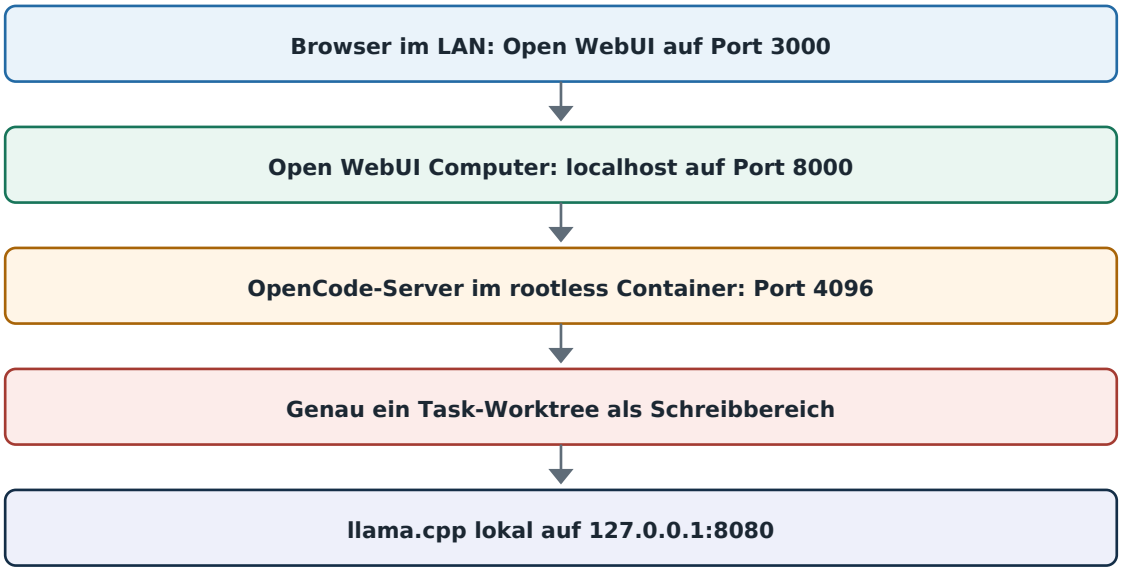
OpenCode erweitern

Open WebUI und Telegram auf Ubuntu 26.04

Vollständige Erweiterungsanleitung für die vorhandene lokale llama.cpp-/OpenCode-Installation mit Open WebUI Computer, Open WebUI im LAN und einem abgesicherten Telegram-Bot.



Zielbild

Open WebUI wird die zusätzliche LAN-Oberfläche. Open WebUI Computer vermittelt agentische Aufträge an den vorhandenen OpenCode-Server und betreibt den Telegram-Bot. Der Modellserver bleibt rein lokal; Computer wird nur auf localhost veröffentlicht.

Komponente	Aufgabe	Erreichbarkeit
llama.cpp	Lokale Modellinferenz	127.0.0.1:8080
OpenCode	Coding-Agent und Werkzeuge	Port 4096, passwortgeschützt
Computer	Workspace, Gateway und Telegram	127.0.0.1:8000
Open WebUI	Chat- und Agentenoberfläche	LAN-Port 3000
Telegram	Fernsteuerung per Long-Polling	kein eingehender Port

Stand: 29. August 2026. Zielsystem: Ubuntu 26.04 LTS, rootless Podman, vorhandener OpenCode-Worktree-Betrieb.

Inhaltsverzeichnis

1. Ergebnis und wichtige Grenzen

Diese Anleitung ist eine Erweiterung der vorhandenen PDF Lokaler Coding-Assistent - Ubuntu 26.04 - OpenCode Web. llama.cpp, das Coding-Modell, OpenCode, dessen Konfiguration und der einzelne Git-Worktree müssen bereits funktionieren.

1.1 Was danach verfügbar ist

- Open WebUI als zusätzliche Oberfläche unter `http://<LAN-IP>:3000`.
- Ein normales Modell für reine Unterhaltung ohne Datei- oder Shellzugriff.
- Ein agentisches Modell `cptr/<workspace>`, das OpenCode im gewählten Worktree ausführt.
- Ein privater Telegram-Bot mit Workspace- und Modellwahl, Sitzungsfortsetzung und Abbruchbefehl.
- Weiterhin genau ein schreibbar eingebundener Task-Worktree; keine Freigabe des Home-Verzeichnisses oder eines Container-Sockets.

1.2 Warum Open WebUI Computer benötigt wird

Open WebUI allein ist eine Modell- und Chatoberfläche. Wird es direkt mit llama.cpp verbunden, kann das Modell antworten, aber nicht selbstständig im Repository arbeiten. Open WebUI Computer stellt Workspaces als OpenAI-kompatible Modelle bereit, unterstützt OpenCode als natives Agenten-Backend und bietet die Telegram-Anbindung.

