029-Bau-Pipeline-Spiegel

DevOps- und Administrationsaufgabe
Anforderungsniveau: Mittel

1 Ziel der Aufgabe

Ein Bau-Server steht in einem abgetrennten Netz und erreicht genau einen Rechner: den internen Spiegel. Dort sollen zwei Container-Abbilder entstehen – eines für einen Node-Dienst, eines für einen Rust-Dienst. Alles, was nicht aus dem Projekt selbst stammt, kommt über den Spiegel und wird beim Bau auf Integrität geprüft. Geprüft wird nicht, ob Sie ein Dockerfile schreiben können, sondern ob Sie vollständig erfassen, woher ein Bau seine Bestandteile zieht, und ob er auch dann noch durchläuft, wenn von außen nichts mehr nachkommt.

Veranschlagte Bearbeitungszeit: 6–8 Stunden

Verwendete Techniken: Docker, BuildKit, mehrstufige Dockerfiles, `docker compose` (Compose V2, ohne `version:`-Schlüssel), Container-Registry im Netz (`registry:2`) mit Verweisen über Digest, Verdaccio als npm-Spiegel, `cargo vendor`, Binärartefakte mit `SHA256SUMS`, ein abgeschotteter `buildx`-Erbauer im Docker-Netz ohne Route nach außen, reproduzierbarer Abnahmelauf mit leeren Zwischenspeichern.

2 Aufgabenbeschreibung

1. **Ausgangslage anlegen**

   Es wird Ihnen kein Archiv mitgeliefert. Sie legen die beiden Beispieldienste selbst an, bewusst winzig:

   - `portal/` – ein Node-Dienst in TypeScript mit genau einem Endpunkt `/gesund`, der eine JSON-Antwort liefert. Er braucht mindestens eine echte Abhängigkeit aus npm, damit in der `package-lock.json` etwas steht.
   - `sla-dienst/` – ein Rust-HTTP-Dienst mit genau einem Endpunkt `/gesund`, ebenfalls JSON, mit `Cargo.lock`.

   Diese beiden Dienste sind ausdrücklich *nicht* der Prüfgegenstand. Halten Sie sie so klein wie möglich; niemand bewertet ihre Fachlichkeit. Sie sind nur der Anlass für einen Bau, der etwas von außen braucht.

   Die Abschottung bilden Sie lokal mit Docker-Netzen nach: ein Netz `bau-netz`, angelegt mit `--internal`, an dem Spiegel und Erbauer hängen, und ein zweites Netz `spiegel-uplink`, an dem der Spiegel nur hängt, solange er befüllt wird.

2. **Bestandsaufnahme**

   Halten Sie vor dem Umbau in einer `komponentenliste.md` fest, was die beiden Bauvorgänge aus dem Netz holen: je Zeile Bestandteil, Bezugsquelle, Version beziehungsweise Digest und der Weg in den Spiegel. Ein Bau zieht mehr als die Zeilen, die im Dockerfile stehen: Basis-Abbilder samt Unterbau, Pakete aus `apt` oder `apk`, npm-Pakete mit ihren Abhängigkeiten, Kisten von crates.io, die Rust-Werkzeugkette und Binärwerkzeuge von den Seiten der Herausgeber. Die Lücken Ihrer Liste finden Sie am schnellsten, indem Sie den Bau ohne Weg nach außen laufen lassen und zusehen, woran er scheitert.

3. **Spiegel bereitstellen**

   Legen Sie unter `spiegel/` einen Compose-Stapel mit drei Diensten an, die jeweils *eigene* Namen tragen – der Dienstname ist der Name im Netz:

   - `registry` – eine Container-Registry (`registry:2`), im Netz erreichbar als `registry:5000`
   - `npm-spiegel` – Verdaccio, im Netz erreichbar als `npm-spiegel:4873`
   - `dateien` – ein statischer Webserver für Binärartefakte und das Rust-Verzeichnis, im Netz erreichbar als `dateien:80`

   Veröffentlichen Sie die Anschlüsse zusätzlich auf `127.0.0.1`, damit Sie den Spiegel vom Entwicklungsrechner aus befüllen können. Dazu gehören `compose.yaml`, ein Skript `befuellen.sh`, das die Bestandteile einmalig mit Netz holt, `artefakte/` mit `SHA256SUMS` und die Ablage der Dienste unter `daten/`, die nicht ins Repository gehört.

