatch (13939 vs 13938)
Layer    ctx:/layers-2/layer       name "usr/local/go/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    60e22bb...                                                               67f2648...
Layer    ctx:/layers-3/layer       length mismatch (4 vs 3)
```

lib/apk/db/scripts.tar differ due to the timestamp information inside scripts.tar
(the “--semantic” flag isn’t still clever enough to ignore this “boring” difference”)

Docker Hub images are actually reproducible?

The “--semantic” flag ignores ”boring” differences (timestamps, file ordering, etc.)

“.wh..wh..opq” (AUFS whiteouts) are missing due to the filesystem difference

```
$ diffoci --semantic diff docker
TYPE     NAME                      INPUT-0                                                                  INPUT-1
Layer    ctx:/layers-1/layer       length mismatch (457 vs 454)
Layer    ctx:/layers-1/layer       name "usr/local/share/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "etc/ca-certificates/.wh..wh..opq" only appears in input 0
Layer    ctx:/layers-1/layer       name "usr/share/ca-certificates/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    eef110e...                                                               e9bfe18...
Layer    ctx:/layers-2/layer       length mismatch (13939 vs 13938)
Layer    ctx:/layers-2/layer       name "usr/local/go/.wh..wh..opq" only appears in input 0
File     lib/apk/db/scripts.tar    60e22bb...                                                               67f2648...
Layer    ctx:/layers-3/layer       length mismatch (4 vs 3)
```

lib/apk/db/scripts.tar differ due to the timestamp information inside scripts.tar
(the “--semantic” flag isn’t still clever enough to ignore this “boring” difference”)

This image is not fully reproducible, but its non-reproducibility is explainable

(So, this image appears to be actually buildable from the public Dockerfile)

Why are images not reproducible?

- Timestamps
- Version of the base image (“`FROM`” images in Dockerfiles)
- Versions of the packages (apt-get, pip, etc.)
- Others:
  - Filesystem characteristics (e.g., OverlayFS)
  - Ordering of files
  - Randomized `mktemp`, etc.

Timestamps

- The images have timestamps in:
  - the “`created`” property in the OCI Image Config
  - the “`history`” property in the OCI Image Config
  - the “`org.opencontainers.image.created`” annotation in the OCI Index
  - the timestamps of the files in the image layers

OCI = Open Container Initiative

Timestamps

- The images have timestamps in:
  - the “`created`” property in the OCI Image Config
  - the “`history`” property in the OCI Image Config
  - the “`org.opencontainers.image.created`” annotation in the OCI Index
  - the timestamps of the files in the image layers

OCI = Open Container Initiative

Timestamps

- BuildKit (since v0.11) supports rewriting the timestamps for OCI Image Config and OCI Index

Unix epoch (int64, seconds from 1970-01-01 00:00:00 UTC)

```
buildctl build --opt build-arg:SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct)
```

```
docker buildx build --build-arg SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct)
```

- Support was incomplete in v0.11 and v0.12;

  using **v0.13 [beta] is recommended** (see the next couple of slides)

OCI = Open Container Initiative

Timestamps

- The images have timestamps in:
  - the “`created`” property in the OCI Image Config
  - the “`history`” property in the OCI Image Config
  - the “`org.opencontainers.image.created`” annotation in the OCI Index
  - the timestamps of the files in the image layers

OCI = Open Container Initiative

Timestamps

- BuildKit v0.13 [beta] supports rewriting the timestamps in the OCI image layers too

```
buildctl build --opt build-arg:SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct) \
--output type=image,name=example.com/image,push=true,rewrite-timestamp=true
```

```
docker buildx build --build-arg SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct) \
--output type=image,name=example.com/image,push=true,rewrite-timestamp=true
```

- Docs: https://github.com/moby/buildkit/blob/master/docs/build-repro.md

OCI = Open Container Initiative

BuildKit v0.13 is still beta and its CLI is still subject to change until its GA

Timestamps

- The `SOURCE_DATE_EPOCH` arg is also propagated to ”RUN” containers as an environment variable
- The `SOURCE_DATE_EPOCH` env var is recognized by gcc, clang, cmake, and a bunch of other tools to make application binaries reproducible: https://reproducible-builds.org/docs/source-date-epoch/

Pinning the base image

```
FROM debian
```

Pinning the base image

```
FROM debian
```

```
FROM debian:bookworm
```

Pinning the base image

```
FROM debian
```

```
FROM debian:bookworm
```

```
FROM debian:bookworm-20230904
```

Pinning the base image

```
FROM debian
```

```
FROM debian:bookworm
```

```
FROM debian:bookworm-20230904
```

```
FROM debian:bookworm-20230904@sha256:b4042f895d5d1f8df415caebe7c416f9dbcf0dc8867abb225955006de50b21f3
```

Pinning the base image

