arik-grahl.de/talks/cds-london-2026

Beyond Docker Builds

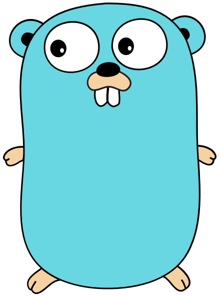
Declarative, Reproducible and Secure OCI Containers with Nix

echo $(whoami)

```yaml
---
apiVersion: v1
kind: Person
metadata:
  name: Arik Grahl
  pronouns: he/him
spec:
  work:
    company: SysEleven
    role: Senior Software Engineer
  contact:
    web: www.arik-grahl.de
    gitlab: gitlab.com/arik.grahl
    github: github.com/arikgrahl
    linkedin: linkedin.com/in/arikgrahl
    mastodon: chaos.social/@arikgrahl
    matrix: @arik:matrix.arik-grahl.de
```

Golang Example Application (1/2)

```go
package main

import (
  "database/sql"
  "io"
  "net/http"

  _ "github.com/mattn/go-sqlite3"
)

func main() {
  db, _ := sql.Open("sqlite3", "weather.db")
  db.Exec(`
    CREATE TABLE IF NOT EXISTS cache (key TEXT PRIMARY KEY, body BLOB, ts DATETIME);
  `)

  http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    // …
  })

  http.ListenAndServe("0.0.0.0:8080", nil)
}
```



Golang Example Application (2/2)

```go
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
  body := []byte{}
  db.QueryRow(`
    SELECT body FROM cache WHERE key = ? AND ts > DATETIME('now', '-1 minute')
  `, r.URL.String()).Scan(&body)
  if len(body) == 0 {
    req, _ := http.NewRequest("GET", "https://wttr.in"+r.URL.String(), nil)
    req.Header.Set("User-Agent", "curl")

    resp, _ := http.DefaultClient.Do(req)
    defer resp.Body.Close()
    body, _ = io.ReadAll(resp.Body)

    db.Exec(`
      INSERT INTO cache(key, body, ts) VALUES (?, ?, CURRENT_TIMESTAMP)
      ON CONFLICT(key) DO UPDATE SET body=excluded.body, ts=CURRENT_TIMESTAMP
    `, r.URL.String(), body)
  }

  w.Write(body)
})
```

Build an Application with Docker: Reproducibility (1/2)

```
$ docker build -t weather:scratch -f Dockerfile .
```

```
[1/2] STEP 1/4: FROM golang:1.25.4-alpine3.22@sha256:d3f0…1bbb                         ✓
[1/2] STEP 2/4: RUN apk --no-cache add gcc musl-dev sqlite-dev                          ✗
[1/2] STEP 3/4: COPY . .                                                                ✓
[1/2] STEP 4/4: RUN CGO_ENABLED=1 go build                                              ⚠
[2/2] STEP 1/5: FROM scratch                                                            ✓
[2/2] STEP 2/5: COPY --from=0 /go/weather /bin/weather                                  ⚠
[2/2] STEP 3/5: COPY --from=0 /lib/ld-musl-x86_64.so.1 /lib/ld-musl-x86_64.so.1         ⚠
[2/2] STEP 4/5: COPY --from=0 /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/        ✓
[2/2] STEP 5/5: ENTRYPOINT ["/bin/weather"]                                             ✓
```

Build an Application with Docker: Reproducibility (2/2)

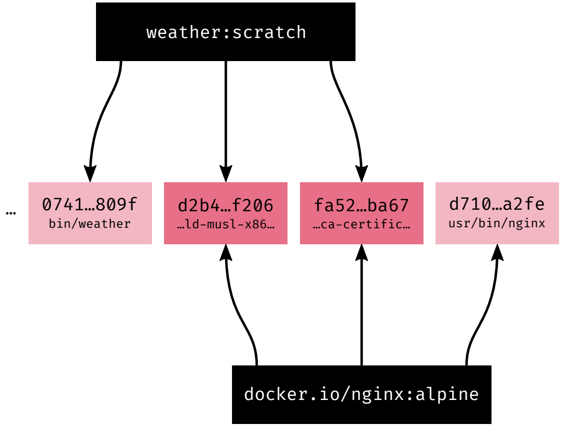
- impure
  - `RUN` with network stack
  - `FROM` without digest
- “it depends”
  - `RUN` of potentially non-reproducible command
  - `COPY` of potentially impure data
- reproducible
  - `FROM` with content addressable hashes
  - `COPY` of static data
  - static configuration such as `ENTRYPOINT`

