Lokale KI & Homelab für Einsteiger

Dein eigener KI-Server zu Hause

Untertitel: ChatGPT, Bilder-KI und Selfhosting komplett auf deiner eigenen Hardware — ohne Cloud-Abo, ohne Datenverkauf, ohne monatliche Kosten.

Verkaufsversprechen: Nach diesem Buch läuft auf deinem eigenen Server eine vollwertige KI-Umgebung: ein Chat wie ChatGPT, eine Bildgenerierung wie Midjourney und ein Web-Frontend, das beides bündelt — alles offline, alles unter deiner Kontrolle. Du lernst nicht nur, die passende Hardware günstig zusammenzustellen, sondern auch, wie du deinen Server absicherst, sicherst und bei Bedarf erweiterst. Kein Vorwissen nötig: Jeder Schritt ist konkret beschrieben, mit Befehlen zum Kopieren und den typischen Fehlern, die dich sonst Stunden kosten.

Einleitung: Warum dein eigener KI-Server — und warum jetzt

Stell dir vor, du tippst eine Frage in ein Chatfenster, und die Antwort kommt — ohne Internet, ohne Abo, ohne dass deine Daten einen fremden Server verlassen. Stell dir vor, du beschreibst ein Bild in zwei Sätzen, und wenige Sekunden später erscheint es auf deinem eigenen Bildschirm, erzeugt von der Grafikkarte, die in deinem Arbeitszimmer brummt. Genau darum geht es in diesem Buch: um einen eigenen KI-Server, der bei dir zu Hause steht und dir gehört.

Die großen KI-Dienste haben in den letzten Jahren eine Tür aufgestoßen. ChatGPT beantwortet Fragen, Midjourney malt Bilder, und für all das zahlen Millionen Menschen Monat für Monat — oder geben ihre Daten her, was oft auf dasselbe hinausläuft. Was kaum jemand weiß: Die Technologie dahinter ist keine Magie, die nur in den Rechenzentren großer Konzerne existiert. Sie ist Software. Und Software kannst du auf einem normalen Computer ausführen.

Das ist der Kern dieses Buches. Du brauchst keinen Abschluss in Informatik und kein dickes Portemonnaie. Du brauchst einen gebrauchten Server oder eine ausgediente Workstation, etwas Zeit und die Bereitschaft, ein paar Dinge selbst in die Hand zu nehmen. Was du dafür bekommst,

ist bemerkenswert: eine KI-Umgebung, die dir niemand wegnehmen kann, die keine Nutzungsbedingungen diktiert und die keine monatlichen Kosten verursacht — einmal angeschafft, gehört sie dir.

Dieses Buch ist für Menschen geschrieben, die Technik verstehen wollen, statt sie nur zu benutzen. Wenn du noch nie einen Server aufgesetzt hast, bist du hier richtig — wir fangen bei null an und erklären jeden Schritt. Wenn du schon ein Homelab betreibst, findest du trotzdem genug Konkretes: genaue Hardware-Empfehlungen, getestete Befehle und die Fallstricke, die dich sonst Stunden kosten. Wir kürzen nichts ab, aber wir blähen auch nichts auf. Jedes Kapitel bringt dich einen Schritt weiter zu einem Ziel, das du anfassen kannst: einem laufenden System.

Der Weg ist klar gegliedert. Zuerst klären wir, warum sich das Ganze lohnt und was deine Hardware können muss. Dann bauen wir das Fundament — das Betriebssystem Proxmox, auf dem später alles läuft. Darauf folgen die beiden großen Anwendungsfälle: Sprachmodelle für den Chat und Bildgenerierung mit ComfyUI. Zum Schluss machen wir aus deinem Experiment ein ernsthaftes Projekt: sicher, energiesparend und mit Backups, die im Ernstfall funktionieren.

Ein ehrlicher Hinweis vorweg: Dein eigener KI-Server wird nicht in jeder Disziplin mit den Cloud-Riesen mithalten. Ein gebrauchter Server für ein paar hundert Euro liefert keine Antworten so schnell wie ein Rechenzentrum mit tausenden Grafikkarten. Aber er liefert Antworten, die privat bleiben. Er arbeitet weiter, wenn das Internet ausfällt. Und er gehört dir — für immer, ohne Monatsbeitrag. Für viele Menschen, die dieses Buch lesen, ist genau dieser Tausch der richtige.

Wir werden nicht lange theoretisieren. Schon im zweiten Kapitel stehen konkrete Kaufempfehlungen, ab dem dritten installierst du. Am Ende hast du nicht nur gelesen, wie es geht — du hast es getan. Und wenn du einmal gesehen hast, wie dein eigener Server deine erste Frage beantwortet, wirst du verstehen, warum das mehr ist als ein Bastelprojekt: Es ist ein Stück Unabhängigkeit.

Leg los. Dein Server wartet schon.

Kapitel 1: Warum ein eigener KI-Server?

Stell dir vor, du tippst eine Frage ein, und die Antwort kommt nicht aus irgendeinem Rechenzentrum am anderen Ende der Welt, sondern aus dem Kasten, der neben deinem Schreibtisch brummt. Kein Konto, kein Login, kein "Ihre Anfrage ist in Bearbeitung". Genau das ist ein eigener KI-Server: ein normaler Rechner, auf dem du große Sprachmodelle (LLMs) wie Llama 3, DeepSeek-R1 oder Qwen lokal betreibst – über Ollama, mit einer Oberfläche wie OpenWebUI, und bei Bedarf mit Bildgenerierung über ComfyUI. Warum sollte man das tun, wenn ChatGPT

und Co. doch nur ein paar Euro im Monat kosten? Die Antwort ist nicht "weil man es kann", sondern eine Mischung aus vier handfesten Gründen – und ein paar ehrlichen Nachteilen, die du kennen solltest, bevor du Geld ausgibst.