   Basis-Abbilder holen Sie nicht zur Bauzeit durch den Spiegel hindurch, sondern legen sie hinein. Binärartefakte, die nicht aus einem Paketverwalter stammen, landen in `artefakte/`, die Prüfsumme daneben – gebildet auf dem Rechner, der die Datei geholt hat, und verglichen mit der Angabe des Herausgebers, soweit es eine gibt. Die Rust-Abhängigkeiten legen Sie mit `cargo vendor` ab und stellen sie als Archiv über `dateien` bereit.

```
# Die Registry im Spiegel spricht HTTP, nicht HTTPS -- daher die TLS-Schalter.
skopeo copy --all --dest-tls-verify=false \
  docker://docker.io/library/node:22-bookworm-slim \
  docker://127.0.0.1:5000/basis/node:22-bookworm-slim

cd spiegel/artefakte
curl -fsSL -O https://github.com/krallin/tini/releases/download/v0.19.0/tini-static-
    amd64
sha256sum tini-static-amd64 >> SHA256SUMS

cd ../../sla-dienst
mkdir -p .cargo
cargo vendor vendor/ > .cargo/config.toml    # Quellersetzung erzeugen
tar czf ../spiegel/artefakte/sla-dienst-vendor-0.1.0.tar.gz vendor/

# Prüfsummen ohne Pfadanteil aufnehmen, sonst findet die Prüfung
# im Bau die Datei nicht wieder.
cd ../spiegel/artefakte
sha256sum sla-dienst-vendor-0.1.0.tar.gz >> SHA256SUMS
```

4. **Bau abschotten**

   `docker build` kennt für `--network` nur `default`, `none` und `host` und lehnt einen eigenen Netznamen ab. Die Abschottung entsteht deshalb über einen eigenen Erbauer mit dem Treiber `docker-container`, der selbst in `bau-netz` steht und von dort nirgendwo anders hinkommt. Weil die Registry im Spiegel HTTP spricht, braucht der Erbauer eine Konfiguration:

```
# spiegel/buildkitd.toml
# [registry."registry:5000"]
#   http = true

docker network create --internal bau-netz
docker buildx create --name abgeschottet --driver docker-container \
  --driver-opt network=bau-netz --config spiegel/buildkitd.toml --bootstrap
docker buildx --builder abgeschottet build --progress=plain --load \
  -t portal:0.1.0 portal/
```

Das Abbild des Erbauers zieht der Docker-Daemon einmalig beim Anlegen – das gehört in die Einrichtung, nicht in den Abnahmelauf. Beide Dockerfiles sprechen danach ausschließlich den Spiegel an. Verbindlich ist:

- Jedes `FROM` verweist auf `registry:5000` und ist über den Digest festgelegt, nicht über ein Etikett.
- Der npm-Lauf nutzt `npm ci` gegen den Spiegel: eine `.npmrc` im Projekt setzt die Registry auf `http://npm-spiegel:4873/`, es gibt keine Auflösung gegen `registry.npmjs.org` und kein `npx` mit Nachladen zur Bauzeit. Achten Sie auf die `resolved`-Einträge in `package-lock.json`. Der Rust-Lauf übersetzt mit `--locked --offline` gegen `vendor/`.
- Distributionspakete kommen aus dem Spiegel oder entfallen durch ein passend gewähltes Basis-Abbild. Der Laufzeitteil enthält keine Werkzeugkette, läuft nicht als `root` und trägt `HEALTHCHECK` sowie die Kennzeichnung nach OCI (`org.opencontainers.image.*`).

```
# portal/Dockerfile (Ausschnitt)
# syntax=docker/dockerfile:1
FROM registry:5000/basis/node:22-bookworm-slim@sha256:0000... AS bau
WORKDIR /bau
COPY .npmrc package.json package-lock.json ./
RUN npm ci --no-audit --no-fund
COPY . .
RUN npm run build
# Laufzeitstufe: eigenes Basis-Abbild, kein npm, kein root
```

5. **Integrität im Bau prüfen**

   Jeder Bestandteil, der als Datei über `dateien` kommt, wird im Bau geprüft. Schlägt die Prüfung fehl, bricht der Bau ab – nicht erst der Start des Containers. Für einzelne Dateien genügt `ADD --checksum`, für Archive und Sammlungen prüfen Sie gegen `SHA256SUMS`:

```
# syntax=docker/dockerfile:1
ADD --checksum=sha256:0000... http://dateien/tini-static-amd64 /usr/local/bin/tini

ADD http://dateien/SHA256SUMS /tmp/SHA256SUMS
ADD http://dateien/sla-dienst-vendor-0.1.0.tar.gz /tmp/
RUN cd /tmp && sha256sum --ignore-missing -c SHA256SUMS \
 && tar xzf sla-dienst-vendor-0.1.0.tar.gz -C /bau
```

   Zeigen Sie in `protokolle/`, dass die Prüfung greift: verfälschte Kopie eines Artefakts in den Spiegel legen, Bau starten, Fehlermeldung und Exit-Code festhalten. Ein Abbruch, den niemand gesehen hat, ist kein Nachweis. Bei den Paketverwaltern übernimmt das deren eigener Weg (`integrity` aus `package-lock.json`, Prüfsummen aus `Cargo.lock`); halten Sie fest, welcher Bestandteil über welchen Mechanismus abgesichert ist.