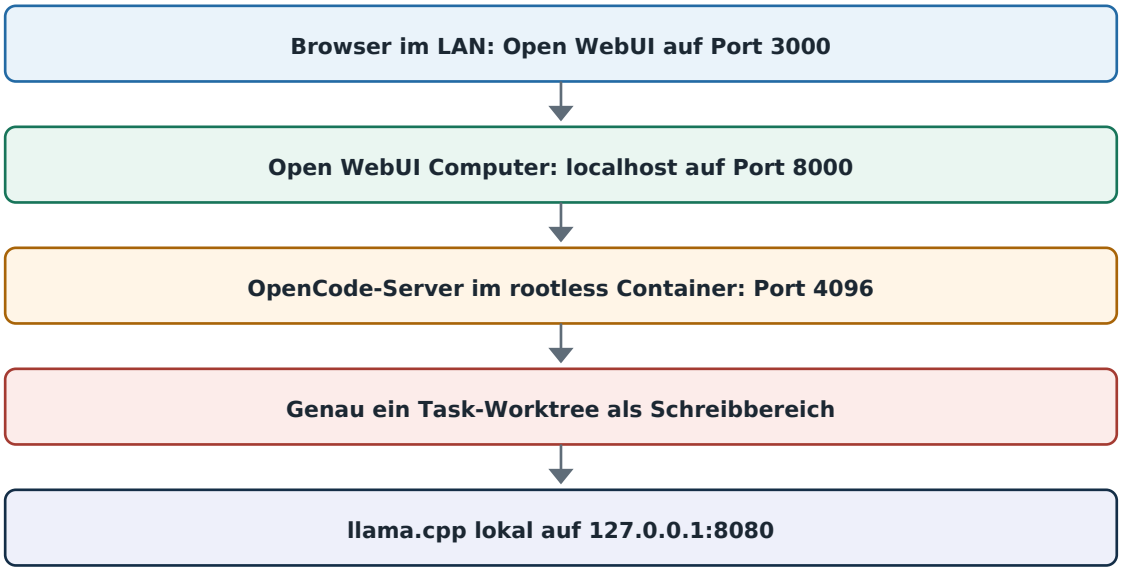
Zwei Produkte — Open WebUI und Open WebUI Computer sind getrennte Dienste. Open WebUI ist die LAN-Chatoberfläche. Computer ist die lokale Workspace-, Agenten- und Bot-Schicht auf Port 8000.

1.3 Sicherheitsgrenzen

Risiko	Gegenmaßnahme	Rest-Risiko
Unbeaufsichtigter Agent	rootless Container und genau ein Worktree	Dateien im Worktree können geändert oder gelöscht werden
Unbekannte Telegram-Nutzer	numerische Absender-ID als Whitelist	gestohlener Telegram-Account oder Bot-Token
Zugriff aus dem Internet	Long-Polling; keine eingehende Telegram-Portfreigabe	Nachrichten passieren Telegram-Server
Unbefugter LAN-Zugriff	Open-WebUI-Login und UFW auf das LAN begrenzen	HTTP ist im LAN unverschlüsselt
Git-Push oder Zugangsdaten	Git-Metadaten read-only; keine SSH-/Socket-Mounts	Geheimnisse im Worktree bleiben für den Agenten sichtbar

Telegram ist nicht rein lokal — Bot-Chats sind keine Telegram Secret Chats. Prompts, Antworten, Dateianhänge und möglicherweise Codeausschnitte werden über Telegram transportiert. Keine Passwörter, Tokens, privaten Schlüssel oder besonders vertraulichen Quelltexte per Telegram senden.

2. Architektur und Portplan



2.1 Kommunikationswege

1. Open WebUI greift direkt auf den lokalen llama.cpp-Endpunkt zu, wenn nur Modellchat benötigt wird.
2. Für einen agentischen Auftrag ruft Open WebUI den Computer-Gateway auf Port 8000 auf.
3. Computer übergibt den Auftrag an den passwortgeschützten OpenCode-Server.
4. OpenCode liest und verändert ausschließlich den gemounteten Task-Worktree.
5. Telegram empfängt Nachrichten per Long-Polling. Computer ordnet sie dem freigegebenen Workspace und dem OpenCode-Modell zu.

2.2 Porttabelle

Port	Bindung	Zweck	Firewall
8080	127.0.0.1	llama.cpp API	keine LAN-Regel
4096	0.0.0.0	OpenCode Web und API	wie bisher, LAN + Passwort
8000	127.0.0.1	Computer UI und Gateway	keine LAN-Regel
3000	0.0.0.0	Open WebUI	nur vertrauenswürdiges LAN

Keine Router-Freigabe

Keinen dieser Ports per Portweiterleitung ins Internet stellen. Ohne Reverse Proxy läuft Open WebUI im LAN per HTTP; das ist nur für ein vertrauenswürdiges Heim- oder Entwicklungsnetz geeignet.

3. Variablen und Vorabtests

3.1 Task- und Netzwerkvariablen setzen

Die Platzhalter an die eigene Installation anpassen:

```
export TASK_SLUG="task-001"

export WORKTREE_PATH="$HOME/opencode-worktrees/<PROJEKTNAME>/$TASK_SLUG"
export WORKTREE_PATH="$(realpath "$WORKTREE_PATH")"

export CONTAINER_WORKTREE="/workspaces/$TASK_SLUG"

export GIT_COMMON_DIR="$(
  git -C "$WORKTREE_PATH" \
    rev-parse --path-format=absolute --git-common-dir
)"

export LAN_IP="192.168.178.20"
export LAN_CIDR="192.168.178.0/24"
```

