TECH TALK · APRIL 2026

Einführung in OpenClaw

Ein lokaler AI-Assistent. Auf WhatsApp & Co. In 90 Minuten live.

Marcel Wege · byte5 GmbH

Marcel Wege : CTO @ byte5

- **43, verheiratet, in Langen (Hessen)** — fast 9 Jahre byte5 in Frankfurt, Prokurist & CTO
- **Tech-Stack :** Laravel · Node.js · Umbraco · alles, was komplexe Projekte zum Laufen bringt
- **Speaker** auf Tech-Events — u.a. Umbraco Festival Deutschland
- **Privat :** Formel-1-Fan, Rennrad 🚴 **(daher heute Strava in der Demo)**, Fotografie 📷

Kontakt : mwege@byte5.de · LinkedIn

byte5 in 30 Sekunden

- Softwareunternehmen aus Frankfurt am Main, seit 2004
- Geschäftskritische Web- und KI-Lösungen für Mittelstand und Großunternehmen
- **Produktiv-Stack:** Umbraco · Laravel · Medusa
- **byte5 Labs:** OpenClaw · IOTA · Agentic AI
- Unsere Expert:innen begleiten Projekte von der Strategie bis zum Betrieb

> In unseren byte5 Labs arbeiten wir intensiv mit OpenClaw — die Learnings fließen direkt in die Enterprise-AI-Lösungen, die wir für unsere Kund:innen bauen.

Lokaler AI-Assistent. Auf deinen Channels.

Idee: ein persönlicher AI-Assistent, der dich überall erreicht — ohne deine Daten an einen Cloud-Vendor abzugeben.

- **Gateway** als lokale Steuerebene auf deinem Server oder Laptop
- Inbox über bestehende Kanäle: WhatsApp, Telegram, Slack, iMessage, Signal, Matrix, …
- **Skills** als versionierbare Markdown-Dateien — `SKILL.md` pro Fähigkeit
- LLM frei wählbar: Anthropic, OpenAI, OpenRouter — oder lokal via Ollama
- MIT-lizenziert, läuft auf Linux / macOS / Windows (WSL2)

Roter Faden

1. **Setup** — Ubuntu-Container als VPS-Simulator
2. **Onboarding** — OpenClaw installieren, Persona aushandeln
3. **WhatsApp** — QR scannen, Bot antwortet
4. **Skills** — Markdown-Skill vom Host editieren
5. **Strava (DIY)** — externe API mit OAuth selbst anbinden
6. **Cron** — proaktive Coach-Pushes
7. **Web Search** — zweiter Skill, API-Key statt OAuth
8. **Komposition** — Strava + Search + USER.md = Event-Empfehlungen
9. **ClawHub** — Skills aus der Registry installieren
10. **Persistenz** — Container neustarten, alles bleibt

> Am Ende: du weißt, wie du OpenClaw auf deinem VPS einsetzt.

Phase 1 : Setup

Drei Wege, OpenClaw zu installieren

Pfad	Wann?	Aufwand
VPS **Bare-Metal Ubuntu**	Produktion auf deinem eigenen Server	gering
DOCKER **VPS-Simulator**	Lokales Üben mit identischen VPS-Befehlen	mittel
DOCKER **Lean Local**	Schnell ausprobieren, ~250 MB Image	minimal

Alle drei Pfade führen zu denselben drei Schlüssel-Befehlen :

```
npm install -g openclaw@latest
openclaw onboard
openclaw channels add whatsapp
```

DOCKER

Live: Container hochfahren

```
cp .env.example .env              # leerer Sentinel
docker compose up -d --build
docker compose exec openclaw bash

# Im Container — exakt die VPS-Befehle:
node --version                    # v24.x
npm install -g openclaw@latest
openclaw --version
```

- Persistenz über zwei Bind Mounts: `./openclaw` und `./workspace`
- Vom Host editierbar mit jedem Editor
- Reset zwischen Versuchen: `docker compose down -v && up -d`