Datenschutz: deine Daten bleiben deine

Das stärkste Argument. Wenn du einen Cloud-Dienst nutzt, verlässt jede einzelne Anfrage dein Gerät. Deine privaten Texte, geschäftlichen Unterlagen, Gesundheitsfragen, Finanzdaten – alles wandert über das Internet zu einem Anbieter, dessen Server irgendwo stehen und dessen Nutzungsbedingungen sich jederzeit ändern können. Viele Anbieter werten Eingaben aus, trainieren darauf oder speichern sie zumindest eine Weile. Das musst du ihnen glauben, kontrollieren kannst du es nicht.

Ein lokaler Server dreht das um: Die Anfrage geht von deinem Rechner ins Modell und wieder zurück – und verlässt dabei nie deine vier Wände. Es gibt keinen Dritten, dem du vertrauen müsstest, keine Datenbank mit deinen Gesprächsverläufen in fremder Hand, keine Klausel, die sich über Nacht ändert. Für alles, was vertraulich ist – Patientendaten, Verträge, Familieninternes – ist das kein Nice-to-have, sondern oft die einzige saubere Lösung.

Kosten: einmal zahlen statt ewig abonnieren

Die Rechnung ist simpel. Ein Cloud-Abo kostet laufend: 20 € hier, 25 € dort, schnell sind es mehrere hundert Euro im Jahr – und du besitzt am Ende nichts. Ein eigener Server kostet einmal Geld für Hardware und danach nur noch Strom. Bei einem Strompreis von 0,40 €/kWh und einem typischen Setup, das beim Antworten kurzzeitig 250 bis 400 Watt zieht und sonst im Leerlauf döst, reden wir realistisch von wenigen Cent pro Stunde aktiver Nutzung und ein paar Euro pro Monat im Dauerbetrieb.

Das Schöne: Die Hardware gehört dir. Keine Preiserhöhung, kein "wir stellen den Tarif um", kein Konto, das gesperrt wird. Und der Server kann nebenbei noch viel mehr, als nur zu chatten – dazu später. Fairerweise muss man sagen: Ein gutes Cloud-Modell ist für Gelegenheitsnutzer oft trotzdem billiger als eine 1.500-€-Maschine. Die Rechnung kippt, wenn du viel nutzt, private Daten hast oder die Maschine ohnehin für andere Dinge gebrauchen kannst.

Kontrolle: du bestimmst, was läuft

Ein Cloud-Dienst ist eine Blackbox. Das Modell ändert sich, wenn der Anbieter es ändert – manchmal wird es "verbessert" und liefert plötzlich andere Antworten, manchmal verschwindet es ganz. Bei deinem eigenen Server entscheidest du: welches Modell, welche Version, welche

Größe, welche System-Prompts, welche Oberfläche. Du kannst ein winziges, schnelles Modell für den Alltag fahren und ein großes für anspruchsvolle Aufgaben. Du kannst Modelle mischen, austauschen, ausprobieren – ohne jemanden zu fragen.

Offline: KI auch ohne Internet

Der unterschätzte Vorteil. Dein Server läuft im lokalen Netzwerk – und das funktioniert auch dann, wenn der Internetanschluss tot ist. Kein Ausfall des Anbieters, keine Wartungsfenster, keine Region, in der ein Dienst nicht verfügbar ist. Für ein Homelab auf dem Land, für den Camper oder einfach für den Fall, dass du nicht von fremden Servern abhängig sein willst: ein handfester Grund.

Cloud-KI vs. lokal im direkten Vergleich

Cloud-Modelle wie GPT-4 oder Claude sind heute schlicht stärker als fast alles, was auf Verbraucher-Hardware läuft. Sie beantworten schwierige Fragen besser, beherrschen mehr Sprachen sauber, und sie laufen auf Geräten ohne GPU – sogar im Browser. Dafür zahlst du mit Geld, Daten und Abhängigkeit.

Lokale Modelle haben in den letzten zwei Jahren enorm aufgeholt. Ein Llama 3 mit 8 Milliarden Parametern, ein DeepSeek-R1-Distillat oder ein Qwen-Modell liefern auf einer halbwegs modernen Grafikkarte erstaunlich gute Antworten – und zwar sofort, privat und kostenlos pro Anfrage. Für Zusammenfassungen, Textarbeit, Programmieren, Übersetzungen und kreatives Schreiben reichen sie oft völlig. Wo sie schwächeln, sind hochkomplexe Schlussfolgerungen, exotische Fachgebiete und die feinsten Nuancen. Für solche Fälle kannst du aber immer noch gelegentlich einen Cloud-Dienst zuschalten – lokal und Cloud schließen sich nicht aus.

Die ehrlichen Nachteile

Ein eigener KI-Server ist kein Selbstläufer. Erstens: die Einstiegshürde. Du brauchst Grundverständnis für Hardware, musst Proxmox VE oder ein direktes Linux-Setup einrichten, Container starten, mit der Kommandozeile umgehen. Das ist machbar – dieses Buch führt dich Schritt für Schritt durch – aber es ist Arbeit, kein Fertigprodukt.

Zweitens: die Qualitätslücke. Wie gesagt, die Spitzenmodelle aus der Cloud sind besser. Wenn du beruflich auf maximale Qualität angewiesen bist, wird dich lokal nicht alles glücklich machen.