OCI Format: Reusability of Shared Dependencies (1/2)

```
$ podman save --format=oci-dir --output=. weather:scratch

├─ oci-layout         { "imageLayoutVersion": "1.0.0" }
├─ index.json         { "manifests": [ … ], … }
└─ blobs
    └─ sha256
        ├─ d817…fdca  { "config": { … }, "layers": [ … ], … }
        ├─ 424c…6634  { "architecture": "…", "config": { "Entrypoint": … }, … }
        ├─ 0741…809f  bin/weather
        ├─ d2b4…f206  lib/ld-musl-x86_64.so.1
        └─ fa52…ba67  etc/ssl/certs/ca-certificates.crt
```

OCI Format: Reusability of Shared Dependencies (2/2)

- no reproducibility $\longrightarrow$ no reusability
- for given reproducibility, reuse as much OCI blobs as possible
  - manifest[1] and configuration[2]
    - small data size (`json`)
    - exclusively artifact-specific content
  - layer[3]
    - potentially large data size (`tar+gzip`)
    - artifact-specific (e.g. `bin/weather`)
    - shared dependencies (e.g. `lib/ld-musl-x86_64.so.1`)

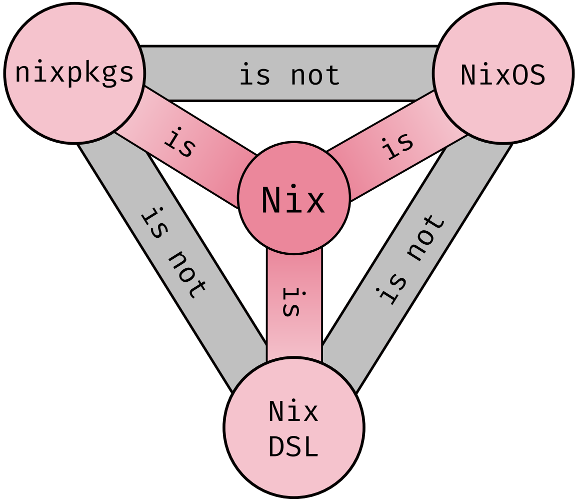


[1] `application/vnd.oci.image.manifest.v1+json`
[2] `application/vnd.oci.image.config.v1+json`
[3] `application/vnd.oci.image.layer.v1.tar+gzip`

Entering Nix

- Nix enables systems, which are
  - reproducible
  - declarative
  - reliable

What is Nix?



- NixOS
  - Linux-based OS on top of Nixpkgs
  - immutable design
  - atomic update model
- Nixpkgs
  - package manager
  - large mono repository
  - collection of over 120k packages
- Nix DSL
  - purely functional
  - lazily evaluated
  - purpose-built

Nix DSL: Purpose-Built

- domain specific language (DSL) for the Nix package manager
  - expressions return a `derivation` representing package payload

```
{ pkgs ? import <nixpkgs> { }, system ? builtins.currentSystem }:
derivation {
  inherit system;
  name = "hello-world";
  version = "0.0.1";
  builder = pkgs.writeShellScript "builder.sh" ''
    echo "echo 'Hello, World!'" > $out
    ${pkgs.coreutils}/bin/chmod +x $out
  '';
}
```

  - library functions for common languages and frameworks
    - `stdenv.mkDerivation`
    - `buildPythonPackage`
    - `buildNpmPackage`
    - `buildDotnetModule`
    - `buildGoModule`
    - `buildRustPackage`
    - `buildFlutterApplication`

Packaging the Example Application: Build a Binary (1/2)

```
{ pkgs ? import <nixpkgs> { } }:
let
  name = "weather";
  version = "0.0.1";
in
pkgs.buildGoModule {
  inherit version;
  pname = name;
  src = ./.;
  vendorHash = "sha256-Swi56SaPh4AN7LZ2a+j3p/jNf/InnbmE6AEErjqLg0g=";
}
```