```
FROM debian
```

```
FROM debian:bookworm
```

```
FROM debian:bookworm-20230904
```

```
FROM debian:bookworm-20230904@sha256:b4042f895d5d1f8df415caebe7c416f9dbcf0dc8867abb225955006de50b21f3
```

apt-get on bookworm-20230904 still installs the latest packages, not the past packages (So, not reproducible)

Pinning packages: Debian and Ubuntu

snapshot.debian.org and snapshot.ubuntu.com keep old packages

```
FROM debian:bookworm-20230904-slim
RUN rm -rf /etc/apt/sources.list* && \
  echo 'deb [check-valid-until=no] http://snapshot.debian.org/archive/debian/20230904T000000Z bookworm main' \
  >/etc/apt/sources.list && \
  echo 'deb [check-valid-until=no] http://snapshot.debian.org/archive/debian-security/20230904T000000Z bookworm-security main' \
  >>/etc/apt/sources.list && \
  echo 'deb [check-valid-until=no] http://snapshot.debian.org/archive/debian/20230904T000000Z bookworm-updates main' \
  >>/etc/apt/sources.list && \
  apt-get update && \
  apt-get install -y gcc
```

Pinning packages: Debian and Ubuntu

Caching is practically necessary, as snapshot servers are slow

repro-sources-list.sh simplifies the Dockerfile, and enables caching dpkg files

```
FROM debian:bookworm-20230904-slim
ADD --chmod=0755 \
  https://raw.githubusercontent.com/reproducible-containers/repro-sources-list.sh/v0.1.0/repro-sources-list.sh \
  /usr/local/bin/repro-sources-list.sh
RUN --mount=type=cache,target=/var/cache/apt \
  repro-sources-list.sh && \
  apt-get update && \
  apt-get install -y gcc
```

More examples at: https://github.com/reproducible-containers/repro-sources-list.sh

Pinning packages: Debian and Ubuntu

- `RUN --mount=type=cache,target=/var/cache/apt` can be saved on GitHub Actions using:

  https://github.com/reproducible-containers/buildkit-cache-dance