Drittens: Hardwarekosten und Strom. Eine brauchbare GPU kostet ein paar hundert Euro, eine richtig gute über tausend. Der Server braucht Platz, macht Geräusche und läuft rund um die Uhr, wenn du ihn immer verfügbar haben willst.

Viertens: Wartung. Updates, Backups, Fehlersuche – das machst du selbst. Niemand übernimmt das für dich.

Wenn du nach dem Lesen dieser Liste denkst "das klingt machbar", dann bist du hier richtig. Der Rest des Buches macht dich Schritt für Schritt dazu in der Lage.

Kapitel 2: Hardware-Auswahl – das richtige Fundament

Bevor du auch nur eine Zeile installierst, steht eine Entscheidung an, die alles Weitere prägt: Auf welcher Hardware soll dein KI-Server laufen? Die gute Nachricht: Du musst dafür weder ein IT-Profi sein noch ein Vermögen ausgeben. Die schlechte: An der falschen Stelle zu sparen rächt sich schnell – und das teuerste Bauteil ist nicht automatisch das richtige.

Das Herzstück: die GPU und ihr VRAM

Für lokale KI zählt ein Wert über fast allem: der Videospeicher deiner Grafikkarte, kurz VRAM. Große Sprachmodelle müssen komplett in diesen Speicher passen, damit sie flüssig laufen. Die Faustregel: Ein Modell mit 7 bis 8 Milliarden Parametern braucht bei 4-Bit-Quantisierung (einer gängigen Komprimierung, die kaum Qualität kostet) ungefähr 4 bis 5 GB VRAM. Ein Modell mit 13 bis 14 Milliarden Parametern braucht etwa 8 bis 10 GB, ein 30-Milliarden-Modell rund 20 GB und ein 70-Milliarden-Modell rund 40 GB.

Daraus ergibt sich die wichtigste Kaufregel: **Mehr VRAM schlägt mehr Rechenleistung.** Eine ältere Karte mit 24 GB VRAM kann ein größeres, klügeres Modell laden als eine flotte neue Karte mit nur 8 GB – auch wenn sie pro Token etwas langsamer ist. Für Einsteiger sind zwei Beispiele prägend: Die gebrauchte GTX 1080 Ti bringt 11 GB VRAM für oft 150 bis 250 € und ist ein legendäres Preis-Leistungs-Wunder, allerdings älter und ohne moderne KI-Beschleunigung. Die RX 6900 XT mit 16 GB ist die AMD-Alternative, leistungsfähiger, aber unter Linux mit etwas mehr Einrichtungsaufwand verbunden, da AMDs ROCm-Treiber-Ökosystem nicht so reibungslos läuft wie Nvidias CUDA.

Für den Start gilt: 8 GB VRAM sind das absolute Minimum, 12 bis 16 GB sind der Sweet Spot für Einsteiger, 24 GB und mehr öffnen die Tür zu richtig großen Modellen.

CPU und RAM: die unterschätzten Mitspieler

Die CPU ist beim reinen Textgenerieren zweitrangig – das Modell läuft auf der GPU. Aber sie übernimmt alles drumherum: Container verwalten, den Webserver bedienen, Eingaben vorverarbeiten, Bildgenerierung anstoßen. Ein moderner 6- bis 8-Kerner reicht für den Anfang locker, mehr Kerne schaden nicht, sind aber selten der Flaschenhals.

Beim Arbeitsspeicher gilt eine einfache Regel: mehr als du denkst. 16 GB sind die Untergrenze, 32 GB komfortabel, 64 GB dann sinnvoll, wenn du mehrere Modelle parallel oder nebenbei noch andere Dienste wie eine Mediathek oder ein NAS betreiben willst. RAM ist billig – hier zu sparen ist ein klassischer Anfängerfehler, der sich später in ruckelnden Containern und Speicherfehlern rächt.

Drei Bauformen: Desktop, Mini-PC oder gebrauchter Server?

Der Desktop-PC ist der Klassiker. Ein normaler Tower mit einer großen Grafikkarte ist am flexibelsten: Du kannst die GPU später tauschen, hast Platz für mehrere Festplatten und Lüfter, und die Kühlung ist gut beherrschbar. Nachteil: Er ist groß, laut und verbraucht im Leerlauf oft 60 bis 100 Watt. Ideal, wenn du Wert auf Aufrüstbarkeit legst und ein separates Zimmer oder einen Kellerraum hast.

Der Mini-PC (Intel NUC und vergleichbare, oft 300 bis 600 €) ist winzig, leise und stromsparend – typischerweise 10 bis 30 Watt im Leerlauf. Aber: Er hat keine starke Grafikkarte, bestenfalls eine kleine integrierte GPU. Für große Modelle taugt er damit kaum. Sein Platz ist ein anderer: Er eignet sich hervorragend als schlanker Dauerläufer für kleine Modelle, als Proxmox-Host für Dienste wie OpenWebUI oder als Zuspieler für eine separate GPU-Maschine.

Der gebrauchte Server (ausgemusterte Enterprise-Hardware, oft 200 bis 600 €) verführt mit viel RAM, vielen Kernen und professioneller Technik. Doch Vorsicht: Diese Maschinen sind laut, stromhungrig – teils 150 Watt und mehr im Leerlauf – und oft ohne passende Grafikleistung, weil Server selten starke GPUs haben. Bei 0,40 €/kWh frisst so ein Gerät schnell 30 bis 50 € Strom im Monat, wenn es rund um die Uhr läuft. Für reine KI ist er meist die falsche Wahl; für ein komplettes Homelab mit vielen Diensten kann er sich lohnen.