Packaging the Example Application: Build a Binary (2/2)

```nix
# weather-bin.nix

{ pkgs ? import <nixpkgs> { } }:
let
  name = "weather";
  version = "0.0.1";
in
pkgs.buildGoModule {
  inherit version;
  pname = name;
  src = ./.;
  vendorHash = "sha256-Swi5…Lg0g=";
}
```

```
$ nix build -f weather-bin.nix

$ file $(readlink -f ./result/bin/weather)
/nix/store/iq7l…0lmx-weather-0.0.1/bin/weather:
ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /nix/store/
j193…ispv-glibc-2.40-66/lib/ld-linux-x86-64.so.2,
for GNU/Linux 3.10.0, not stripped

$ ./result/bin/weather &

$ curl -s http://localhost:8080 | head -n 7
Weather report: London

    \  /       Partly cloudy
  _ /"".-.     +10(9) °C
    \_(   ).   ↑ 7 km/h
    /(___(__)  10 km
               0.0 mm
```

Packaging the Example Application: Build an OCI Image (1/2)

```
{ pkgs ? import <nixpkgs> { } }:
let
  name = "weather";
  version = "0.0.1";
  bin = pkgs.buildGoModule {
    inherit version;
    pname = name;
    src = ./.;
    vendorHash = "sha256-Swi56SaPh4AN7LZ2a+j3p/jNf/InnbmE6AEErjqLg0g=";
  };
in
pkgs.dockerTools.buildLayeredImage {
  inherit name;
  tag = "v${version}";
  contents = with pkgs; [ cacert ];
  config.Entrypoint = [ "${bin}/bin/weather" ];
}
```

Packaging the Example Application: Build an OCI Image (2/2)

```nix
# weather-oci.nix

{ pkgs ? import <nixpkgs> { } }:
let
  name = "weather";
  version = "0.0.1";
  bin = pkgs.buildGoModule {
    inherit version;
    pname = name;
    src = ./.;
    vendorHash = "sha256-Swi5…Lg0g=";
  };
in
pkgs.dockerTools.buildLayeredImage {
  inherit name;
  tag = "v${version}";
  contents = with pkgs; [
    cacert
  ];
  config.Entrypoint = [
    "${bin}/bin/weather"
  ];
}
```

```
$ nix build -f weather-oci.nix

$ file $(readlink -f ./result)
/nix/store/79y3…59xm-weather.tar.gz:
gzip compressed data, from Unix,
original size modulo 2^32 16896000

$ docker load < ./result
Loaded image: weather:v0.0.1

$ docker run -d -p 8080:8080 weather:v0.0.1

$ curl -s http://localhost:8080 | head -n 7
Weather report: London

     \  /       Partly cloudy
   _ /"".-.     +10(9) °C
     \_(   ).   ↑ 7 km/h
     /(___(__)  10 km
                0.0 mm
```

Revisiting Reusability of Shared Dependencies (1/2)

- introducing another example container
  - complementing the example application
  - reverse proxy based on Nginx

```
{ pkgs ? import <nixpkgs> { } }:
pkgs.dockerTools.buildLayeredImage {
  name = "nginx";
  tag = "v${pkgs.nginx.version}";
  config = {
    Entrypoint = [ "${pkgs.nginx}/bin/nginx" ];
    Cmd = [ "-g" "daemon off;" ];
  };
}
```

Revisiting Reusability of Shared Dependencies (2/2)

```
$ podman save \
  --format=oci-dir \
  --output=. \
  nginx:v1.28.0
├─ …
└─ blobs
   └─ sha256
      ├─ …
      ├─ 2a9d…5ed0 …/bin/nginx
      ├─ 674a…2802 …/lib/ld-musl-x86_64.so.1
      └─ …
```

```
$ podman save \
  --format=oci-dir \
  --output=. \
  weather:v0.0.1
├─ …
└─ blobs
   └─ sha256
      ├─ …
      ├─ f796…2330 …/bin/weather
      ├─ 674a…2802 …/lib/ld-musl-x86_64.so.1
      └─ …
```

```nix
{ pkgs ? import <nixpkgs> { } }:
pkgs.dockerTools.buildLayeredImage {
  name = "nginx";
  tag = "v${pkgs.nginx.version}";
  config = {
    Entrypoint = [ "${pkgs.nginx}/bin/nginx" ];
    Cmd = [ "-g" "daemon off;" ];
  };
}
```

