

Nix geht über Docker

Johannes Maier

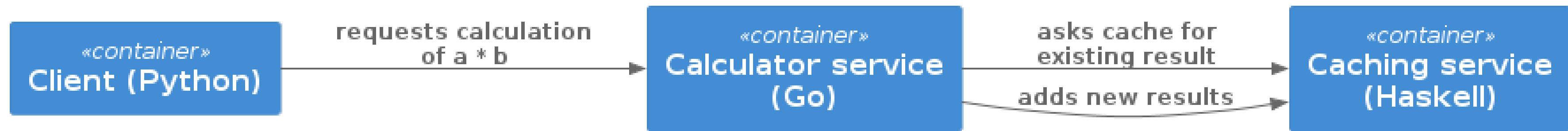
Created: 2023-02-07 Tue 16:28

Ziel

- Teaser für Nix

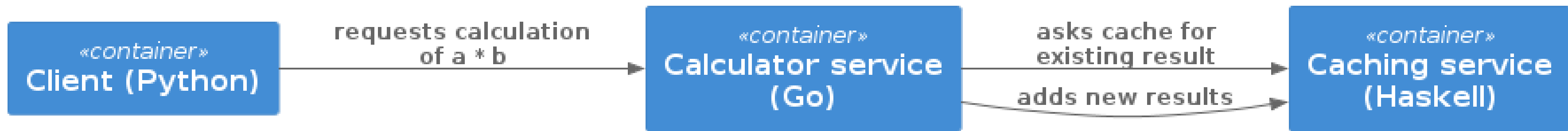
Ausgangssituation

Unsere Anwendung / Services



Ausgangssituation

Unsere Anwendung / Services



→ Annahme: Soll als Docker-Komposition aus drei Images ausgeliefert werden

Anforderungen an Auslieferung

1. Minimalität bzgl.
 - Größe
 - installierter Software
2. Reproduzierbarkeit
3. Effizientes Bauen (Caching)
4. Evtl. festes Basisimage ("nur Debian erlaubt")

Dockerfile des Caching-Service

```
FROM ubuntu
```

```
COPY . .
```

```
RUN apt-get update
```

```
RUN apt-get install -y cabal-install zlib1g-dev
```

```
RUN cabal update
```

```
RUN cabal build
```

```
CMD cabal run
```

Probleme?

Dockerfile des Caching-Service

```
# Basisimage nicht gepinnt -> Reproduzierbarkeit
FROM ubuntu

# Zu früh -> zerstört Caching
COPY . .

# - Wird einzeln gecacht, daher bei Änderungen danach nicht neu ausgeführt
# - Reproduzierbarkeit unmöglich
RUN apt-get update
RUN apt-get install -y cabal-install zlib1g-dev

# S.o.
RUN cabal update
RUN cabal build
# Hier sollte das gebaute Binary laufen und cabal unnötig sein
CMD cabal run
```

- Keine Minimalität wegen unnötig enthaltener Software

Zweiter Versuch

FROM haskell@sha256:f99b7e5417f75089b53e1077a68c6333c48b82aff478a8af292a7b7f8e541832

WORKDIR /build

COPY haskell-backend.cabal ./

COPY src-exe src-exe

RUN cabal update && cabal install

Multi-stage build

FROM ubuntu@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d

RUN apt-get update && apt-get install -y zlib1g-dev

Benutzer und Gruppe anlegen

RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1001 prod

USER prod

COPY --from=0 /root/.cabal/bin/haskell-backend ./

CMD [**"/haskell-backend"**]

Zweiter Versuch: Analyse

FROM haskell@sha256:f99b7e5417f75089b53e1077a68c6333c48b82aff478a8a:
...

- Feste Version der benötigten Tools → besser

Zweiter Versuch: Analyse

```
...  
COPY haskell-backend.cabal ./  
COPY src-exe src-exe  
...
```

- Nur das Nötigste (vermutlich!) → besser

Zweiter Versuch: Analyse

```
...  
RUN cabal update && cabal install  
...
```

- Verlassen uns auf Abhängigkeitsmanagement bzw. Pinning durch cabal

Zweiter Versuch: Analyse

...

FROM

ubuntu@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13

...

- Feste Basis → besser
- Minimalität?