Netzwerkdaten bei Bedarf bestimmen:

```
ip -4 -brief address
ip -4 route
```

3.2 Pfade validieren

```
test -d "$WORKTREE_PATH" || {
  echo "Worktree fehlt: $WORKTREE_PATH"
  exit 1
}

test -d "$GIT_COMMON_DIR" || {
  echo "Git-Metadaten fehlen: $GIT_COMMON_DIR"
  exit 1
}

git -C "$WORKTREE_PATH" status --short

printf 'Host-Worktree:   %s\n' "$WORKTREE_PATH"
printf 'Container-Pfad: %s\n' "$CONTAINER_WORKTREE"
printf 'Git-Metadaten:  %s\n' "$GIT_COMMON_DIR"
printf 'Open WebUI:     http://%s:3000\n' "$LAN_IP"
```

3.3 Bestehende Dienste testen

```
systemctl --user is-active llama-server.service
curl -fsS http://127.0.0.1:8080/health

OPENCODE_USERNAME="$(
  sed -n 's/^OPENCODE_SERVER_USERNAME=//p' \
    "$HOME/.config/opencode/server.env"
)"

OPENCODE_PASSWORD="$(
  sed -n 's/^OPENCODE_SERVER_PASSWORD=//p' \
    "$HOME/.config/opencode/server.env"
)"

: "${OPENCODE_USERNAME:=opencode}"

curl -fsS \
  -u "${OPENCODE_USERNAME}:${OPENCODE_PASSWORD}" \
  http://127.0.0.1:4096/global/health

unset OPENCODE_PASSWORD
```

Der OpenCode-Test muss ein JSON-Objekt mit `healthy: true` liefern. Falls Port 4096 noch nicht läuft, wird er im nächsten Kapitel dauerhaft gestartet.

4. OpenCode für dauerhaften Betrieb umstellen

Der bisherige Startbefehl verwendete wahrscheinlich `--rm -it`. Für Telegram muss OpenCode im Hintergrund weiterlaufen. Der Container erhält einen stabilen Namen und mountet den Task unter demselben Pfad, den später Computer verwendet.

4.1 Alten Container beenden

Bei einem Vordergrundprozess zuerst Ctrl+C drücken. Danach:

```
podman stop -t 20 opencode-task-001 2>/dev/null || true
podman rm opencode-task-001 2>/dev/null || true

podman stop -t 20 opencode-agent 2>/dev/null || true
podman rm opencode-agent 2>/dev/null || true
```

Keine Projektdaten werden gelöscht	Diese Befehle entfernen nur alte Container. Worktree, Git-Daten, OpenCode-Konfiguration und persistente Cache-Verzeichnisse bleiben erhalten.

4.2 OpenCode im Hintergrund starten

```
podman run -d \
  --name opencode-agent \
  --network host \
  --restart unless-stopped \
  --cap-drop all \
  --security-opt no-new-privileges \
  --pids-limit 512 \
  --memory 20g \
  --read-only \
  --tmpfs /tmp:rw,nosuid,nodev,size=2g \
  --mount type=bind,src="$WORKTREE_PATH",dst="$CONTAINER_WORKTREE",rw \
  --mount type=bind,src="$GIT_COMMON_DIR",dst="$GIT_COMMON_DIR",ro \
  --mount type=bind,src="$HOME/.config/opencode",\
dst=/root/.config/opencode,ro \
  --mount type=bind,src="$HOME/.local/share/opencode-container",\
dst=/root/.local/share/opencode,rw \
  --mount type=bind,src="$HOME/.cache/opencode-container",\
dst=/root/.cache/opencode,rw \
  --env-file "$HOME/.config/opencode/server.env" \
  --env GIT_OPTIONAL_LOCKS=0 \
  --workdir "$CONTAINER_WORKTREE" \
  localhost/opencode-agent:ubuntu26 \
  web --hostname 0.0.0.0 --port 4096
```

4.3 Container und API prüfen

```
podman ps --filter name=opencode-agent
podman logs --tail 80 opencode-agent

OPENCODE_PASSWORD="$(
  sed -n 's/^OPENCODE_SERVER_PASSWORD=//p' \
    "$HOME/.config/opencode/server.env"
)"

curl -fsS \
  -u "opencode:${OPENCODE_PASSWORD}" \
  http://127.0.0.1:4096/global/health

unset OPENCODE_PASSWORD
```

5. Open WebUI Computer installieren

Computer verwaltet Workspaces, Chat-Sitzungen, Agentenprofile, den OpenAI-kompatiblen Gateway und den Telegram-Bot. Seine Daten liegen in einem eigenen Podman-Volume. Der Dienst wird nur auf localhost veröffentlicht.

5.1 Image und Volume

```
podman pull ghcr.io/open-webui/computer:latest
podman volume create cptr-data
```

5.2 Container starten

```
podman run -d \
  --name cptr \
  --publish 127.0.0.1:8000:8000 \
  --restart unless-stopped \
  --security-opt no-new-privileges \
  --pids-limit 512 \
  --memory 4g \
  --mount type=volume,src=cptr-data,dst=/data \
  --mount type=bind,src="$WORKTREE_PATH",dst="$CONTAINER_WORKTREE",rw \
  --mount type=bind,src="$GIT_COMMON_DIR",dst="$GIT_COMMON_DIR",ro \
  --env GIT_OPTIONAL_LOCKS=0 \
  --workdir "$CONTAINER_WORKTREE" \
  ghcr.io/open-webui/computer:latest
```

5.3 Einrichtungsadresse öffnen

```
podman ps --filter name=cptr
podman logs --tail 120 cptr
```

In den Logs erscheint einmalig eine Adresse ähnlich `http://localhost:8000/?token=...`. Sie funktioniert nur, solange noch kein Benutzerkonto existiert. Adresse auf dem Ubuntu-Rechner öffnen und das Administratorkonto anlegen.

