028-Caddy-Reverse-Proxy

DevOps- und Administrationsaufgabe
Anforderungsniveau: Mittel

1 Ziel der Aufgabe

Sie betreiben einen kleinen Stack aus zwei Diensten hinter einem Caddy-Reverse-Proxy, gestartet über `docker compose`. Die Proxy-Konfiguration liegt als einzelne Datei auf dem Host und ist in den Container eingehängt. Im laufenden Betrieb – ohne Neustart – ergänzen Sie eine weitere Route und weisen nach, dass die Änderung im Container angekommen ist. Geprüft wird nicht, ob Sie eine Konfigurationsdatei schreiben können, sondern ob Sie den Unterschied zwischen *geändert* und *wirksam* belegen statt ihn anzunehmen.

Veranschlagte Bearbeitungszeit: 4–6 Stunden

Verwendete Techniken:

- Docker und `docker compose` (Compose V2, ohne `version`:-Schlüssel), Bind-Mounts, Netzwerk zwischen Containern
- Caddy 2 als Reverse-Proxy, Konfiguration über ein Caddyfile
- Neuladen im laufenden Container (`caddy reload`) und die Admin-Schnittstelle auf Port 2019
- Nachweis per `docker compose exec`, `curl` und einem Prüfskript mit sprechenden Exit-Codes

Voraussetzung: Die Aufgabe setzt *Docker Engine unter Linux* voraus – eine virtuelle Maschine, WSL2 oder ein Server. Unter Docker Desktop für macOS oder Windows liegt eine zusätzliche Dateisystem-Schicht zwischen Host und Container, und das Einhängen einzelner Dateien verhält sich dort abweichend; die Aufgabe ist dann nicht sinnvoll bearbeitbar. Halten Sie in der README fest, auf welcher Umgebung Sie gearbeitet haben (Ausgabe von `docker version` genügt).

2 Aufgabenbeschreibung

1. **Stack aufsetzen**

   Repository mit `compose.yaml`, `Caddyfile`, `neu-laden.sh` und `README.md`. Die Hintergrunddienste müssen nichts können außer antworten – fertige Abbilder, nichts selbst gebaut.

```
services:
  caddy:
    image: caddy:2.8-alpine
    ports:
      - "8080:80"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
    depends_on: [dienst-a, dienst-b]

  dienst-a:
    image: traefik/whoami:v1.10.2
```

```
  dienst-b:
    image: hashicorp/http-echo:1.0.0
    command: ["-listen=:5678", "-text=antwort von dienst-b"]
```

Feste Tags statt `latest`. Nur der Proxy veröffentlicht einen Port – die beiden Dienste sind ausschließlich über das Compose-Netzwerk unter ihren Dienstnamen erreichbar.

2. **Ausgangskonfiguration**

   Das Caddyfile kennt anfangs genau eine weitergeleitete Route. Die Admin-Schnittstelle bleibt an, wird aber nicht nach außen veröffentlicht.

```
{
        admin :2019
        auto_https off
}

:80 {
        handle_path /a/* {
                reverse_proxy dienst-a:80
        }

        handle /gesundheit {
                respond "ok" 200
        }

        handle {
                respond "keine Route" 404
        }
}
```

```
docker compose up -d
curl -i http://localhost:8080/a/
curl -i http://localhost:8080/b/      # muss 404 liefern
```

3. **Route im laufenden Betrieb ergänzen**

   Der Stack läuft und soll laufen bleiben: **kein `down`, kein `up -force-recreate`, kein `restart` des Proxy-Containers.** Ergänzen Sie im Caddyfile auf dem Host eine zweite Route:

```
        handle_path /b/* {
                reverse_proxy dienst-b:5678
        }
```

   Ändern Sie die Datei so, wie Sie es im Alltag tun würden – mit Ihrem Editor oder auf der Kommandozeile, etwa per `sed -i`. Danach prüfen und neu laden:

```
docker compose exec caddy caddy validate --config /etc/caddy/Caddyfile
docker compose exec caddy caddy reload --config /etc/caddy/Caddyfile
curl -i http://localhost:8080/b/
```

4. **Wirksamkeit nachweisen**

   `caddy reload` meldet einen Erfolg, und ein `cat` auf dem Host zeigt Ihre Änderung. Beides zusammen ist noch kein Nachweis. Belegen Sie an drei Stellen, in dieser Reihenfolge:

   (a) Welchen Dateiinhalt sieht der Prozess wirklich?
   `docker compose exec caddy cat /etc/caddy/Caddyfile`

(b) Was hat der laufende Prozess übernommen?
`docker compose exec caddy wget -qO- http://localhost:2019/config/`

(c) Verhalten von außen: liefert `/b/` die Antwort von `dienst-b` und `/a/` unverändert die von `dienst-a`?

Weichen Hoststand, Containerstand und geladene Konfiguration voneinander ab, ist das keine Randnotiz, sondern das Ergebnis dieser Aufgabe. Finden Sie die Ursache, lösen Sie es ohne Neustart des Stacks und halten Sie Beobachtung und Erklärung in der README fest. Nennen Sie zwei Wege, dieselbe Lage im Voraus zu vermeiden, und sagen Sie, welchen Sie für den Produktivbetrieb wählen würden.