Konkrete Preisbereiche für den Einstieg

Ein realistisches Budget sieht in etwa so aus:

- **Bis 300 €:** Gebrauchter Desktop mit einer GTX 1080 Ti (11 GB) oder ähnlich. Läuft, um Llama-3-8B und vergleichbare Modelle flüssig zu fahren – der ideale, billige Test.
- **500 bis 900 €:** Gebrauchter oder selbst zusammengestellter Rechner mit 16 GB VRAM (z. B. RX 6900 XT oder RTX-Karten der 3000er-Serie). Damit laufen 13- bis 14-Milliarden-Modelle bequem, und du hast Reserven.

- **1.000 bis 2.000 €:** Solide neue oder fast neue Hardware mit 24 GB VRAM (z. B. eine gebrauchte RTX 3090). Hier beginnt der Bereich, in dem du 30- bis 70-Milliarden-Modelle ernsthaft nutzen kannst.

Die Empfehlung für Einsteiger

Starte nicht mit dem teuersten Gerät. Der beste erste Schritt ist ein gebrauchter Desktop-PC mit 12 bis 16 GB VRAM – genug, um Ollama, OpenWebUI und ein paar Modelle richtig kennenzulernen, günstig genug, dass ein Fehlkauf nicht wehtut. Achte auf drei Dinge: eine Nvidia-Karte (CUDA erspart dir das meiste Treiber-Elend), mindestens 32 GB RAM und eine SSD. Installiere darauf Proxmox VE als Fundament – dann kannst du jeden Dienst in seinem eigenen Container betreiben, sauber getrennt und leicht zu sichern. Wenn die Begeisterung wächst und du merkst, wo dich deine Maschine limitiert, kaufst du gezielt nach – mehr VRAM, mehr RAM, oder beides.

Denk immer an die Stromrechnung: Bei 0,40 €/kWh kostet ein Server, der im Schnitt 100 Watt zieht, rund 29 € im Monat im Dauerbetrieb. Eine Maschine mit 300 Watt kommt auf etwa 86 €. Wer den Server nur bei Bedarf hochfährt, spart einen großen Teil davon – ein einfacher Schalter für den Geldbeutel.

Kapitel 3: Proxmox VE — das Betriebssystem deines Servers

Was ist Proxmox VE?

Proxmox Virtual Environment (kurz Proxmox VE oder PVE) ist die Grundlage deines Homelabs. Es ist ein eigenständiges Betriebssystem, das du direkt auf einen Server oder einen alten PC installierst. Anders als Windows oder ein normales Linux-Desktop ist Proxmox eine Virtualisierungsplattform: Darauf laufen nicht direkt deine Programme, sondern viele kleine, voneinander getrennte Umgebungen — Container und virtuelle Maschinen.

Der große Vorteil: Du kannst auf einer einzigen Kiste gleichzeitig einen Dateiserver, eine Firewall, ein Heimautomatisierungs-System und später auch Ollama samt OpenWebUI betreiben. Jede dieser Umgebungen ist sauber von den anderen getrennt. Wenn ein Experiment kaputtgeht, löschst du einfach den Container und startest einen neuen — der Rest des Servers läuft unberührt weiter.

Technisch basiert Proxmox auf Debian Linux und nutzt zwei Virtualisierungstechniken unter einer Haube: KVM für vollständige virtuelle Maschinen und LXC für leichtgewichtige Container. Beides verwaltest du über ein übersichtliches Web-Interface, das du vom Browser aus erreichst. Du musst also nicht dauerhaft am Server sitzen — nach der Installation läuft alles bequem vom Schreibtisch-Rechner oder Laptop aus.

Proxmox ist für den privaten Gebrauch kostenlos, wird von einer aktiven Community gepflegt und ist seit vielen Jahren in Produktion erprobt. Für den Einstieg ins Homelab ist es der mit Abstand beste Startpunkt: gut dokumentiert, stabil und mit einer riesigen Menge an Anleitungen im Netz.

Installation

Die Installation dauert etwa 20 Minuten und ist auch für Anfänger machbar. Du brauchst dafür:

- einen Rechner mit mindestens 4 GB RAM (besser 8 GB oder mehr, wenn später LLMs laufen sollen),
- einen USB-Stick mit mindestens 2 GB,
- Zugriff auf Tastatur, Monitor und Netzwerk am Server.

So gehst du vor:

1. Lade das ISO-Image von der offiziellen Seite herunter: <https://www.proxmox.com/downloads>. Wähle die aktuelle stabile Version („Proxmox VE ... ISO Installer").

2. Erstelle einen bootfähigen USB-Stick. Unter Windows eignet sich dafür das kostenlose Tool **Rufus**. Wähle das heruntergeladene ISO aus und schreibe es im „DD"-Modus auf den Stick.

3. Stecke den Stick in den Server, starte ihn und boote vom USB-Stick. Meist musst du dafür beim Start eine Taste wie F11, F12 oder Entf drücken, um ins Bootmenü zu kommen.

4. Folge dem Installationsassistenten. Wichtig sind zwei Entscheidungen:

- **Ziel-Festplatte**: Hier wird das komplette Laufwerk für Proxmox verwendet. Alle vorhandenen Daten darauf werden gelöscht.