Alternative für einen Rechner ohne Desktop

```
ssh -L 8000:127.0.0.1:8000 <BENUTZER>@<LAN-IP-DES-SERVERS>
```

Anschließend auf dem eigenen Rechner `http://127.0.0.1:8000` öffnen.

5.4 Workspace hinzufügen

1. In Computer einen neuen Workspace hinzufügen.
2. Als Pfad `/workspaces/task-001` verwenden.
3. Als Anzeigenamen `task-001` eintragen.
4. Prüfen, dass Dateiansicht und Git-Status genau diesen Worktree zeigen.

Keine breiten Mounts	Nicht das gesamte Home-Verzeichnis, den Elternordner aller Worktrees, `~/.ssh`, das Dateisystem-Wurzelverzeichnis, einen SSH-Agenten oder einen Docker-/Podman-Socket einbinden.

6. Computer mit OpenCode verbinden

6.1 OpenCode-Passwort anzeigen

```
sed -n 's/^OPENCODE_SERVER_PASSWORD=//p' \
  "$HOME/.config/opencode/server.env"
```

6.2 Agentenprofil anlegen

In Computer: Settings - Admin - Agents - neues Profil.

Feld	Wert
Name	Local OpenCode
Type	OpenCode
Profile ID	opencode-local
Command	opencode
Server URL	http://host.containers.internal:4096
Password	Wert aus server.env
Mode	Enabled oder Auto

Computer kann OpenCode selbst starten oder sich über Server-URL und Passwort mit einem vorhandenen OpenCode-Server verbinden. In diesem Aufbau wird der bereits gehärtete OpenCode-Container verwendet.

6.3 Hostnamen des Container-Hosts prüfen

```
podman exec cptr getent hosts host.containers.internal
```

Falls keine Adresse ausgegeben wird, Computer neu erzeugen:

```
podman stop cptr
podman rm cptr
```

Danach den Startbefehl aus Kapitel 5 wiederholen und zusätzlich diese Option aufnehmen:

```
--add-host host.containers.internal:host-gateway
```

6.4 Modell auswählen und lesend testen

Nach erfolgreicher Erkennung erscheint ein Modell ähnlich `agent:opencode-local/<provider>/<modell>`. Das Modell auswählen, das auf den lokalen llama.cpp-Alias `qwen2.5-coder-7b-q4km` verweist.

```
Arbeite ausschließlich lesend.

Zeige den aktuellen Branch, git status und eine kurze
Zusammenfassung des Projekts. Verändere keine Dateien und
führe keine Befehle mit Schreibwirkung aus.
```

Danach auf dem Host kontrollieren:

```
git -C "$WORKTREE_PATH" status --short
```

7. Open WebUI installieren

Open WebUI benötigt in diesem Aufbau keinen eigenen GPU-Zugriff. Die Inferenz bleibt bei llama.cpp. Deshalb wird das normale Image verwendet, nicht das Nvidia-CUDA- oder das gebündelte Ollama-Image.

7.1 Secret und persistentes Volume

```
install -m 700 -d "$HOME/.config/open-webui"

OPENWEBUI_SECRET="$(openssl rand -hex 32)"

printf 'WEBUI_SECRET_KEY=%s\n' "$OPENWEBUI_SECRET" \
  > "$HOME/.config/open-webui/open-webui.env"

chmod 600 "$HOME/.config/open-webui/open-webui.env"
unset OPENWEBUI_SECRET

podman volume create open-webui-data
podman pull ghcr.io/open-webui/open-webui:main
```

7.2 Administratorkonto nur über localhost erstellen

```
podman run -d \
  --name open-webui-bootstrap \
  --publish 127.0.0.1:3000:8080 \
  --security-opt no-new-privileges \
  --pids-limit 512 \
  --memory 4g \
  --env-file "$HOME/.config/open-webui/open-webui.env" \
  --mount type=volume,src=open-webui-data,dst=/app/backend/data \
  ghcr.io/open-webui/open-webui:main
```

```
podman logs --tail 120 open-webui-bootstrap
curl -I http://127.0.0.1:3000
```

Im Browser `http://127.0.0.1:3000` öffnen. Das erste Konto wird Administrator. Direkt danach im Adminbereich die Registrierung weiterer Benutzer deaktivieren.

7.3 Bootstrap-Container entfernen

```
podman stop open-webui-bootstrap
podman rm open-webui-bootstrap
```

Konto, Chats und Einstellungen bleiben im Volume erhalten.

7.4 Open WebUI im LAN starten

```
podman run -d \
  --name open-webui \
  --network host \
  --restart unless-stopped \
  --security-opt no-new-privileges \
  --pids-limit 512 \
  --memory 4g \
  --env-file "$HOME/.config/open-webui/open-webui.env" \
  --env PORT=3000 \
  --env ENABLE_SIGNUP=False \
  --env WEBUI_URL="http://${LAN_IP}:3000" \
  --mount type=volume,src=open-webui-data,dst=/app/backend/data \
  ghcr.io/open-webui/open-webui:main
```

7.5 Dienst prüfen

```
podman ps --filter name=open-webui
podman logs --tail 120 open-webui
curl -fsS http://127.0.0.1:3000/health
```

8. Firewall und LAN-Zugriff

8.1 UFW-Regeln

```
sudo ufw allow from "$LAN_CIDR" to any port 3000 proto tcp \
  comment 'Open WebUI LAN'

sudo ufw deny 3000/tcp
sudo ufw status numbered
```

Die spezifische Allow-Regel muss vor der allgemeinen Deny-Regel stehen. Bei bereits vielen UFW-Regeln die Reihenfolge über `ufw status numbered` prüfen.