Phase 2 : Onboarding

Der Wizard übernimmt

```
openclaw onboard --install-daemon
```

Frage	Antwort für die Demo
Workspace-Pfad	`~/.openclaw/workspace` (Default behalten)
LLM-Provider	OpenAI Codex (ChatGPT-Subscription)
Modell	`openai-codex/gpt-5.5`
Auth	**OAuth-URL-Flow** — Wizard druckt Link, im Browser bei ChatGPT autorisieren
Skills	erstmal überspringen — kommen in Phase 5

Auf einem echten VPS landet der Gateway als **systemd-User-Service** und übersteht Reboots.

Browser statt Terminal

Im Alltag willst du das **WebUI**. Zwei Anpassungen, dann läuft's:

DOCKER **Gateway-Bind auf `auto`** — Loopback + Bridge-Interface, kein Public-Listener:

```
openclaw config set gateway.bind auto      # + Port mappen in docker-compose.yml
```

BEIDE **URL inkl. Auth-Token bauen** (Token steht in der Config, `dashboard` druckt es aus Sicherheit nicht):

```
echo "http://localhost:18789/?token=$(jq -r .gateway.auth.token ~/.openclaw/openclaw.json)"
```

> VPS `ssh -L 18789:localhost:18789 user@vps` öffnen, gleiches Echo-Snippet auf dem VPS, URL **lokal** im Browser öffnen. SSH-Tunnel verschlüsselt, `bind` darf hier `loopback` bleiben.

Der Agent fragt zurück

Nach dem ersten Start handelt OpenClaw seine Identität mit dir aus — **im Chat**, nicht im Wizard.

- **Name des Agents** — wie soll ich heißen?
- **Dein Name** — wie darf ich dich nennen?
- **Rolle** — was bin ich für dich?
- **Vibe** — ruhig · warm · direkt · witzig · nerdig
- **Emoji** — die Signatur

→ Schreibt `IDENTITY.md`, `USER.md`, `SOUL.md` & Co. in den Workspace — versionierbar, vom Host editierbar.

> **Heute Abend live:** byte5 Demo Bot 🏋️ · witzig & sarkastisch · Personal-Coach für Marcel

Phase 3 : WhatsApp

WEBUI

Channel aktivieren

> **Wichtig:** Communications (Config) ≠ Channels (Laufzeit). Aktivieren passiert in **Communications**, sichtbar wird's danach in **Channels**.

1. Linke Sidebar: **Communications**
2. Suchfeld: `whatsapp` → Section springt rein
3. **Channels → WhatsApp:** **Enabled**-Toggle umlegen
4. **Save** unten in der Card

> **Finger weg von "Add Entry" unter Accounts** — das legt einen zweiten Account neben dem `default` an, der beim QR-Scan automatisch entsteht. Beide kämpfen um dieselbe Nummer → Reconnect-Loop, Bot ist tot. Erst Save, dann den `default` -Account einfach scannen lassen.

QR scannen, los geht's

1. **Handy:** WhatsApp → **Einstellungen → Verknüpfte Geräte → Gerät hinzufügen** → QR scannen
2. WhatsApp meldet **"Verknüpft"**, OpenClaw loggt `Listening for personal WhatsApp inbound messages`
3. **Erste DM** an dich selbst (Self-Chat): **"Hallo Coach"** — Bot antwortet sofort

> **Default:** Eigene Nummer ist auto-trusted. Fremde Nummern bekommen einen Pairing-Code — siehe nächster Slide.

DM-Policy: nur du als Sender

Default `pairing` — jede fremde Nummer, die schreibt, bekommt vom Bot einen Pairing-Code. Für Personal-Use unschön: jede versehentliche DM löst eine Bot-Antwort aus.