5. **Prüfskript schreiben**

`neu-laden.sh` nimmt dem Betrieb genau die Handgriffe ab, die Sie eben von Hand gemacht haben – und verlässt sich dabei nicht auf die Erfolgsmeldung des Neuladens.

```
#!/usr/bin/env bash
set -euo pipefail
# 1. Caddyfile pruefen  2. neu laden  3. Datei im Container gegen Host
# vergleichen  4. Route /a/ und /b/ pruefen  5. je Schritt eine Zeile ausgeben
host_summe=$(sha256sum Caddyfile | cut -d' ' -f1)
container_summe=$(docker compose exec -T caddy sha256sum /etc/caddy/Caddyfile | cut
    -d' ' -f1)

# Exit-Codes
# 0  geladen und in allen Pruefungen bestaetigt
# 1  Stack laeuft nicht oder Container nicht erreichbar
# 2  Caddyfile ist syntaktisch fehlerhaft
# 3  Host- und Containerstand weichen voneinander ab
# 4  Neuladen gemeldet, aber eine Route antwortet nicht wie erwartet
```

Schritt 4 muss auch wirklich fehlschlagen, wenn die Route fehlt. Ein Skript, das nur den Exit-Code von `curl` auswertet oder einen Status unter 500 als Erfolg wertet, hält eine 404-Antwort für grün. Prüfen Sie Statuscode **und** erwarteten Inhalt.

6. **Gegenprobe**

Nehmen Sie die Route `/b/` wieder heraus, laden Sie neu und rufen Sie `neu-laden.sh` auf. Ausgabe und Exit-Code gehören in die README. Ein Prüfskript, das nur im guten Fall gelaufen ist, hat nichts bewiesen.

Falls in Ihren Beispielen oder Ausgaben Zugangsdaten oder Hostnamen auftauchen, verwenden Sie ausschließlich erfundene Werte – etwa Benutzer `pruefer` mit Passwort `hunter2` und `example.invalid` als Hostname. Es gehören keine echten Kennungen, Tokens oder Kundendaten in dieses Repository, auch nicht in Zeilen, die Sie später wieder entfernen.

3 Anforderungen

- `docker compose up -d` startet den vollständigen Stack ohne weitere Handgriffe; danach sind `/a/` und `/gesundheit` erreichbar. `docker compose down -v` entfernt ihn rückstandsfrei.
- Die zweite Route wird **im laufenden Betrieb** ergänzt: kein `down`, kein `restart`, kein Neubau des Proxy-Containers.
- Der Nachweis erfolgt an den drei genannten Stellen. Die Erfolgsmeldung von `caddy reload` allein gilt nicht als Nachweis.

- `neu-laden.sh` beginnt mit `set -euo pipefail`, gibt je Prüfung eine Zeile aus und belegt die Exit-Codes 0 bis 4. Die Routenprüfung wertet Statuscode und Antwortinhalt aus.
- Nur der Proxy veröffentlicht einen Port; die Admin-Schnittstelle auf 2019 ist von außen nicht erreichbar – weisen Sie das nach.
- Alle Abbilder mit festem Tag, kein `latest`.
- **Kein Formatierer ohne Projektkonfiguration**: das Caddyfile wird mit dem mitgelieferten `caddy fmt` formatiert; ein Werkzeug für YAML oder Shell nur, wenn eine passende Konfigurationsdatei im Repository liegt und mitgegeben wird.
- Keine echten Zugangsdaten oder Kundennamen – weder im Arbeitsstand noch in der Historie. Prüfen Sie das mit `git log -p -all | grep -iE 'passwor|token|secret'`.

4 Abgabe

- Git-Repository mit nachvollziehbarer Historie; Ausgangszustand und ergänzte Route sind getrennte Commits, damit die Änderung im Diff sichtbar wird.
- `README.md` mit Startanleitung und den Prüfaufrufen samt erwarteter Antwort; dem Protokoll der Routenergänzung (abgesetzter Befehl, Meldung des Proxys, Ergebnis der drei Nachweise); Ihrer Erklärung samt Lösungsweg, falls Host- und Containerstand auseinandergefallen sind; zwei Wegen zur Vermeidung mit begründeter Wahl; Ausgabe und Exit-Code der Gegenprobe; der verwendeten Umgebung.
- `neu-laden.sh` ist ausführbar eingecheckt (`chmod +x`).
- `.gitignore` für lokale Umgebungsdateien und alles, was der Betrieb erzeugt – etwa ein Datenverzeichnis von Caddy, falls Sie eines einhängen.

5 Bewertungskriterien

- **Nachweisführung** – wird belegt, dass die Änderung im Prozess angekommen ist, oder wird auf die Erfolgsmeldung vertraut? Hier entscheidet sich die Aufgabe.
- **Ursachensuche** – wird eine Abweichung zwischen Host und Container bemerkt, benannt und erklärt, oder wird so lange neu gestartet, bis es zufällig passt?
- **Qualität des Prüfskripts** – schlägt es im Fehlerfall tatsächlich fehl? Sind die Exit-Codes sauber getrennt und für ein aufrufendes Skript brauchbar?
- **Aufbau des Stacks** – sinnvolle Netzführung, keine überflüssig veröffentlichten Ports, feste Tags, keine echt aussehenden Zugangsdaten oder Hostnamen im Repository.
- **Betriebliches Denken und Dokumentation** – ist der Weg für den Produktivbetrieb begründet? Lässt sich der Stack allein anhand der README starten, ändern und prüfen?