- each dependency translates to an OCI layer
- dependencies can be shared across OCI images
- dependency management by self-contained Nixpkgs

Software Supply Chain: SBOM and SLSA (1/2)

```
{ pkgs ? import <nixpkgs> { } }:
let
  name = "weather";
  version = "0.0.1";
  bin = pkgs.buildGoModule {
    inherit version;
    pname = name;
    src = ./.;
    vendorHash = "sha256-Swi5…Lg0g=";
  };
in
pkgs.dockerTools.buildLayeredImage {
  inherit name;
  tag = "v${version}";
  contents = with pkgs; [
    cacert
  ];
  config.Entrypoint = [
    "${bin}/bin/weather"
  ];
}
```

- Software Bill of Materials (SBOMs)[4]
  - declarative approach streamlines SBOM generation
  - no need to scan the output
  - reducing
    - false positives (“SBOM claims it is included, but it is not”)
    - false negatives (“SBOM claims it is not included, but it is”)
- Supply-chain Levels for Software Artifacts (SLSA)[5]
  - explicit provenance inputs
  - hermetic and isolated builds
  - reproducibility
  - precise dependency graph

[4] https://spdx.dev

[5] https://slsa.dev

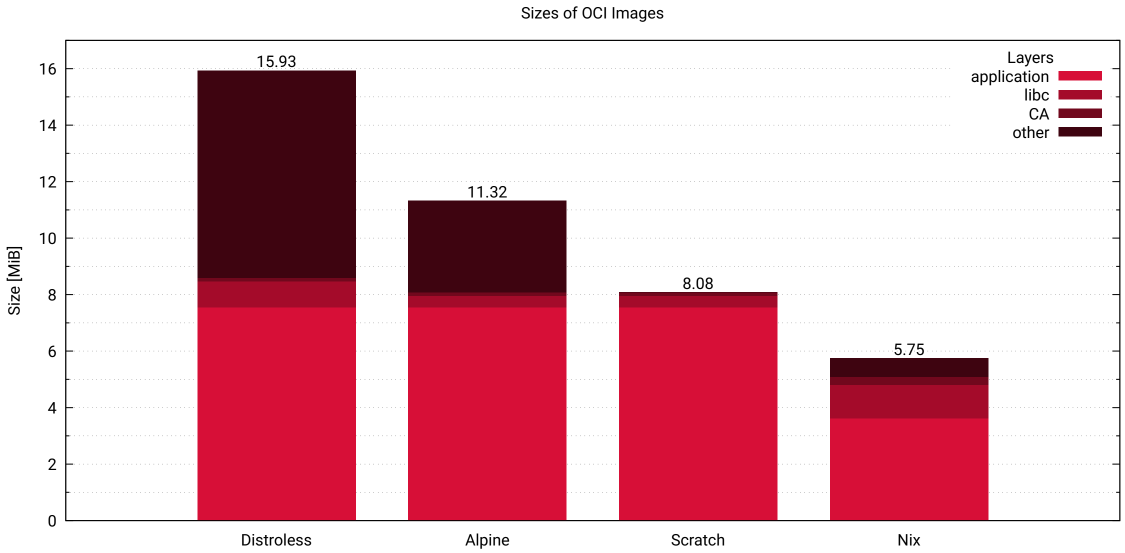
Software Supply Chain: SBOM and SLSA (2/2)

```
$ nix profile install github:tiiuae/sbomnix
```

```
$ sbomnix --depth=2 ./result
spdxVersion: SPDX-2.3
dataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
name: SPDXRef-nix-store-q33b…q3r6-weather-0.0.1.tar.gz.drv
documentNamespace: sbomnix://…
packages:
  - name: musl
    SPDXID: SPDXRef-nix-store-s37i…8m8x-musl-1.2.5.drv
    versionInfo: 1.2.5
    externalRefs:
      - referenceCategory: SECURITY
        referenceType: cpe23Type
        referenceLocator: cpe:2.3:a:musl:musl:1.2.5:…
      - referenceCategory: PACKAGE-MANAGER
        referenceType: purl
        referenceLocator: pkg:nix/musl@1.2.5
  - name: nss-cacert
    SPDXID: SPDXRef-nix-store-qr3v…f7yv-nss-cacert-3.117.drv
    versionInfo: '3.117'
    externalRefs:
      - referenceCategory: SECURITY
        referenceType: cpe23Type
  # …
```

```
$ provenance --recursive ./result
_type: https://in-toto.io/Statement/v1
subject:
  - name: out
    uri: /nix/store/y688…yysp-weather-0.0.1.tar.gz
    digest:
      sha256: 0csh…ny7l
predicateType: https://slsa.dev/provenance/v1
predicate:
  buildDefinition:
    resolvedDependencies:
      - name: musl-1.2.5
        uri: /nix/store/f597…7fgw-musl-1.2.5
        digest:
          sha256: 1pry…qdl7
      - name: nss-cacert-3.117
        uri: /nix/store/4css…w2sr-nss-cacert-3.117.drv
        digest:
          sha256: 0ga0…3ipm
      - name: gcc-14.3.0
        uri: /nix/store/38az…y6fh-gcc-14.3.0.drv
        digest:
          sha256: 1fq6…j9dw
      # …
```