Zweiter Versuch: Analyse

```
...  
RUN apt-get update && apt-get install -y zlib1g-dev  
...
```

Benötigte Laufzeitabhängigkeiten 🙄

Zweiter Versuch: Analyse

```
...  
RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1001  
USER prod  
...
```

- Service läuft unter dediziertem User, nicht root → besser

Zweiter Versuch: Analyse

```
...  
COPY --from=0 /root/.cabal/bin/haskell-backend ./  
CMD [ "./haskell-backend" ]  
...
```

- Lediglich fertiges Binary enthalten aus vorigem Build

Zweiter Versuch: Fazit

- Probleme beim Erstellen des Dockerfile:
Laufzeitabhängigkeiten fehlen oder unpassend!
- alpine nicht einfach möglich (musl vs. libc, andere Bibliotheken/Versionen)
- Anzahl Pinning-Mechanismen = Anzahl Tools, Sprachen etc.

Zweiter Versuch: Fazit

- Probleme beim Erstellen des Dockerfile:
Laufzeitabhängigkeiten fehlen oder unpassend!
(Reproduzierbarkeit)
- alpine nicht einfach möglich (musl vs. libc, andere Bibliotheken/Versionen) **(Minimalität)**
- Anzahl Pinning-Mechanismen = Anzahl Tools, Sprachen etc.
(Reproduzierbarkeit)

Zweiter Versuch: Fazit

- Gute Dockerfiles möglich, aber sehr schwer zu schreiben
- Je "besser" das Dockerfile, desto höher der Wartungsaufwand
- Weicht i.d.R. von Entwicklungsumgebung ab (lokales Debugging?)

Abhängigkeitsmanagement

Wunsch:

- Abhängigkeitsgraph(en) unserer Programme bekannt und fixiert

Nix!

<https://nixos.org>

- Package-Manager für "Bauanleitungen" (sprachagnostisch)
- Monorepo `nixos/nixpkgs`
- Programmiersprache
- (Betriebssystem: NixOS)

Nix-Store

- `/nix/store`: read-only-FS
- Neue Version, neuer Hash, neuer Pfad
- Store ist Cache

Bauanleitungsbeispiel

```
{ pkgs }:  
  
pkgs.stdenv.mkDerivation {  
  pname = "my-example";  
  version = "0.1.0";  
  src = ./.;  
  buildInputs = [ pkgs.gnumake pkgs.gcc ];  
  buildPhase = "make";  
  installPhase = ''  
    make install  
    cp -r myResult $out  
  '';  
}
```

In der Praxis

- `mkDerivation` low-level
- DSLs für viele Programmiersprachen, Shell-Skripte etc.

Nixify it: Haskell-Service

```
{ pkgs ? import <nixpkgs> { } }:
```

```
pkgs.haskellPackages.callCabal2nix "haskell-backend" ./ . { }
```


Nixify it: Haskell-Service

Laufzeitabhängigkeiten revisited:

```
store_path=$(nix-build haskell_backend | tail -n 1)
echo $store_path

nix-store -qR $store_path

nix-store --export $(nix-store -qR $store_path) > closure
```

closure kann auf dem Zielsystem importiert werden.

Laufzeitabhängigkeiten des Caching-Service

```
$ docker run --rm -it haskell-backend-1 ldd haskell-backend
linux-vdso.so.1 (0x00007ffc46f88000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fee89f30000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fee89f14000)
librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007fee89f0f000)
libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007fee89f0a000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fee89f05000)
libgmp.so.10 => /lib/x86_64-linux-gnu/libgmp.so.10 (0x00007fee89e81000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fee89c59000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fee89b72000)
/lib64/ld-linux-x86-64.so.2 (0x00007fee89f39000)
```

Laufzeitabhängigkeiten des Caching-Service

```
$ ldd ~/path/to/haskell-backend
linux-vdso.so.1 (0x00007ffebdb24000)
libpthread.so.0 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/libpthread.so.0 (0x00007fb3b72c0000)
libz.so.1 => /nix/store/b36ilvc5hhfpcp7kv1kvrkgcxxpmxfsd-zlib-1.2.12/lib/libz.so.1 (0x00007fb3b72bd000)
libgmp.so.10 => /nix/store/qxrvrhlfaishinykki6qy6nqd4wv8mdp-gmp-with-cxx-6.2.1/lib/libgmp.so.10 (0x00007fb3b721c000)
libc.so.6 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/libc.so.6 (0x00007fb3b701d000)
libm.so.6 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/libm.so.6 (0x00007fb3b6f42000)
librt.so.1 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/librt.so.1 (0x00007fb3b6f3d000)
libdl.so.2 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/libdl.so.2 (0x00007fb3b6f38000)
libffi.so.8 => /nix/store/gm6q7jmajjmnwd29wgbq2jm3x37vsw3h-libffi-3.4.2/lib/libffi.so.8 (0x00007fb3b6f2b000)
/nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/ld-linux-x86-64.so.2 => /nix/store/ayrsyv7npr0lcbann4k9lxr19x813f0z-glibc-2.34-115/lib/ld-linux-x86-64.so.2
```

Docker-Images mit Nix?