```
steps:
  - uses: actions/cache@v3
    with:
      path: var-cache-apt
      key: var-cache-apt-${{ hashFiles('Dockerfile') }}
  - uses: reproducible-containers/buildkit-cache-dance@v2.1.2
    with:
      cache-source: var-cache-apt
      cache-target: /var/cache/apt
```

Pinning packages: Debian and Ubuntu

- The (checksums of the) packages on snapshot.debian.org are signed by Debian, just like regular apt-get repositories
- The signatures are fetched and verified against the package metadata checksums on running `apt-get update` (Not on `apt-get install`)
- If `/var/lib/apt` (metadata) is compromised, `apt-get update` will fail
- If `/var/cache/apt` (dpkg files) is compromised, `apt-get install` will fail
- The situation is same for snapshot.ubuntu.com (signed by Canonical)

Pinning packages: Debian and Ubuntu

- If you don’t trust the latest package signatures, you can reproduce the most of the packages by yourself:

  https://wiki.debian.org/ReproducibleBuilds/Howto

tests.reproducible-builds.org

Debian navigation

Suite/architecture overviews

Tested architectures:

amd64 arm64 armhf i386

Tested suites:

buster bullseye **bookworm** trixie unstable experimental

Test results statistics

Results for **bookworm/amd64**

Unreproducible packages:

with notes

without notes

Other package states:

package sets

Recently tested packages:

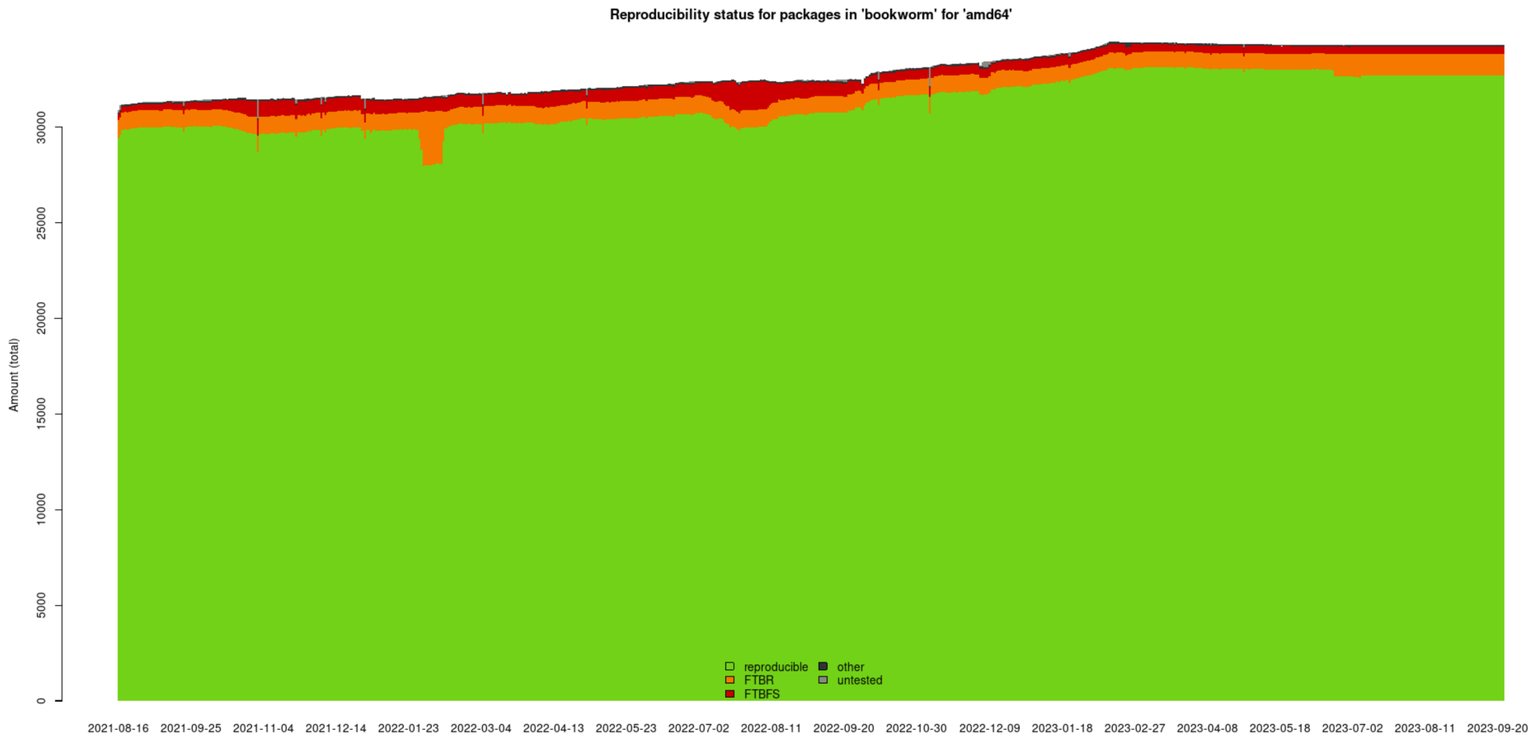
last 24h

Overview of reproducible builds for packages in bookworm for amd64

Reproducibility rate: 95.3%

32694 packages (95.3%) successfully built reproducibly in bookworm/amd64.
1146 packages (3.3%) failed to build reproducibly.
377 packages (1.1%) failed to build from source.

23 packages (0.1%) timed out during the build.
1 (0.0%) source packages had build-depends which could not be satisfied.
59 (0.2%) packages which are neither Architecture: 'any', 'all', 'amd64', 'linux-any', 'linux-amd64' nor 'any-amd64' will not be built here.
0 (0.0%) source packages could not be downloaded.
0 (0.0%) packages have been blacklisted on bookworm/amd64.



https://tests.reproducible-builds.org/debian/bookworm/index_suite_amd64_stats.html (Captured on 2023-09-24)

Pinning packages: NixOS

- Repro build is much easier with NixOS

  (although NixOS per se is often considered to be hard to learn)

- The `flake.lock` file contains the checksums of the sources

- If the binary is present on cache.nixos.org, the cached binary is used;

  otherwise the package is built from the source, with very good reproducibility

  (99.77% for `nixos.iso_minimal.x86_64-linux` installation, according to https://r13y.com/)

Pinning packages: Alpine, Rocky, Alma, etc.

- These distros do not provide snapshot servers like snapshot.debian.org
- You have to preserve `/etc/apk/cache` , `/var/cache/dnf`, etc. by yourself
- Examples can be found at:

  https://github.com/reproducible-containers/repro-pkg-cache
- In the long term, BuildKit frontends *may* have features to help pinning packages: https://github.com/moby/buildkit/issues/4259

Future work (Help wanted)

- Proposal to make well-known images reproducible

  (at least for Debian-based ones)

- ”Single-click” platform for attesting reproducibility and sharing the result

Recap

- Repro builds prove that an image is actually buildable from its source
- Whether the source is harmless or not is another topic

- Ideally, every image should be bit-for-bit reproducible
- Practically, subtle differences can be allowed, **when they are explainable** (e.g., timestamps)

Recap

Tools and examples: https://github.com/reproducible-containers

- diffoci: diff for OCI images, to analyze non-reproducible builds
- repro-sources-list.sh: reproducibility helper for Debian, Ubuntu, etc.
- repro-pkg-cache: reproducibility helper for Alpine, Alma, Rocky, etc.
- buildkit-cache-dance: apt-get cache for GitHub Actions

BuildKit docs: https://github.com/moby/buildkit/blob/master/docs/build-repro.md

OCI = Open Container Initiative

Recap

- Slides will be uploaded to https://github.com/AkihiroSuda

  (README → “Presentation slides”)