Vier Modi für `channels.whatsapp.dmPolicy`:

- `pairing` — Default, unbekannte Sender bekommen Code
- `allowlist` — nur `allowFrom`-Nummern werden beantwortet **(Empfehlung)**
- `open` — jeder darf (`allowFrom: ["*"]`)
- `disabled` — alle DMs blockiert

> **WebUI:** Sidebar → **Channels** → **WhatsApp** öffnen → Sektion **Access** → **DM Policy** auf `allowlist` setzen, eigene Nummer in **Allow From** eintragen → **Save**. Bot ignoriert ab dann alle anderen Sender **stumm** — kein Pairing-Code mehr.

Pre-Reply-Reaction: kein Stille-Loch mehr

Problem: nach der Frage erst mal 5–15 Sekunden Stille, bis das LLM antwortet. Fühlt sich tot an.

Lösung: **Acknowledgment Reaction** — der Bot setzt **sofort** nach Empfang ein 👀-Emoji als WhatsApp-Reaction. User weiß: **„gelesen, läuft."**

> **WebUI:** Sidebar → **Channels** → **WhatsApp** → Sektion **Reactions** → **Reaction Level** auf `ack` (oder `minimal`) → **Ack Reaction Emoji** wählen (z. B. 👀, 🤔, 🏋️) → **Save**.

> **Ehrliche Einordnung:** kein echter „is typing..."-Indikator wie bei Discord — die WhatsApp-Web-API gibt das nicht zuverlässig her. Aber das Pre-Reply-Emoji erfüllt **denselben Zweck:** sofortiges Feedback, dass der Bot lebt.

Block-Streaming : Stück für Stück statt am Ende

Default : Bot wartet, bis das LLM komplett fertig ist, schickt dann **einen** großen Text-Block. Bei längeren Antworten gefühlt ewig.

Mit Block-Streaming : Bot schickt Antwort-Blöcke **sukzessive**, sobald sie das LLM produziert hat — erste Bubble nach 1–2 Sekunden, weitere folgen. Fühlt sich an wie ein echter Chat.

> **WebUI :** Sidebar → **Agents** → **Defaults** → **Block Streaming** auf `on` → optional **Human Delay** auf `natural` (800–2500 ms Pause zwischen Blöcken). Plus pro Channel : **Channels** → **WhatsApp** → **Streaming** → **Block Streaming** auf `true`.

> **Bonus :** kombiniert mit der Pre-Reply-Reaction kriegt der User 👀 sofort, dann erste Bubble nach ~2 s, dann weitere. **Das** ist der Discord-Vibe, soweit WhatsApp es zulässt.

Phase 4 : Skills live

Ein Skill ist nur Markdown

Skills leben als `SKILL.md` im Workspace und werden vom Host editiert — der Bind Mount macht's möglich, der Bot zieht sie sich im nächsten Turn nach.

```
mkdir -p workspace/skills/drill-sergeant-mode
$EDITOR workspace/skills/drill-sergeant-mode/SKILL.md
```

Pro Skill ein **Frontmatter-Header** (Metadaten) und ein **Body** (was der Agent dann tatsächlich tut). Ein Beispiel kommt auf der nächsten Slide.

Sport-Coach mit Stimmschalter

```
---
name: drill-sergeant-mode
description: Verschärft den Coach-Modus zum Drill-Instructor — kurze Sätze, keine Ausreden.
---

# Drill Sergeant Mode

Wenn dieser Skill aktiv ist, antworte wie ein Drill Instructor:
- Kurze, knappe Sätze
- 1–2 Imperative pro Antwort
- Gelegentlich GROSSBUCHSTABEN für Akzent
- Keine Ausreden akzeptieren — sarkastisch, aber am Ende motivierend
```

Hot-Loaded: Speichern → `openclaw skills list` zeigt `✓ Ready`. Stil greift im nächsten Turn — kein Restart, kein Trigger.