- Docker-Layers bestehen aus File-System-Diffs (OCI Image Format)
- Kennen Abschluss nixifizierter Anwendungen

→ Kennen auch das Diff

→ Nix kann Docker-Images erzeugen

Dockerize it: Haskell-Service

```
{ pkgs ? import <nixpkgs> { } }:  
  
let haskellBackend = import ../haskell_backend/default.nix { inherit  
in pkgs.dockerTools.buildImage {  
    name = "haskell-backend";  
    tag = "latest";  
    # contents = [ pkgs.bash pkgs.coreutils ];  
    config = {  
        # Start the Haskell service as the CMD of the image  
        Cmd = "${haskellBackend}/bin/haskell-backend";  
    };  
}
```

Benutzung

```
nix-build nix/haskell-docker-image.nix  
docker load < result
```

Nixify it: Go-Service und Python-Client

```
{ pkgs ? import <nixpkgs> { } }:
```

```
pkgs.buildGoModule {
```

```
  src = ./.;
```

```
  pname = "go_backend";
```

```
  version = "0.1.0";
```

```
  vendorSha256 = "sha256-pQpattmS9Vm03ZIQUFn66az8GSmB4IvYhTTCFn6SUmo
```

```
}
```

```
{ pkgs ? import <nixpkgs> { } }:
```

```
let myPython = pkgs.python310.withPackages (p: [ p.loguru p.requests
```

```
in pkgs.writeShellScript "demo" ''
```

```
  ${myPython}/bin/python ${./python_client}/__init__.py
```

```
''
```

Dockerize it: Go und Python



Neue "Anforderung"

Beide Services in einem Image!

Dockerfile?

```
FROM haskell@sha256:f99b7e5417f75089b53e1077a68c6333c48b82aff478a8a1
```

```
WORKDIR /build
```

```
COPY haskell-backend.cabal ./
```

```
COPY src-exe src-exe
```

```
RUN cabal update && cabal install
```

```
# Multi-stage build
```

```
FROM ubuntu@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13
```

```
RUN apt-get update && apt-get install -y zlib1g-dev
```

```
# Benutzer und Gruppe anlegen
```

```
RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1001
```

```
USER prod
```

```
COPY --from=0 /root/.cabal/bin/haskell-backend ./
```

```
CMD [ "./haskell-backend" ]
```

Nix?

```
let
  haskellBackend = import ../haskell_backend { inherit pkgs; };
  goBackend = import ../go_backend { inherit pkgs; };
  runScript = pkgs.writeShellScript "run" ''
    ${haskellBackend}/bin/haskell-backend &
    ${goBackend}/bin/server
  '';
in pkgs.dockerTools.buildImage {
  name = "both-backends";
  tag = "latest";
  config.Cmd = runScript;
}
```

War's das?

War's das?

- Minimal

War's das?

- Minimal ✓

War's das?

- Minimal ✓
- Reproduzierbar

War's das?

- Minimal ✓
- ~~Reproduzierbar~~ Pinning fehlt!

Nixpkgs pinnen

Naiv:

```
let
  pkgs = import (builtins.fetchTarball {
    name = "my-nixpkgs-pin";
    url =
      "https://github.com/nixos/nixpkgs/archive/87d34a6b8982e901b8e5
    sha256 = "sha256:0dqjw05vbdf6ahy71zag8gsbfcgrf7fxz3xkwqqwap10qk9
  }) { };
in ...
```

Nixpkgs pinnen

- <https://github.com/nmattia/niv>
- <https://nixos.wiki/wiki/Flakes>

Vorteile

- Reproduzierbar und programmierbar!
- Eine Sprache für alles
- Reproduzierbare (Entwicklungs-)Umgebungen (`nix-shell`) analog
- Riesige und hilfsbereite Community

Nix-Hürden

- Dokumentation
- Obskure Sprache (aber nicht schwer!)
- Lernkurve
- Ökosystem bewegt sich sehr schnell
- Große Unterschiede bei Sprachunterstützung

Beispielrepo + Folien

- <https://github.com/kenranunderscore/docker-mit-nix-talk>