- **Netzwerkkonfiguration**: Vergebe eine feste IP-Adresse aus deinem Heimnetz, zum Beispiel `192.168.1.100`, dazu die passende Netzmaske (`255.255.255.0` bzw. `/24`) und das Gateway (meist deine Router-IP, etwa `192.168.1.1`). Eine feste IP ist wichtig, weil du den Server später immer unter dieser Adresse erreichst.

5. Nach dem Neustart zeigt dir der Server eine Übersicht mit einer Adresse wie `https://192.168.1.100:8006`. Das ist dein Tor zur gesamten Verwaltung.

Öffne diese Adresse im Browser deines Arbeitsrechners. Dein Browser wird wegen des selbstsignierten Zertifikats eine Warnung anzeigen — bestätige sie einmalig als Ausnahme. Der Login ist standardmäßig `root` mit dem Passwort, das du während der Installation gewählt hast.

LXC-Container vs. virtuelle Maschine

In Proxmox hast du die Wahl zwischen zwei Arten von Umgebungen, und diese Entscheidung solltest du bewusst treffen.

LXC-Container sind leichtgewichtige, Linux-basierte Umgebungen. Sie teilen sich den Kernel des Hosts, starten in Sekunden und verbrauchen kaum Ressourcen. Für die meisten Serverdienste — und auch für Ollama — sind Container die richtige Wahl. Ein Container mit einem Dienst braucht oft nur 50 bis 200 MB RAM im Leerlauf.

Virtuelle Maschinen (VMs) emulieren dagegen einen kompletten Rechner inklusive eigenem Kernel. Du kannst darin auch Windows oder ein anderes Betriebssystem installieren. VMs sind besser isoliert und flexibler, kosten dafür aber deutlich mehr Ressourcen, weil jede VM ihren eigenen Speicher- und Rechen-Overhead hat.

Als Faustregel für Einsteiger: **Immer zuerst an Container denken.** Nur wenn du ein fremdes Betriebssystem brauchst, eine besonders starke Isolation willst oder Software einsetzt, die auf einem echten Kernel besteht, nimmst du eine VM. Für unser Ziel — Ollama und OpenWebUI betreiben — ist ein LXC-Container ideal.

Netzwerk-Bridge

Damit deine Container ins Netzwerk kommen, brauchst du eine Netzwerk-Bridge. Proxmox legt bei der Installation automatisch eine Bridge namens `vmbr0` an, die mit deiner physischen Netzwerkkarte verbunden ist.

Du kannst sie unter **System** → **Netzwerk** im Web-Interface kontrollieren. Dort siehst du den Eintrag `vmbr0` mit deiner festen IP-Adresse. Wenn du einen neuen Container anlegst, wählst du als Netzwerkgerät einfach `vmbr0` und vergibst dem Container eine eigene IP aus deinem Netz. So ist jede Umgebung aus deinem Heimnetz erreichbar, als wäre sie ein eigener kleiner Rechner.

Für fortgeschrittene Setups gibt es zusätzlich interne Bridges, die nur für die Kommunikation der Container untereinander gedacht sind — etwa wenn ein Dienst nicht direkt aus dem Internet erreichbar sein soll. Für den Anfang genügt aber die eine Bridge `vmbr0` völlig.

Erste Schritte und das Web-Interface

Das Web-Interface ist in drei Bereiche gegliedert. Links die **Ressourcen-Übersicht**, in der du deinen Server („Datacenter"), den Knoten und später alle Container und VMs siehst. In der Mitte die Details und rechts oben die Aktionen.

Lege als ersten Test einen Container an:

1. Klicke oben rechts auf **„CT anlegen"** (CT steht für Container).

2. Wähle eine **Vorlage** (Template). Beim ersten Mal musst du Templates herunterladen: Gehe im Web-Interface zu deinem Knoten, dann **„lokaler Speicher"** → **„CT Templates"** → **„Templates"** und lade dort ein aktuelles Debian- oder Ubuntu-Template herunter. Danach steht es dir bei der Container-Erstellung zur Auswahl.

3. Vergib eine **Container-ID**, einen Hostnamen und ein **Root-Passwort**.

4. Wähle bei **„Vorlage"** dein heruntergeladenes Debian-Template.

5. Setze **Festplatte** (8 GB reichen zum Testen) und **CPU/RAM** (1 Kern, 512 MB sind für den Anfang genug).

6. Unter **„Netzwerk"** wählst du `vmbr0` und vergibst eine freie IP aus deinem Netz.

7. Bestätige mit **„Fertigstellen"**.

Der Container erscheint links in der Liste. Markiere ihn, klicke **„Start"** und dann **„Konsole"**. Du landest in einer Textkonsole, in der du dich mit `root` und deinem Passwort anmeldest. Damit hast du bewiesen, dass dein Server funktioniert: Du hast gerade deine erste eigene Linux-Umgebung erstellt und gestartet.

Mit diesen Grundlagen bist du gerüstet. Der nächste Schritt ist, in so einem Container Ollama zu installieren — das ist das Thema des folgenden Kapitels.

Kapitel 4: Ollama — lokale LLMs zum Ausprobieren

Was ist Ollama?

Ollama ist ein Werkzeug, mit dem du große Sprachmodelle (LLMs) direkt auf deiner eigenen Hardware ausführst — ohne Cloud, ohne Abo und ohne dass deine Eingaben an einen externen

Server geschickt werden. Es bündelt die komplexe Technik hinter Modellen wie Llama oder Mistral in einfachen Befehlen, die du in der Kommandozeile ausführst.