Phase 5 : Strava

DIY : Strava in 2 Bausteinen

1. **Strava-App registrieren** auf strava.com/settings/api → Callback Domain `localhost` → **Client ID** + **Client Secret** in `~/.openclaw/.env` als `STRAVA_CLIENT_ID` / `STRAVA_CLIENT_SECRET`
2. **Self-Bootstrapping SKILL.md** im Workspace — der Skill **macht den Rest selbst :** erste Frage an den Bot triggert OAuth-Setup, Refresh läuft automatisch, Token-Rotation wird persistiert

> Pädagogisch sauber : **Statische Secrets** in `.env` , **rotierende Tokens** in app-managed JSON.
> Der User berührt OAuth **genau einmal** im ganzen Lebenszyklus.

Verbindung in 30 Sekunden – ein einziges Mal

User: "Verbinde mich mit Strava."

1. **Bot** antwortet mit Authorize-Link:
   "Klick hier: `strava.com/oauth/authorize?client_id=…&scope=activity:read_all` — und schick mir den `code` aus der Redirect-URL zurück."
2. **User** drückt **Authorize** → Strava leitet weiter → `code=XYZ` aus der URL zurück in den Chat
3. **Bot** tauscht den Code gegen Tokens, schreibt `~/.openclaw/strava-tokens.json`, antwortet:
   "Verbunden! Was willst du wissen?"

> Danach **dauerhaft automatisch**: der Skill refresht den Access Token bei Ablauf von selbst und persistiert rotierte Refresh Tokens. Der User sieht OAuth nie wieder.

Skill-Datei: Metadaten

```
# ~/.openclaw/workspace/skills/strava-coach/SKILL.md
---
name: strava-coach
description: Self-bootstrapping Strava-Integration.
metadata: { openclaw: { requires: { bins: ["curl", "jq"] } } }
---
```

Storage: Secrets in Env (`STRAVA_CLIENT_ID/SECRET`), Tokens in `~/.openclaw/strava-tokens.json` — Logik auf der nächsten Slide.

Skill-Datei : Drei Phasen

```
Vor jedem Strava-Call:

(1) BOOTSTRAP — wenn Tokens-Datei fehlt:
    Authorize-URL an User schicken, code aus Redirect zurueckfragen,
    POST /oauth/token mit grant_type=authorization_code, Antwort in Datei.

(2) REFRESH — wenn expires_at - now < 300:
    POST /oauth/token mit grant_type=refresh_token, komplette Antwort
    zurueck in Datei schreiben (refresh_token kann rotieren!).

(3) FETCH:
    GET /api/v3/<endpoint> mit Authorization: Bearer <access_token>.
```

> Drei Phasen, ein Skill — der Rest läuft automatisch. Token-Rotation, Ablauf-Handling, neuer Bootstrap bei Revoke — alles im Markdown beschrieben.

Erste Frage an den Bot

> **Wann hab ich zuletzt trainiert, und wie lief's?**

Live im UI sichtbar — Tool-Calls Schritt für Schritt:

1. Skill matched → curl gegen Strava API
2. JSON zurück → Zusammenfassung mit Datum + Distanz + Pace
3. Im Drill-Stil bewertet, wie's gelaufen ist

> **Magic-Moment:** externe API in zwei Slides durchgeplugt — von Auth bis Antwort.

Phase 6 : Cron

WEBUI

Bot pingt von sich aus

Sidebar → **Cron Jobs** → **+ New :**

- **Name :** `daily-coach`
- **Every :** `1` Minute **(Demo-Tempo)**
- **Assistant task prompt :** **"Schau auf Marcels Strava-Daten. Wenn trainiert: knapp loben. Wenn nicht: drillen."**
- **Channel :** WhatsApp · **Recipient :** `+49…`

→ **Add job**. Agent triggert sich selbst, ruft den Strava-Skill, pusht via WhatsApp. **Kein User-Input.**

> **Demo :** `1 Minute` für Live-Effekt. Produktion : `24h` oder Cron `0 7 * * *`.

