DevOps

- Rootless buildah als GitLab CI Build-Host - Debian 13 (POC)

Rootless buildah als GitLab CI Build-Host - Debian 13 (POC)

Ziel

Container-Images aus der GitLab CI/CD Pipeline bauen und in die GitLab Container Registry pushen, ohne Docker-Daemon und ohne root im Build-Pfad. Runner-Prozess und alle Builds laufen als dedizierter, unprivilegierter `gitlab-runner` User (Shell-Executor + rootless buildah).

Architektur

Shell-Executor + buildah rootless direkt auf dem Host.

- Kein Docker-Daemon, kein `docker.sock`, kein privileged Container.
- buildah läuft daemonless und rootless, per User-Namespace (subuid/subgid) gemappt.
- AppArmor bleibt aktiv.

Bewusst nicht gewählt: buildah-in-container über den Docker-Executor. Das hätte für rootless-in-container in der Praxis einen privileged Container oder aufgeweichtes seccomp/AppArmor erfordert. Preis des Shell-Executors: er isoliert Jobs nicht voneinander, das fängt eine dedizierte Build-VM plus `concurrent = 1` ab.

Voraussetzungen

- Debian 13 (Trixie) VM, dediziert als Build-Host.
- Unprivilegierte User-Namespaces sind auf Debian 13 per default offen, kein Freischalten nötig (Kontrolle in Schritt 3). Debian übernimmt bewusst nicht das restriktive AppArmor-userns-Modell von Ubuntu, der Ubuntu-Schalter `kernel.apparmor_restrict_unprivileged_userns` greift hier also nicht.

- Kernel aktuell halten: Weil userns offen ist, gehört ein gepatchter Kernel auf den Build-Host (`unattended-upgrades` mit Security-Updates aktiv lassen). Trixie ist gegen die userns-LPE CVE-2026-46331 über den Security-Channel gefixt, Debian 11/12 waren zum Berichtszeitpunkt noch verwundbar.
- GitLab-URL und ein Runner-Authentication-Token (`glrt-...`), erstellen.

Platzhalter in diesem Dokument: `https://gitlab.example.com` und `glrt-XXXX` durch die echten Werte ersetzen.

Hinweis zu den Datei-Schritten: Alle Dateien unter `/home/gitlab-runner/...` werden als der User `gitlab-runner` angelegt, damit die Rechte für rootless stimmen. Also jeweils per `sudo -u gitlab-runner vim <pfad>` editieren, nicht als root.

1. Pakete installieren

`uidmap` liefert `newuidmap`/`newgidmap` und ist auf Debian ein eigenes Paket, das nicht zuverlässig als Dependency gezogen wird. Ohne das kein rootless. `passt` liefert `pasta`, das rootless-Netzwerk-Backend, das buildah/podman unter Trixie (podman 5.x) per default nutzen; fehlt es, bricht schon der erste `RUN`-Schritt im Build mit "could not find pasta" ab. `libcap2-bin` liefert `getcap` für den Capability-Check in Schritt 3.

```
sudo apt update
sudo apt install -y buildah podman fuse-overlayfs passt uidmap libcap2-bin git

curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh"
| sudo bash
sudo apt install -y gitlab-runner
```

2. System-Unit des Runners deaktivieren

Wir fahren einen rootless User-Service, nicht den mitgelieferten System-Service. Das deb-Paket ruft im postinst `gitlab-runner install` auf und legt eine echte Unit-Datei an, daher scheitert `mask` zunächst mit "File already exists". Deshalb erst entfernen, dann maskieren:

```
sudo systemctl disable --now gitlab-runner.service
sudo rm -f /etc/systemd/system/gitlab-runner.service
sudo systemctl daemon-reload
```

```
sudo systemctl mask gitlab-runner.service
```

Kontrolle: `systemctl is-enabled gitlab-runner.service` muss `masked` sagen. Nach einem `apt upgrade gitlab-runner` kurz erneut prüfen, da das Post-Install-Script die Unit theoretisch wieder anlegen kann.