Der Kern von Ollama ist ein kleiner Hintergrunddienst, der auf deinem Server läuft und die Modelle verwaltet. Du sprichst ihn über die Befehlszeile an — oder später bequem über eine grafische Oberfläche wie OpenWebUI. Ollama übernimmt dabei automatisch Aufgaben, die sonst Handarbeit wären: das Herunterladen der Modellgewichte, die Auswahl der passenden Laufzeit und die Verwaltung des Grafikkartenspeichers.

Für dein Homelab ist Ollama die ideale Einstiegsdroge in das Thema lokale KI: Du installierst es in wenigen Minuten, lädtst ein Modell mit einem einzigen Befehl und kannst sofort chatten. Wenn dir ein Modell nicht gefällt, wechselst du einfach zu einem anderen.

Installation

Ollama installierst du am besten in dem Debian-Container, den du im vorigen Kapitel angelegt hast. Melde dich in der Konsole des Containers an und führe diesen Befehl aus:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Das Skript lädt die passende Version herunter, installiert sie und startet den Hintergrunddienst automatisch. Ob alles funktioniert, prüfst du mit:

```
ollama --version
```

Sollte der Befehl nicht gefunden werden, hat die Installation nicht gegriffen — dann schau, ob `curl` installiert ist (`apt install curl`). Wichtig: Ollama läuft auch ohne Grafikkarte. Ohne GPU rechnet es nur auf der CPU, was bei kleinen Modellen durchaus brauchbar ist, bei großen Modellen aber langsam wird.

Damit Ollama auch von OpenWebUI erreichbar ist, muss der Dienst auf dem Netzwerk lauschen. Standardmäßig bindet er sich nur an die lokale Schnittstelle. Setze deshalb die Umgebungsvariable:

```
OLLAMA_HOST=0.0.0.0
```

Am einfachsten startest du den Dienst danach mit dieser Variable manuell oder trägst sie dauerhaft in die Systemd-Konfiguration ein. Für erste Tests genügt es aber, Ollama zunächst lokal im Container auszuprobieren.

Modelle pullen

Ein Modell lädst du mit `ollama pull` herunter, gefolgt vom Modellnamen. Die populärsten Modelle findest du in der Modellbibliothek unter <https://ollama.com/library>. Für den Einstieg sind diese drei gut geeignet:

```
ollama pull llama3.2:3b
ollama pull mistral
ollama pull gemma2:2b
```

`llama3.2:3b` ist ein kleines, schnelles Modell von Meta, `mistral` ein solider Allrounder und `gemma2:2b` ein sehr leichtgewichtiges Modell von Google. Mit `ollama list` siehst du jederzeit, welche Modelle bereits heruntergeladen sind und wie viel Speicherplatz sie belegen.

Die Bezeichnung hinter dem Doppelpunkt gibt die Größenklasse an: `3b` bedeutet 3 Milliarden Parameter, `2b` entsprechend 2 Milliarden. Je mehr Parameter ein Modell hat, desto fähiger ist es in der Regel — aber desto mehr Speicher braucht es auch.

Modellgrößen vs. VRAM

Wie groß darf ein Modell sein, damit es auf deiner Hardware flüssig läuft? Entscheidend ist der Arbeitsspeicher deiner Grafikkarte (VRAM). Als Faustregel gilt: Das Modell sollte samt Kontext vollständig in den VRAM passen. Tut es das nicht, lagert Ollama Teile auf den normalen RAM oder die Festplatte aus — das funktioniert, wird aber spürbar langsamer.

Diese Tabelle gibt dir eine grobe Orientierung (bei üblicher 4-Bit-Quantisierung):

| Modellgröße | ungefährer Speicherbedarf | typische Hardware |

| --- | --- | --- |

| 2B–3B | 2–3 GB | CPU oder kleine GPU |

| 7B–8B | 4–6 GB | GPU mit 8 GB VRAM |

| 13B–14B | 8–10 GB | GPU mit 12–16 GB VRAM |

| 32B–33B | 20–24 GB | GPU mit 24 GB oder mehr |

| 70B+ | 40 GB und mehr | mehrere GPUs / Server-Hardware |

Für den Start mit einer mittleren Grafikkarte (8 bis 12 GB VRAM) ist ein Modell im Bereich 7B bis 8B der beste Kompromiss aus Qualität und Geschwindigkeit. Wer nur eine CPU ohne Grafikkarte hat, bleibt bei 2B- bis 3B-Modellen — die sind klein genug, um auch so flüssig zu antworten.

Quantisierung: Q4 und Q8 erklärt

Modelle werden ursprünglich mit hoher Präzision trainiert: Jedes Gewicht belegt mehrere Bytes. Diese Präzision ist für den normalen Betrieb aber gar nicht nötig. Bei der **Quantisierung** werden die Zahlen mit weniger Bits gespeichert — vergleichbar mit einem Foto, das du von RAW in ein stark komprimiertes JPEG umwandelst. Das Modell wird deutlich kleiner und schneller, verliert dabei aber ein wenig an Genauigkeit.

Die gängigen Stufen heißen **Q4** und **Q8**:

- **Q4 (4 Bit)**: Die Gewichte werden auf 4 Bit pro Wert reduziert. Das Modell schrumpft auf etwa ein Viertel der Originalgröße und läuft deutlich schneller. Der Qualitätsverlust ist für Alltagsaufgaben meist kaum spürbar. Q4 ist die Standardwahl für lokale LLMs.
- **Q8 (8 Bit)**: Hier bleiben 8 Bit pro Wert erhalten. Das Modell ist doppelt so groß wie bei Q4, liefert aber eine Qualität, die dem unquantisierten Original sehr nahe kommt. Q8 lohnt sich, wenn du genug Speicher hast und höchste Genauigkeit willst — etwa für Übersetzungen oder komplexe Texte.