6. **Abnahmelauf**

   `abnahme.sh` führt den Bau unter den Bedingungen des Bau-Servers durch und endet mit sprechendem Exit-Code. Es muss mindestens:

   a) den Spiegel vom äußeren Netz trennen (`docker network disconnect spiegel-uplink ...` für jeden Spiegeldienst),
   b) die Zwischenspeicher leeren (`docker buildx prune -af --builder abgeschottet`, beide Abbilder entfernen),
   c) beide Abbilder mit dem Erbauer `abgeschottet` bauen,
   d) sie starten und je einen Aufruf gegen `/gesund` absetzen,
   e) die Digests der Basis- und der erzeugten Abbilder protokollieren.

Der Lauf muss zweimal hintereinander durchgehen und beim zweiten Mal dieselben Eingangs-Digests verwenden; beide Protokolle liegen bei.

7. **Betrieb dokumentieren**

   `BETRIEB.md` beschreibt die Pflege des Spiegels im Alltag, konkret bezogen auf die Zeilen Ihrer `komponentenliste.md`:

   - Wie kommt eine neue Version in den Spiegel, und wer prüft dabei was?
   - Liegt jeder Bestandteil als eigene Kopie im Spiegel, oder wird er bei Bedarf von außen nachgereicht? Gehen Sie die Liste Zeile für Zeile durch und kennzeichnen Sie beides.
   - Ein Bestandteil wird an seiner Ursprungsquelle nicht mehr angeboten – Etikett verschwunden, Projekt archiviert, Seite abgeschaltet. Was passiert beim nächsten Bau, und wie stellen Sie ihn wieder her?

3 Anforderungen

- Beide Abbilder entstehen mit dem Erbauer im internen Netz: kein Bauschritt greift nach außen. Alle `FROM`-Zeilen verweisen auf `registry:5000` und sind über `@sha256:` festgelegt.
- Jedes über HTTP geholte Artefakt wird im Bau gegen eine Prüfsumme gehalten; ein Fehlschlag bricht den Bau ab, der Nachweis liegt bei.
- `komponentenliste.md` deckt sich mit dem, was der Abnahmelauf anfasst; `abnahme.sh` läuft nach dem Entfernen von Abbildern und Zwischenspeichern ohne Eingriff durch.
- Der Laufzeitteil enthält keine Werkzeugkette, keinen Paketverwalter-Zwischenspeicher, keine Zugangsdaten, kein `root`.
- Keine echten Zugangsdaten im Repository: erfundene Werte (`example.invalid`, `hunter2`, `AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE`), und vor der Abgabe die Historie prüfen, nicht nur den Arbeitsbaum.
- **Kein Formatierer ohne Projektkonfiguration:** Prettier, `rustfmt`, `shfmt` nur mit Konfiguration im Repository, nur über ohnehin geänderte Dateien.

4 Abgabe

- Ein Git-Repository mit nachvollziehbarer Commit-Historie: `portal/`, `sla-dienst/`, `spiegel/`, beide Dockerfiles, `abnahme.sh`, `komponentenliste.md`, `BETRIEB.md`.
- `protokolle/` mit beiden Abnahmeläufen, dem Fehlschlag der Prüfsumme und den verwendeten Digests. Große Binärartefakte gehören nicht ins Repository: über `befuellen.sh` anlegen, nur `SHA256SUMS` versionieren.
- Eine README: Spiegel und Erbauer auf einem leeren Rechner einrichten, Abnahmelauf starten, benötigte Zeit je Abschnitt, eine Notiz, was mit mehr Zeit dazugekommen wäre.

5 Bewertungskriterien

- **Bestandsaufnahme (30 %):** Sind alle externen Bestandteile erfasst, auch die nicht im Dockerfile genannten?
- **Abschottung (25 %):** Läuft der Bau nachweislich ohne Weg nach außen, oder stützt er sich still auf einen Zwischenspeicher? Positiv fällt auf, wer eigene Kopien im Spiegel von bloß weitergereichten Bestandteilen unterscheidet.
- **Integritätsprüfung (20 %):** Ist jeder Bestandteil abgesichert, bricht der Bau im Fehlerfall ab, ist das belegt?
- **Betriebliches Denken (15 %):** Taugt `BETRIEB.md` für den Alltag?

- **Handwerk (10 %):** Dockerfiles, Laufzeitabbilder, Lesbarkeit der Skripte, Dokumentation. Die beiden Beispieldienste zählen hier nicht mit, solange sie starten und antworten.