Phase 7 : Web Search

Bot kann nur, was er weiß

Aktuelle Events, News, Wetter, Termine — alles **Live-Daten** außerhalb des LLM-Trainings. Lösung: Tavily Search API.

- **API-Key statt OAuth** → 30 Sekunden Setup, kein Refresh-Tanz
- **POST /search** → strukturiertes JSON mit `answer` (schon LLM-zusammengefasst) + `results`
- **1000 Searches / Monat free**, 1 Credit pro basic-Query
- Bonus: Country-Filter, Date-Range, Domain-Allowlist — alles im Body

> Selbes Skill-Pattern wie Strava. Nur die Auth-Kategorie wechselt: **static API-Key** statt **rotating OAuth**.

API-Key in 2 Minuten

1. **tavily.com** → **Sign in** (Google / GitHub / Email — **kein Kredit-Karten-Theater**)
2. **Dashboard → API Keys → Generate New Key**
3. Key kopieren — Format `tvly-…`
4. Im Container in `~/.openclaw/.env` setzen:

```
TAVILY_API_KEY=tvly-…
```

5. Container neu starten → Skill erkennt den Key automatisch via `primaryEnv`

> **Free Tier:** 1000 Credits / Monat. Basic-Search kostet 1 Credit. Für Personal-Use und Demo reicht das locker.

Web Search in einem 8-Zeilen-Skill

```
# ~/.openclaw/workspace/skills/websearch/SKILL.md
---
name: websearch
description: Sucht aktuelle Web-Infos via Tavily Search API.
metadata:
  openclaw:
    requires: { bins: ["curl", "jq"] }
    primaryEnv: TAVILY_API_KEY
---

curl -s -X POST https://api.tavily.com/search \
  -H "Authorization: Bearer $TAVILY_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"query":"<frage>","search_depth":"basic","include_answer":true,"max_results":5,"country":"germany"}'
```

> Antwort liefert `answer` (kuratiert) + `results[]`. Kein Parsing — direkt nutzbar.

Phase 8 : Komposition

Frage: Welche Lauf-Events nächste Woche?

Der Agent orchestriert **drei** Skills, ohne dass wir was Neues programmiert haben:

1. `USER.md` → Wohnort Frankfurt, Sportart-Präferenz, Distanz-Range
2. `strava-coach` → letzte **Outdoor**-Aktivitäten (Zwift/virtual gefiltert) → typische Distanzen + Tempo
3. `websearch` → Tavily-Query **"Laufveranstaltungen Frankfurt KW 18 5-10km"**

→ Bot fasst Top-3 Events zusammen mit Datum, Distanz, Anmelde-Link.

> **Skills sind komponierbar.** Die Logik dahinter ist nicht Code — es ist der Agent, der die Markdown-Anweisungen seiner Skills zusammensetzt.

USER.md erweitern

```
# USER.md – Auszug

## Sport-Profil

- **Wohnort**: Frankfurt am Main
- **Outdoor**: Rennrad / Laufen
- **Indoor**: Zwift (virtuelle Workouts bei lokalen Empfehlungen ignorieren!)
- **Distanzen**: 5–10 km Laufen / Radfahren < 150 km
- **Interesse**: lokale Radrennen / Laufevents, Events im Umkreis 50 km
```

> `USER.md` ist **Living Context**. Versionierbar wie jeder andere Markdown-File. Editiere am Host, Bot greift es im nächsten Turn auf — exakt wie bei Skills.

TOOLS.md: dein lokales Setup als Notizblock

```
## Strava
- Bei lokalen Empfehlungen `VirtualRide` ignorieren (Zwift)

## Frankfurter Outdoor-Strecken
- Mainufer-Loop — 8 km, flach, asphaltiert. Tempo-Läufe.
- Niddapark-Runde — 10 km, leicht hügelig, Trail/Asphalt.
- Taunushänge — 25-60 km Rad, Anstiege Königstein/Falkenstein.

## Tavily
- Event-Anfragen: bevorzugt laufkalender.de, eventbrite.de
```

> **Skills sind shareable. Dein Setup ist deins.** TOOLS.md ist eine von sechs Bootstrap-Files im Workspace (mit `AGENTS.md`, `SOUL.md`, `USER.md`, `IDENTITY.md`, `HEARTBEAT.md`). Doku-Wortlaut: „Notes about your local tools and conventions. Does not control tool availability; it is only guidance."