3. Rootless-Grundlagen für den Runner-User

Subuid/subgid setzen (der `gitlab-runner` wird als System-User angelegt und bekommt auf Debian keine Ranges automatisch):

```
grep gitlab-runner /etc/subuid /etc/subgid || {
  echo "gitlab-runner:100000:65536" | sudo tee -a /etc/subuid
  echo "gitlab-runner:100000:65536" | sudo tee -a /etc/subgid
}
```

Unprivilegierte User-Namespaces prüfen. `sysctl` und `getcap` liegen unter `/usr/sbin` und sind im PATH eines normalen Users nicht enthalten, daher mit `sudo` (oder vollem Pfad) aufrufen, sonst kommt "command not found":

```
sudo sysctl kernel.unprivileged_userns_clone
```

Auf Standard-Trixie sind userns per default offen, erwartet wird `1`. Meldet der Kernel stattdessen "No such file or directory", ist der Key in diesem Kernel schlicht nicht exponiert; das ist unkritisch, da userns ohnehin offen sind, dann diesen Check überspringen. Nur falls der Wert explizit `0` ist (gehärtetes Image), dauerhaft aktivieren:

```
echo "kernel.unprivileged_userns_clone=1" | sudo tee /etc/sysctl.d/99-userns.conf
sudo sysctl --system
```

Capabilities von newuidmap/newgidmap prüfen (erwartet: `cap_setuid=ep` bzw. `cap_setgid=ep`; falls leer: `sudo apt install --reinstall uidmap`):

```
sudo getcap /usr/bin/newuidmap /usr/bin/newgidmap
```

Linger aktivieren, damit `/run/user/<uid>` und der User-systemd-Manager dauerhaft und ohne Login existieren.

```
sudo loginctl enable-linger gitlab-runner
```

4. Storage auf fuse-overlayfs festnageln

fuse-overlayfs explizit setzen, damit der Storage-Treiber unabhängig von Kernel- und Distro-Defaults reproduzierbar bleibt.

Verzeichnis anlegen:

```
sudo -u gitlab-runner mkdir -p /home/gitlab-runner/.config/containers
```

Datei `/home/gitlab-runner/.config/containers/storage.conf` als `gitlab-runner` anlegen (`sudo -u gitlab-runner vim /home/gitlab-runner/.config/containers/storage.conf`) mit folgendem Inhalt:

```
[storage]
driver = "overlay"

[storage.options.overlay]
mount_program = "/usr/bin/fuse-overlayfs"
```

5. Smoke-Test

buildah rootless braucht ein korrektes `HOME` (findet sonst die storage.conf nicht) und `XDG_RUNTIME_DIR`.

Test-Verzeichnis anlegen:

```
sudo -u gitlab-runner mkdir -p /home/gitlab-runner/smoketest
```

Datei `/home/gitlab-runner/smoketest/Containerfile` als `gitlab-runner` anlegen (`sudo -u gitlab-runner vim /home/gitlab-runner/smoketest/Containerfile`) mit folgendem Inhalt:

```
FROM registry.access.redhat.com/ubi9/ubi-minimal
RUN echo hello-rootless
```

Build ausführen:

```
RUNNER_UID=$(id -u gitlab-runner)

sudo -u gitlab-runner env HOME=/home/gitlab-runner XDG_RUNTIME_DIR=/run/user/$RUNNER_UID \
  buildah build -t smoketest:local /home/gitlab-runner/smoketest

sudo -u gitlab-runner env HOME=/home/gitlab-runner buildah images
```

Verifizieren:

```
sudo -u gitlab-runner env HOME=/home/gitlab-runner buildah info
```

Erwartet: `store.graphDriverName: overlay` und `host.security.rootless: true`. Danach kann das Test-Verzeichnis wieder weg: `sudo -u gitlab-runner rm -rf /home/gitlab-runner/smoketest`.

6. Runner registrieren

Wichtig beim neuen Authentication-Token-Workflow: `--tag-list`, `--locked`, `--run-untagged` usw. sind serverseitig und dürfen beim `register` nicht mehr angegeben werden (sonst FATAL). Tags werden in der GitLab-UI am Runner gesetzt.

```
sudo -iu gitlab-runner gitlab-runner register \
  --non-interactive \
  --url "https://gitlab.example.com" \
  --token "glrt-XXXX" \
  --executor "shell" \
  --description "rootless-buildah-debian13"
```

Tag am Runner in der UI setzen (Settings > CI/CD > Runners > Runner öffnen > Tags), passend zum `tags:`-Block der Pipeline, hier `rootless-buildah`. Ohne passendes Tag bleibt der Job auf pending.