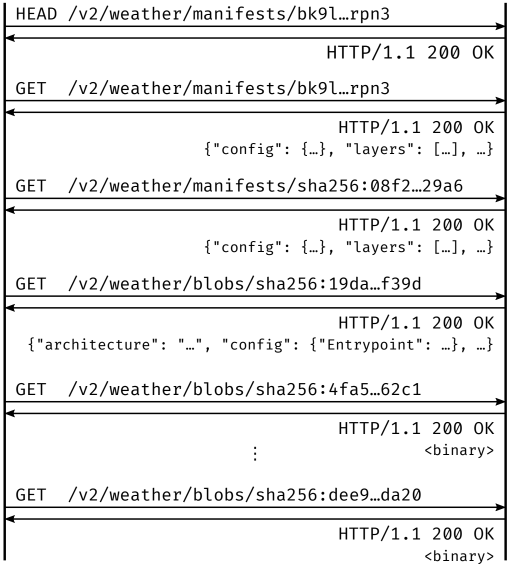
Comparing OCI Image Sizes



Advanced Use Cases: Nix OCI

gitlab.com/arik.grahl/nix-oci

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: weather
spec:
  containers:
  - name: weather
    image: example.com/
     weather:bbk9l…rpn3
```



```
HEAD /v2/weather/manifests/bk9l…rpn3
                                                  HTTP/1.1 200 OK
GET  /v2/weather/manifests/bk9l…rpn3
                                                  HTTP/1.1 200 OK
                              {"config": {…}, "layers": […], …}
GET  /v2/weather/manifests/sha256:08f2…29a6
                                                  HTTP/1.1 200 OK
                              {"config": {…}, "layers": […], …}
GET  /v2/weather/blobs/sha256:19da…f39d
                                                  HTTP/1.1 200 OK
 {"architecture": "…", "config": {"Entrypoint": …}, …}
GET  /v2/weather/blobs/sha256:4fa5…62c1
                                                  HTTP/1.1 200 OK
                          ⋮                              <binary>
GET  /v2/weather/blobs/sha256:dee9…da20
                                                  HTTP/1.1 200 OK
                                                         <binary>
```

```
/nix/store/bk9l…rpn3-weather.tar.gz
├─ oci-layout
├─ index.json
└─ blobs
   └─ sha256
      ├─ 08f2…29a6
      ├─ 19da…f39d
      ├─ 4fa5…62c1
      ├─ 5253…fa1c
      ├─ 60de…a117
      ├─ 9759…9caa
      ├─ cde9…f780
      ├─ cfad…89f5
      └─ dee9…da20
```