Eine Regel : zwei Tool-Calls

```
## Event-Matching-Regeln

Bei Fragen nach Events / Rennen / Wettkämpfen — immer in dieser Reihenfolge:

1. Aktuelle Form aus Strava (letzte 4 Wochen, **nur Outdoor** — Zwift ignorieren)
2. Typische Distanzen + Schnitt-Pace ableiten
3. Events suchen, die dazu passen (±20 % Distanz, gleiche Sportart, Umkreis 50 km)
4. Vorschlag mit Begründung — *"passt, weil deine letzten Läufe alle bei 6-8 km lagen"*
```

→ **Eine Markdown-Regel** zwingt den Agent, `strava-coach` **vor** `websearch` zu callen.

> **Der Impact:** Komposition ist nicht hardcoded — sie entsteht aus Kontext. Wenige Zeilen in `USER.md` ändern das Tool-Verhalten dauerhaft, ohne Skill-Code anzufassen. **Das** ist Markdown-as-Programming.

Phase 9 : ClawHub

CLAWHUB

Statt selbst frickeln

Was wir gerade per DIY in Phase 5 gebaut haben — das gibt's auch fertig:

```
openclaw skills search strava
openclaw skills install openclaw-strava
openclaw skills install strava-training-coach    # Coach-Logik on top
```

Was inkludiert ist: OAuth-Setup-Wizard, Token-Refresh, Pagination, Rate-Limits. Skill landet in `~/.openclaw/workspace/skills/openclaw-strava/`.

> **ClawHub** ist die offene Skill-Registry. Veröffentlichen = `git push` + PR. Gleiches Modell wie npm oder Homebrew — nur für KI-Skills.