8.2 Zugriff von einem zweiten Gerät

```
http://192.168.178.20:3000
```

Die Beispieladresse durch die tatsächliche LAN-IP ersetzen. Anmelden und prüfen, dass keine Registrierung für weitere Konten angeboten wird.

HTTP im LAN	Ohne Reverse Proxy gibt es kein TLS. Benutzername, Passwort und Chatdaten werden im lokalen Netz nicht transportverschlüsselt. Nur in einem vertrauenswürdigen Netz einsetzen; keine offenen oder fremden WLANs.

9. llama.cpp als normales Chatmodell einrichten

9.1 Verbindung hinzufügen

In Open WebUI: Admin Panel - Settings - Connections - Add Connection.

Feld	Wert
Typ	OpenAI-kompatibel
Base URL	http://127.0.0.1:8080/v1
API Key	local
Model IDs	zunächst leer; sonst qwen2.5-coder-7b-q4km
Prefix	optional local

Verbindung testen, speichern und das Modell im Chat auswählen.

9.2 Modellchat testen

```
Schreibe eine Java-Methode, die eine Liste von Strings
nach Länge und anschließend alphabetisch sortiert.
Führe keine Befehle aus.
```

Keine Werkzeuge Dieses direkt mit llama.cpp verbundene Modell kann Text erzeugen, hat aber keinen Zugriff auf Worktree, Git oder Shell. Es ist für Fragen, Erklärungen und Codeentwürfe gedacht.

10. Computer als agentisches Modell einrichten

10.1 Gateway-Key erzeugen

1. Computer unter `http://127.0.0.1:8000` öffnen.
2. Settings - Admin - Gateway aufrufen.
3. Das lokale OpenCode-Modell als Gateway-Modell auswählen.
4. Neuen API-Key erzeugen und den Wert `sk-cptr-...` sofort kopieren.

Gateway-Key schützen	Der Schlüssel wird nur einmal vollständig angezeigt und erlaubt unbeaufsichtigte agentische Aufgaben. Wie ein SSH-Zugang behandeln und nicht in Git oder Chatnachrichten speichern.

10.2 Verbindung in Open WebUI hinzufügen

In Open WebUI eine weitere OpenAI-kompatible Verbindung anlegen:

Feld	Wert
Base URL	http://127.0.0.1:8000/v1
API Key	sk-cptr-...
Prefix	leer
Models	automatisch erkennen

10.3 Custom Headers

In den erweiterten Verbindungseinstellungen eintragen:

```
{
  "X-OpenWebUI-Chat-Id": "{{CHAT_ID}}",
  "X-OpenWebUI-Message-Id": "{{MESSAGE_ID}}",
  "X-OpenWebUI-User-Message-Id": "{{USER_MESSAGE_ID}}",
  "X-OpenWebUI-User-Message-Parent-Id": "{{USER_MESSAGE_PARENT_ID}}",
  "X-OpenWebUI-Task": "{{TASK}}"
}
```

Diese Header erhalten die Sitzungszuordnung bei Folgefragen, Bearbeitungen und Regenerierungen. Die vollständige Header-Funktion benötigt Open WebUI 0.10.0 oder neuer.

10.4 Gateway per curl testen

```
read -rsp "Computer Gateway-Key: " CPTR_KEY
printf '\n'

curl -fsS \
  -H "Authorization: Bearer $CPTR_KEY" \
  http://127.0.0.1:8000/v1/models

unset CPTR_KEY
```

Die Modellliste muss unter anderem `cptr/task-001` enthalten.

10.5 Erster agentischer Test

In Open WebUI das Modell cptr/task-001 auswählen:

```
Arbeite ausschließlich lesend.

1. Nenne den aktuellen Branch.
2. Führe git status aus.
3. Fasse README und Projektstruktur kurz zusammen.
4. Verändere keine Datei.
```

```
git -C "$WORKTREE_PATH" status --short
```

Unbeaufsichtigter Gateway	Open-WebUI-Gateway-Aufträge können nicht für jeden einzelnen Werkzeugaufruf auf eine Bestätigung warten. Sie laufen mit voller Werkzeugfreigabe. Die harte Grenze ist der Container und der einzelne Worktree-Mount.

11. Telegram-Bot erstellen

11.1 BotFather verwenden

1. In Telegram einen privaten Chat mit `@BotFather` öffnen.
2. Den Befehl `/newbot` senden.
3. Anzeigenamen und eindeutigen Bot-Benutzernamen vergeben.
4. Den erzeugten Token kopieren und vertraulich behandeln.
5. Optional mit `/setjoingroups` den Gruppenbeitritt deaktivieren.