```nix
pkgs.dockerTools.buildLayeredImage {
  inherit name;
  tag = "v${version}";
  contents = with pkgs; [
    cacert
  ];
  config.Entrypoint = [
    "${bin}/bin/weather"
  ];
}
```

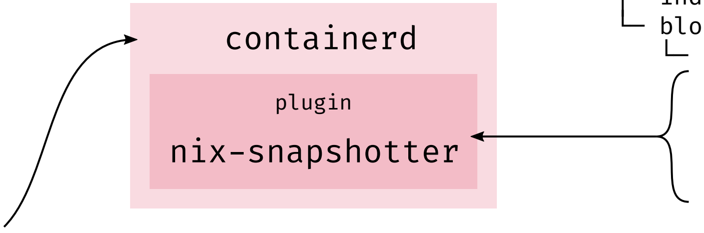
Advanced Use Cases: Nix-Snapshotter

github.com/pdtpartners/nix-snapshotter

```nix
pkgs.nix-snapshotter.buildImage {
  inherit name;
  resolvedByNix = true;
  copyToRoot = with pkgs; [ cacert ];
  config.entrypoint = [ "${bin}/bin/weather" ];
}
```

```
/nix/store/dzkl…fasj-nix-image-weather.tar
├─ oci-layout
├─ index.json
└─ blobs
   └─ sha256
      ├─ 9158…ad18
      ├─ ab57…cc8e
      ├─ 4523…06dc
      ├─ f796…2330
      └─ 674a…2802
```

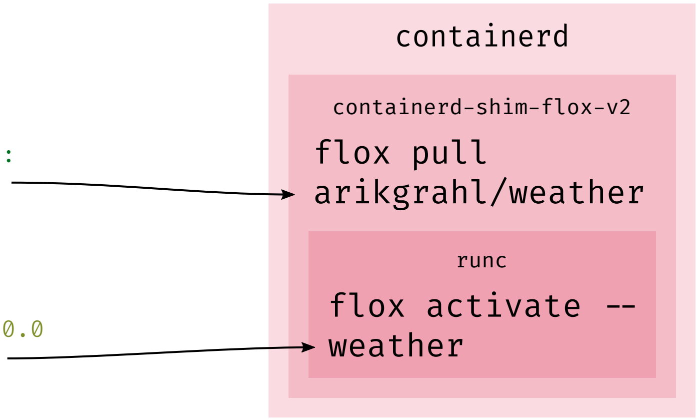
```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: weather
spec:
  containers:
  - name: weather
    image: nix:0/nix/store/
     dzkl…fasj-nix-image-weather.tar
```



Advanced Use Cases: Flox Imageless Kubernetes

flox.dev/kubernetes

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: weather
  annotations:
    flox.dev/environment:
     "arikgrahl/weather"
spec:
  runtimeClassName: flox
  containers:
  - name: weather
    image: flox/empty:1.0.0
    command: ["weather"]
```



upcoming talk at SCALE by Morgan Helton: Deterministically Built Containers

Conclusion (1/3)

- Docker
  - imperative configuration (`Dockerfile`)
  - impure and potentially non-reproducible builds
  - no abstraction for shared dependencies

Conclusion (2/3)

- Nix
  - declarative and reproducible per design
  - Nixpkgs
    - self-contained dependency management
    - large and attractive ecosystem
  - `dockerTools` translates Nix’s benefits to the OCI ecosystem
    - declarative and reproducible OCI containers
    - share dependencies accross OCI images
    - strong software supply chain security (SSCS)
    - potentially small OCI images

Conclusion (3/3)

- tooling
  - drop-in replacement, e.g. `docker load < ./result`
  - network-centric approach, e.g. Nix OCI[6]
  - container runtime-centric approach, e.g. Nix-Snapshotter[7]
  - environment-centric approach, e.g. Flox Imageless Kubernetes[8]

[6] gitlab.com/arik.grahl/nix-oci

[7] github.com/pdtpartners/nix-snapshotter

[8] flox.dev/kubernetes

Questions & Answers

arik-grahl.de/talks/cds-london-2026

arik-grahl.de/tags/nix

sys11.it/mka