`concurrent = 1` erzwingen, damit parallele Jobs sich nicht Cache/Workspace teilen, und verifizieren:

```
sudo -iu gitlab-runner sed -i 's/^concurrent = .*/concurrent = 1/' ~/.gitlab-
runner/config.toml
sudo -iu gitlab-runner gitlab-runner verify
```

7. Runner als User-systemd-Service

Verzeichnis anlegen:

```
sudo -u gitlab-runner mkdir -p /home/gitlab-runner/.config/systemd/user
```

Datei `/home/gitlab-runner/.config/systemd/user/gitlab-runner.service` als `gitlab-runner` anlegen (`sudo -u gitlab-runner vim /home/gitlab-runner/.config/systemd/user/gitlab-runner.service`) mit folgendem Inhalt:

```
[Unit]
Description=GitLab Runner (rootless)
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/usr/bin/gitlab-runner run --working-directory %h --config %h/.gitlab-runner/config.toml
Restart=always
RestartSec=5

[Install]
WantedBy=default.target
```

Aktivieren und starten:

```
RUNNER_UID=$(id -u gitlab-runner)

sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user daemon-reload
sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user enable --now gitlab-runner
sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user status gitlab-runner
```

8. Pipeline (Push in die GitLab Container Registry)

Datei `.gitlab-ci.yml` im Repo anlegen. `CI_REGISTRY`, `CI_REGISTRY_USER`, `CI_REGISTRY_PASSWORD` (= `CI_JOB_TOKEN`) und `CI_REGISTRY_IMAGE` liefert GitLab automatisch.

```
stages:
  - build

build-image:
  stage: build
  tags:
    - rootless-buildah
  variables:
    STORAGE_DRIVER: overlay
    BUILDAH_FORMAT: docker
  before_script:
    - export XDG_RUNTIME_DIR="${XDG_RUNTIME_DIR:-/run/user/$(id -u)}"
    - buildah login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" "$CI_REGISTRY"
  script:
    - buildah build -t "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA" .
    - buildah push "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA"
    - |
      if [ "$CI_COMMIT_BRANCH" = "$CI_DEFAULT_BRANCH" ]; then
        buildah tag "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA" "$CI_REGISTRY_IMAGE:latest"
        buildah push "$CI_REGISTRY_IMAGE:latest"
      fi
  after_script:
    - buildah logout "$CI_REGISTRY" || true
```

Es muss eine Build-Datei (`Containerfile` oder `Dockerfile`) im Kontext liegen, sonst bricht der Build mit "cannot find Containerfile or Dockerfile" ab. Liegt die Datei woanders, per `-f` und Kontextpfad angeben, z. B. `buildah build -f docker/Containerfile -t "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA" docker/`.

Debian-Besonderheit bei Kurznamen: Anders als Rocky liefert Debian keine `unqualified-search-registries` in `/etc/containers/registries.conf`. Ein `FROM alpine` bricht daher mit "short-name resolution" ab. Entweder in den Containerfiles voll qualifizierte Namen verwenden (`FROM docker.io/library/alpine`), oder systemweit ergänzen. Datei `/etc/containers/registries.conf` als

root editieren (`sudo vim /etc/containers/registries.conf`) und folgende Zeile ergänzen:

```
unqualified-search-registries = ["docker.io"]
```

9. Wartung: Image-Store aufräumen

Der Store unter `/home/gitlab-runner/.local/share/containers` wächst mit jedem Build. Manuell:

```
sudo -iu gitlab-runner buildah rmi --prune
```

Automatisiert als User-Timer. Datei `/home/gitlab-runner/.config/systemd/user/buildah-prune.service` als `gitlab-runner` anlegen (`sudo -u gitlab-runner vim /home/gitlab-runner/.config/systemd/user/buildah-prune.service`) mit folgendem Inhalt:

```
[Unit]
Description=Prune dangling buildah images

[Service]
Type=oneshot
Environment=HOME=%h
ExecStart=/usr/bin/buildah rmi --prune
```

Datei `/home/gitlab-runner/.config/systemd/user/buildah-prune.timer` als `gitlab-runner` anlegen (`sudo -u gitlab-runner vim /home/gitlab-runner/.config/systemd/user/buildah-prune.timer`) mit folgendem Inhalt:

```
[Unit]
Description=Weekly buildah image prune

[Timer]
OnCalendar=weekly
Persistent=true

[Install]
WantedBy=timers.target
```

Timer aktivieren:

```
RUNNER_UID=$(id -u gitlab-runner)
sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user daemon-reload
sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user enable --now
buildah-prune.timer
```

Bei knappem `/home` kann `graphroot` in der storage.conf auf ein größeres Volume gelegt werden.

10. Reboot-Test

Nach einem Neustart prüfen, ob der Runner durch Linger von allein hochkommt:

```
RUNNER_UID=$(id -u gitlab-runner)
sudo -u gitlab-runner env XDG_RUNTIME_DIR=/run/user/$RUNNER_UID systemctl --user status
gitlab-runner
```