11.2 Numerische Telegram-Benutzer-ID bestimmen

Bei Bedarf jq installieren:

```
sudo apt update
sudo apt install -y jq
```

Dem neuen Bot zunächst in Telegram eine Nachricht senden, beispielsweise `/start`. Solange der Bot noch nicht in Computer gestartet wurde:

```
read -rsp "Telegram Bot-Token: " TELEGRAM_TOKEN
printf '\n'

curl -fsS \
  "https://api.telegram.org/bot${TELEGRAM_TOKEN}/getUpdates" \
  | jq -r '.result[-1].message.from.id'

unset TELEGRAM_TOKEN
```

Die Ausgabe ist die eigene numerische Benutzer-ID. Bei `null` dem Bot erneut schreiben, einige Sekunden warten und den Befehl wiederholen.

Nur ein Long-Polling-Client	Nach dem Start in Computer nicht parallel weiter getUpdates aufrufen. Computer übernimmt dann das Long-Polling. Für Telegram ist weder ein Webhook noch ein öffentlicher Server-Port nötig.

12. Telegram in Computer konfigurieren

12.1 Bot anlegen

In Computer: Settings - Admin - Bots - Create Bot.

Feld	Wert
Platform	Telegram
Credential	Token von BotFather
Workspace	task-001
Model	lokales OpenCode-Agentenmodell
Allowed senders	ausschließlich die eigene numerische Benutzer-ID

1. Token einfügen und Verify ausführen.
2. Workspace und Modell auswählen.
3. Allowed senders eintragen und nochmals kontrollieren.
4. Bot starten.

Whitelist ist Pflicht Eine leere Allowed-senders-Liste bedeutet, dass jeder Finder des Bots agentische Aufträge auslösen könnte. Der Bot darf nicht gestartet werden, solange die Liste leer ist.

12.2 Bot-Kommandos

Befehl	Funktion
/help	verfügbare Befehle anzeigen
/new oder /reset	neue Unterhaltung beginnen
/stop	laufenden Task abbrechen
/retry	letzte Nachricht erneut ausführen
/model [id]	Modell anzeigen oder wechseln
/workspaces	verfügbare Workspaces anzeigen
/workspace	Workspace wechseln und neuen Chat starten

12.3 Funktionstest

Im privaten Bot-Chat nacheinander senden:

```
/help
/workspaces
/workspace task-001
/model
/new
```

Danach ein rein lesender Auftrag:

```
Arbeite nur lesend.

Zeige den aktuellen Branch und git status.
Fasse danach die oberste Verzeichnisstruktur zusammen.
Verändere keine Dateien und installiere keine Pakete.
```

Auf dem Host kontrollieren:

```
git -C "$WORKTREE_PATH" status --short
```

13. Einen echten Coding-Task durchführen

13.1 Auftragsschablone

```
Du arbeitest ausschließlich im aktuell geöffneten Worktree.

Aufgabe:
<genaue Aufgabe>

Akzeptanzkriterien:
1. <Kriterium>
2. <Kriterium>
3. <Kriterium>

Erlaubt:
- Dateien im Worktree lesen und bearbeiten
- vorhandene Tests und Build-Befehle ausführen
- neue Tests innerhalb des Worktrees hinzufügen

Nicht erlaubt:
- git push oder Branchwechsel
- Commits erzeugen
- Zugangsdaten lesen oder verändern
- Systempakete installieren
- Dateien außerhalb des Worktrees verändern
- Docker- oder Podman-Sockets verwenden

Vorgehen:
1. Analysiere zuerst den Ist-Zustand.
2. Nenne deinen kurzen Plan.
3. Implementiere die Änderung.
4. Führe die relevanten Tests aus.
5. Fasse Dateien, Tests und Risiken zusammen.
6. Stoppe bei unklaren oder destruktiven Schritten.
```

13.2 Änderungen auf dem Host prüfen

```
git -C "$WORKTREE_PATH" status --short
git -C "$WORKTREE_PATH" diff --stat
git -C "$WORKTREE_PATH" diff
git -C "$WORKTREE_PATH" diff --check
```

13.3 Tests außerhalb des Agenten wiederholen

Beispiele:

```
cd "$WORKTREE_PATH"
./gradlew test
```

```
cd "$WORKTREE_PATH"
mvn test
```

13.4 Commit weiterhin selbst erstellen

```
cd "$WORKTREE_PATH"
git add -p
git commit
```

> **Review bleibt Pflicht** Telegram ist nur ein weiterer Eingabekanal. Die lokale Modellqualität ändert sich dadurch nicht. Diff und Tests vor jedem Commit manuell prüfen.

14. Automatischer Start

14.1 Rootless-Benutzerdienste nach Neustart

```
sudo loginctl enable-linger "$USER"
loginctl show-user "$USER" -p Linger
```

Erwartet wird `Linger=yes`.