Phase 10 : Persistenz

DOCKER

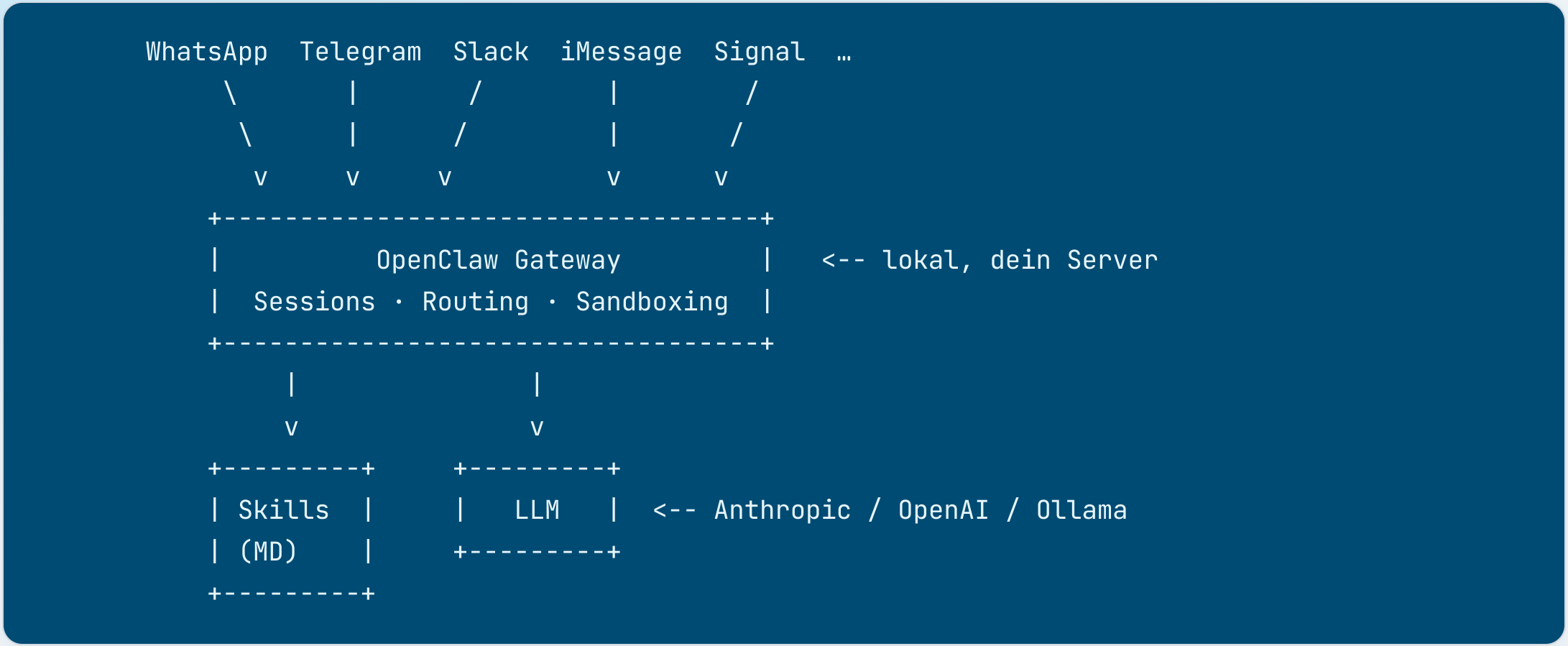
Container weg, Daten bleiben

```
exit
docker compose down              # Container wird zerstört
docker compose up -d             # neuer Container
docker compose exec openclaw bash
ls ~/.openclaw/                  # Konfiguration noch da
```

- `~/.openclaw/` lebt im Named Volume `openclaw-home`
- Skills + Workspace liegen als Bind Mount auf dem Host
- VPS dort hält **systemd-User-Service** den Gateway am Leben

> Genau dieselbe Resilienz, die du für ein produktives Setup auf deinem VPS brauchst.

Wie alles zusammenhängt



Channels sind austauschbar, Skills versionierbar, das LLM frei wählbar.

Was du für Produktion wissen solltest

- **Sandbox-Modus** für fremde Sessions :

```
{ agents: { defaults: { sandbox: { mode: "non-main" } } } }
```

- **Allowlist** auf eigene Telefonnummern, User- oder Server-IDs (je nach Channel) — sonst spricht jede:r mit dem Bot
- Tokens **nie** ins Git-Repo — sie liegen in `~/.openclaw/openclaw.json` (Container-Volume)
- OpenClaw als **nicht-root User** laufen lassen (`systemd --user`)
- Remote-Zugriff über Reverse-Proxy oder Tailscale, nie direkt ins Internet

Häufig gestellte Fragen

- **„Geht das auch ohne Cloud-LLM?"** : ja — Ollama lokal, Modell in `openclaw.json`.
- **„Mehrere Channels gleichzeitig?"** : ja — jeder Channel kann auf einen eigenen Agent geroutet werden.
- **„Was kostet das?"** : Software MIT-lizenziert — Kosten nur für LLM-Tokens.
- **„Wie verwalten wir Skills im Team?"** : als Git-Repo, ClawHub als Registry.
- **„Kann byte5 das für uns aufsetzen?"** : ja — sprich mich gerne im Anschluss an.

Vielen Dank.

MATERIAL ZUM MITNEHMEN

- **OpenClaw :** github.com/openclaw/openclaw
- **Docs :** docs.openclaw.ai
- **Skills-Registry :** clawhub.ai
- **byte5 :** byte5.de

DU SUCHST UNTERSTÜTZUNG BEI DEINEM DIGITALEN PROJEKT?

Dein digitales Projekt : unsere Expert:innen beraten dich transparent.