14.2 systemd-Benutzerdienst anlegen

```
install -m 700 -d "$HOME/.config/systemd/user"
install -m 600 /dev/null \
  "$HOME/.config/systemd/user/local-ai-stack.service"

nano "$HOME/.config/systemd/user/local-ai-stack.service"
```

Dateiinhalt:

```
[Unit]
Description=Lokaler AI-Stack mit OpenCode, Computer und Open WebUI
Wants=network-online.target llama-server.service
After=network-online.target llama-server.service

[Service]
Type=oneshot
RemainAfterExit=yes

ExecStart=/usr/bin/podman start opencode-agent
ExecStart=/usr/bin/podman start cptr
ExecStart=/usr/bin/podman start open-webui

ExecStop=/usr/bin/podman stop -t 30 open-webui
ExecStop=/usr/bin/podman stop -t 30 cptr
ExecStop=/usr/bin/podman stop -t 30 opencode-agent

TimeoutStartSec=0
TimeoutStopSec=120

[Install]
WantedBy=default.target
```

14.3 Aktivieren und prüfen

```
systemctl --user daemon-reload
systemctl --user enable local-ai-stack.service
systemctl --user start local-ai-stack.service

systemctl --user status local-ai-stack.service
podman ps
```

14.4 Abnahmetest nach einem Neustart

```
systemctl --user is-active llama-server.service
systemctl --user is-active local-ai-stack.service
podman ps

curl -fsS http://127.0.0.1:8080/health
curl -fsS http://127.0.0.1:3000/health
```

Abschließend dem Telegram-Bot `/help` senden.

15. Auf einen neuen Task-Worktree wechseln

15.1 Neuen Worktree erzeugen

```
cd <REPO>

git worktree add \
  -b task/task-002 \
  "$HOME/opencode-worktrees/<PROJEKTNAME>/task-002" \
  <BASISBRANCH>
```

15.2 Agentencontainer stoppen und entfernen

```
systemctl --user stop local-ai-stack.service

podman rm opencode-agent
podman rm cptr
```

Volumes behalten	Nicht cptr-data, open-webui-data, OpenCode-Konfigurationsverzeichnisse oder Git-Worktrees entfernen. Nur die beiden Container werden mit neuen Mounts wieder erzeugt.

15.3 Variablen auf den neuen Task setzen

```
export TASK_SLUG="task-002"

export WORKTREE_PATH="$HOME/opencode-worktrees/<PROJEKTNAME>/$TASK_SLUG"
export WORKTREE_PATH="$(realpath "$WORKTREE_PATH")"

export CONTAINER_WORKTREE="/workspaces/$TASK_SLUG"

export GIT_COMMON_DIR="$(
  git -C "$WORKTREE_PATH" \
    rev-parse --path-format=absolute --git-common-dir
)"
```

OpenCode und Computer mit den Befehlen aus Kapitel 4 und 5 neu erzeugen. In Computer den Workspace `/workspaces/task-002` hinzufügen. Danach die Bot-Konfiguration auf den neuen Workspace umstellen oder in Telegram `/workspace task-002` verwenden.

16. Betrieb, Updates und Backups

16.1 Status und Logs

```
podman ps
podman logs --tail 200 open-webui
podman logs --tail 200 cptr
podman logs --tail 200 opencode-agent

journalctl --user -u llama-server.service -n 200 --no-pager
```

16.2 Container-Images aktualisieren

```
podman pull ghcr.io/open-webui/open-webui:main
podman pull ghcr.io/open-webui/computer:latest
```

Ein Pull allein ersetzt keinen bestehenden Container. Vor einem Upgrade Volumes sichern, Container mit denselben Optionen neu erzeugen und danach alle Abnahmetests wiederholen. Für einen reproduzierbaren Dauerbetrieb später konkrete Release-Tags statt rollender Tags verwenden.

16.3 Volumes sichern

```
BACKUP_DIR="$HOME/ai-backups/$(date +%F-%H%M)"
mkdir -p "$BACKUP_DIR"

podman stop open-webui cptr

podman volume export open-webui-data \
  > "$BACKUP_DIR/open-webui-data.tar"

podman volume export cptr-data \
  > "$BACKUP_DIR/cptr-data.tar"

podman start cptr open-webui

ls -lh "$BACKUP_DIR"
```

16.4 Zusätzlich sichern

- `~/.config/open-webui/open-webui.env`
- `~/.config/opencode` und `~/.config/opencode/server.env`
- die AGENTS.md des Projekts und eigene OpenCode-Regeln,
- alle nicht eingecheckten Änderungen der aktiven Worktrees.

17. Fehlerdiagnose

17.1 Open WebUI ist nicht erreichbar

```
podman ps --filter name=open-webui
podman logs --tail 200 open-webui
ss -ltn | grep ':3000'
sudo ufw status numbered
```

17.2 Lokales Modell fehlt

```
curl -fsS http://127.0.0.1:8080/v1/models
podman logs --tail 200 open-webui
```

In Open WebUI kontrollieren: Base URL `http://127.0.0.1:8080/v1`, API-Key `local`.

17.3 Computer erreicht OpenCode nicht

```
OPENCODE_PASSWORD="$(
  sed -n 's/^OPENCODE_SERVER_PASSWORD=//p' \
    "$HOME/.config/opencode/server.env"
)"

curl -fsS \
  -u "opencode:${OPENCODE_PASSWORD}" \
  http://127.0.0.1:4096/global/health

unset OPENCODE_PASSWORD

podman exec cptr getent hosts host.containers.internal
podman logs --tail 200 opencode-agent
podman logs --tail 200 cptr
```

17.4 cptr/task-001 fehlt in Open WebUI

```
read -rsp "Computer Gateway-Key: " CPTR_KEY
printf '\n'

curl -fsS \
  -H "Authorization: Bearer $CPTR_KEY" \
  http://127.0.0.1:8000/v1/models \
  | jq

unset CPTR_KEY
```

Wenn das Modell hier erscheint, liegt der Fehler in der Open-WebUI-Verbindung. Die Base URL muss auf `http://127.0.0.1:8000/v1` enden.

17.5 Telegram antwortet nicht

```
podman ps --filter name=cptr
podman logs --tail 300 cptr
```

- Bot-Status in Computer muss Running sein.
- Token muss erfolgreich verifiziert sein.
- Allowed senders muss die eigene numerische ID enthalten.
- Workspace und OpenCode-Modell müssen vorhanden sein.
- Kein anderes Programm darf gleichzeitig getUpdates aufrufen.

17.6 Telegram-Task hängt

Im Bot-Chat zuerst:

```
/stop
/new
```

Danach Logs prüfen:

```
podman logs --tail 300 cptr
podman logs --tail 300 opencode-agent
journalctl --user -u llama-server.service -n 300 --no-pager
```

18. Sicherheits- und Abnahmecheck

18.1 Pflichtregeln

1. Keine Router-Portweiterleitung für 3000, 4096, 8000 oder 8080.
2. Telegram Allowed senders enthält ausschließlich bekannte numerische IDs.
3. Bot bleibt aus Gruppen fern.
4. Computer und OpenCode mounten nur einen einzelnen Task-Worktree.
5. Git-Metadaten bleiben read-only.
6. Kein Home-, SSH-, Container-Socket- oder SSH-Agent-Mount.
7. Gateway-Key und Telegram-Token werden nicht versioniert oder verschickt.
8. Keine Geheimnisse oder vertraulichen Codeausschnitte per Telegram.
9. Diff und Tests werden vor jedem Commit auf dem Host geprüft.
10. Commit und Push bleiben manuelle Host-Aktionen.

18.2 Abnahmetabelle

Nr.	Prüfung	Sollzustand
1	llama.cpp	127.0.0.1:8080 gesund
2	OpenCode	Health-API mit Passwort erreichbar
3	Computer	nur 127.0.0.1:8000
4	OpenCode-Profil	lokales Modell auswählbar
5	Open WebUI	LAN-Port 3000, Anmeldung erforderlich
6	Registrierung	deaktiviert
7	Modellchat	antwortet ohne Datei-/Shellzugriff
8	cptr-Modell	cptr/task-001 sichtbar
9	Lesetest	kein veränderter Git-Status
10	Telegram	nur erlaubte Benutzer-ID
11	Bot-Kommandos	/help, /workspace, /stop funktionieren
12	Neustart	alle Dienste starten automatisch

Fertig

Wenn alle Punkte erfüllt sind, kann derselbe isolierte OpenCode-Agent über OpenCode Web, Open WebUI und Telegram angesprochen werden. Modellport und Computer-Gateway bleiben lokal; nur Open WebUI und das bereits abgesicherte OpenCode Web sind im vertrauenswürdigen LAN erreichbar.

19. Befehlsreferenz

19.1 Zustände prüfen

```
systemctl --user is-active llama-server.service
systemctl --user is-active local-ai-stack.service
curl -fsS http://127.0.0.1:8080/health
curl -fsS http://127.0.0.1:3000/health
podman ps
git -C "$WORKTREE_PATH" status --short
sudo ufw status numbered
```

19.2 Stack stoppen und starten

```
systemctl --user stop local-ai-stack.service
systemctl --user start local-ai-stack.service
```

19.3 Logs verfolgen

```
podman logs -f open-webui
```

```
podman logs -f cptr
```

```
podman logs -f opencode-agent
```

20. Offizielle Quellen und Versionshinweise

Alle Quellen wurden am 29. August 2026 geprüft. Open WebUI und Computer entwickeln sich schnell; Menünamen und einzelne Felder können sich mit späteren Releases verschieben. Vor Updates Release Notes und aktuelle Dokumentation prüfen.

Thema	Offizielle Quelle
Open WebUI Schnellstart und Podman	https://docs.openwebui.com/getting-started/quick-start/
Open WebUI Umgebungsvariablen	https://docs.openwebui.com/reference/env-configuration/
Computer Docker-Installation	https://docs.openwebui.com/ecosystem/computer/install/docker/
OpenCode als Computer-Backend	https://docs.openwebui.com/ecosystem/computer/ai/coding-agents/
Computer in Open WebUI	https://docs.openwebui.com/ecosystem/computer/automate/open-webui/
Computer Gateway API	https://docs.openwebui.com/ecosystem/computer/reference/gateway-api/
Computer Messaging-Bots	https://docs.openwebui.com/ecosystem/computer/automate/messaging-bots/
Telegram Bot API	https://core.telegram.org/bots/api
OpenCode Server API	https://opencode.ai/docs/server/

20.1 Wesentliche Versionsannahmen

- Ubuntu 26.04 LTS mit rootless Podman ist bereits eingerichtet.
- Open WebUI verwendet das offizielle Image `ghcr.io/open-webui/open-webui:main`.
- Computer verwendet das offizielle Image `ghcr.io/open-webui/computer:latest`.
- Die vollständigen Open-WebUI-Sitzungsheader setzen Version 0.10.0 oder neuer voraus.
- Computer-Telegram verwendet Long-Polling; es ist kein Webhook oder öffentlicher Port nötig.
- Gateway- und Bot-Aufträge laufen unbeaufsichtigt mit voller Werkzeugfreigabe. Die Containergrenze ist deshalb verpflichtend.