Bei Ollama erkennst du die Quantisierung am Tag hinter dem Modellnamen. `llama3.2:3b` ist die Standardversion, während zum Beispiel `llama3.2:3b-q8_K_M` oder `llama3.2:3b-q4_K_M` explizit eine Quantisierungsstufe benennen. `_K_M` steht dabei für eine ausgewogene Mischung („Medium"). Für den Anfang genügt es, einfach die Standardversion ohne Tag zu nehmen — Ollama wählt dann automatisch eine sinnvolle Quantisierung.

Erste Chat-Tests

Sobald ein Modell gepullt ist, startest du den interaktiven Chat direkt in der Konsole:

```
ollama run llama3.2:3b
```

Du landest in einem Prompt, in dem du direkt Fragen eingeben kannst. Probiere ein paar typische Tests aus:

- **Allgemeinwissen**: „Erkläre in drei Sätzen, was ein Homelab ist."
- **Kreativ**: „Schreibe eine kurze Begrüßung für mein Heimnetz-Dashboard."
- **Logik**: „Wenn ein Zug mit 80 km/h fährt, wie weit kommt er in 45 Minuten?"

Beobachte dabei die Ausgabegeschwindigkeit: Läuft das Modell auf der GPU, erscheinen die Antworten fast sofort und fließend. Läuft es auf der CPU, siehst du die Wörter oft einzeln eintrudeln — ein klares Zeichen dafür, dass du ein kleineres Modell oder eine Grafikkarte brauchst.

Den Chat beendest du mit `/bye`. Mit `ollama run` und dem Befehl `/help` siehst du weitere Tastenkombinationen, zum Beispiel um ein Gespräch neu zu starten oder gespeicherte Sitzungen zu laden.

Weitere nützliche Befehle für den Alltag:

```
ollama list          # zeigt alle installierten Modelle
ollama show llama3.2:3b   # zeigt Details zu einem Modell
ollama rm llama3.2:3b     # entfernt ein Modell und gibt Speicher frei
```

Wenn Ollama läuft und das Modell antwortet, hast du den wichtigsten Meilenstein geschafft: Auf deinem eigenen Server läuft jetzt ein komplettes Sprachmodell — komplett lokal, ohne Internetverbindung und ohne dass Daten dein Haus verlassen. Im nächsten Schritt bekommst du dafür eine schöne grafische Oberfläche: OpenWebUI.

Kapitel 5: OpenWebUI – ein ChatGPT-ähnliches Interface für deine lokale KI

Ollama läuft, deine Modelle sind heruntergeladen – aber bisher redest du mit ihnen nur über die Kommandozeile. Das ist für Experimente in Ordnung, für den Alltag aber unpraktisch. Hier kommt OpenWebUI ins Spiel: eine vollständige Weboberfläche, die aussieht und sich anfühlt wie ChatGPT – nur eben komplett lokal, ohne Abo und ohne dass deine Daten das Haus verlassen.

Was ist OpenWebUI?

OpenWebUI ist eine Open-Source-Anwendung, die dir ein browserbasiertes Chat-Interface bereitstellt. Du öffnest einfach `http://localhost:8080` im Browser und hast eine vertraute Chat-Oberfläche: links eine Liste deiner bisherigen Unterhaltungen, rechts das Chatfenster, unten das Eingabefeld. Wer ChatGPT kennt, findet sich in Sekunden zurecht.

Der entscheidende Unterschied: OpenWebUI spricht nicht mit den Servern von OpenAI, sondern mit deinem eigenen Ollama-Server. OpenWebUI nutzt die OpenAI-kompatible API von Ollama, die standardmäßig unter Port 11434 lauscht. Du kannst OpenWebUI also auch später problemlos auf andere Backends umstellen – etwa einen entfernten Server, einen zweiten Rechner oder einen kommerziellen API-Anbieter.

Installation

Der einfachste Weg ist Docker. Wenn du die Ollama-Installation aus einem früheren Kapitel schon hinter dir hast, ist Docker vermutlich ohnehin schon eingerichtet – ansonsten installiere Docker Desktop und starte es einmal.

Öffne ein Terminal und führe folgenden Befehl aus:

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:main
```

Was passiert hier? Der Container wird im Hintergrund (`-d`) gestartet, Port 3000 deines Rechners wird auf Port 8080 im Container gemappt – du erreichst die Oberfläche also unter `http://localhost:3000` . Der Parameter `--add-host=host.docker.internal:host-gateway` sorgt dafür, dass der Container deinen Ollama-Server erreichen kann, der ja außerhalb des Containers läuft. Das Volume `open-webui` speichert deine Chats und Einstellungen dauerhaft, und `--restart always` startet OpenWebUI bei jedem Neustart automatisch mit.

Du kannst OpenWebUI aber auch ganz ohne Docker installieren, wenn du Python auf dem Rechner hast:

```
pip install open-webui
open-webui serve
```

Nach der Installation öffnest du `http://localhost:3000` (bzw. `http://localhost:8080` bei der Python-Variante). Beim ersten Start legst du ein Administratorkonto an – das ist das erste Konto, das angelegt wird, und es bekommt automatisch Admin-Rechte.

Verbindung zu Ollama

Normalerweise findet OpenWebUI deinen Ollama-Server automatisch. Falls nicht, klicke oben rechts auf dein Profilbild, gehe zu **Einstellungen** → **Verbindungen** und trage unter „Ollama Base URL" ein:

```
http://host.docker.internal:11434
```

(Falls du Ollama und OpenWebUI direkt auf demselben Rechner ohne Docker betreibst, ist es schlicht `http://localhost:11434`.)

Sobald die Verbindung steht, erscheinen oben im Chatfenster alle Modelle, die du mit `ollama pull` heruntergeladen hast. Du kannst sie per Dropdown auswählen und sofort losschreiben.

Die wichtigsten Features

OpenWebUI kann deutlich mehr als nur chatten. Hier die Funktionen, die du als Einsteiger wirklich nutzen wirst:

Modelle durchschalten. Über das Dropdown oben wechselst du zwischen Modellen – zum Beispiel einem großen Modell für anspruchsvolle Aufgaben und einem kleinen, schnellen für einfache Fragen.

Eigene Dokumente als Wissensbasis. Über **Arbeitsbereich** → **Dokumente** kannst du PDFs, Textdateien oder Markdown-Dateien hochladen. OpenWebUI zerlegt sie in Abschnitte und erstellt daraus eine durchsuchbare Wissensbasis. Im Chat fragst du dann „Was steht in meinem Dokument zu …?" – und das Modell antwortet auf Basis deiner Dateien statt nur aus seinem allgemeinen Wissen. Das nennt man Retrieval-Augmented Generation (RAG), und es ist eines der nützlichsten Werkzeuge überhaupt: Dein Modell kann plötzlich deine eigene Dokumentation, Verträge oder Notizen „kennen".

Web-Suche. Wenn du in den Einstellungen eine Such-API (etwa von SearXNG) hinterlegst, kann das Modell aktuelle Informationen aus dem Internet beziehen – nützlich, weil lokale Modelle sonst nur ihr Trainingswissen kennen.

System-Prompts und Modell-Dateien. Du kannst für jedes Modell festlegen, wie es sich verhalten soll – zum Beispiel „Antworte immer auf Deutsch und halte dich kurz". Das passiert über die Modelleinstellungen oder eigene Modell-Dateien.

Code-Hervorhebung und Formatierung. Codeblöcke werden mit Syntax-Hervorhebung angezeigt, Markdown wird gerendert, Tabellen und Listen werden sauber dargestellt.

Mehrere Nutzer verwalten

Ein großer Vorteil von OpenWebUI gegenüber einem rein lokalen Setup: Es ist von Anfang an mehrbenutzerfähig. Du kannst also deinen Rechner als kleinen „KI-Server" für die ganze Familie oder das Büro nutzen.

Über **Einstellungen** → **Benutzer** (Admin-Bereich) legst du weitere Konten an. Jeder Nutzer bekommt seine eigenen Chats, seine eigenen hochgeladenen Dokumente und seine eigenen Einstellungen – die Unterhaltungen der anderen sieht er nicht. Du als Admin kannst außerdem festlegen, welche Modelle für wen sichtbar sind, und Rollen vergeben (Admin, Benutzer, nur lesend).

Das Anmelden funktioniert einfach über die Login-Seite – jeder im Netzwerk kann dann unter `http://<deine-IP>:3000` auf die Oberfläche zugreifen, solange dein Rechner erreichbar ist. Für den Einstieg im Heimnetz reicht das völlig; wenn du OpenWebUI ins Internet stellen willst, kommst du um zusätzliche Absicherung (Reverse-Proxy, HTTPS, starke Passwörter) nicht herum.

Modellauswahl: welches Modell für welchen Zweck?

Die Qualität deiner Antworten hängt stark davon ab, welches Modell du wählst. Als Faustregel für den Einstieg:

- **Kleine Modelle (7B-Klasse):** schnell, laufen auch auf schwächerer Hardware, gut für kurze Fragen, Zusammenfassungen und einfache Aufgaben.
- **Mittlere Modelle (13B–30B):** spürbar besser bei Logik, Deutsch und längeren Texten – brauchen aber mehr RAM und Geduld.
- **Große Modelle (70B+):** am nächsten an kommerziellen Diensten dran, aber nur sinnvoll mit viel VRAM oder einem starken Rechner.

Experimentiere ruhig: Du kannst in OpenWebUI das Modell mitten im Gespräch wechseln und dieselbe Frage an mehrere Modelle stellen, um die Unterschiede zu sehen. Genau diese Vergleichbarkeit macht OpenWebUI zur idealen Spielwiese, um dein Homelab kennenzulernen.

Kapitel 6: ComfyUI – Bilder generieren wie Midjourney

Chatten ist nur die eine Hälfte der lokalen KI. Die andere, deutlich visuellere Hälfte ist die Bildgenerierung – und hier ist ComfyUI das Werkzeug der Wahl. Während Dienste wie Midjourney ihre Bilder auf fremden Servern und gegen Abo berechnen, erzeugst du mit ComfyUI und Stable Diffusion Bilder direkt auf deiner eigenen Grafikkarte – unbegrenzt, kostenlos und ohne Zensurfilter.

Was ist ComfyUI?

ComfyUI ist eine grafische, knotenbasierte Oberfläche für Stable Diffusion. Statt eines einfachen „Prompt rein, Bild raus"-Formulars baust du deinen Workflow aus einzelnen Bausteinen zusammen, die du mit Linien verbindest: ein Knoten lädt das Modell, einer nimmt deinen Prompt entgegen, einer steuert den Sampler, einer speichert das Bild. Das sieht auf den ersten Blick komplizierter aus als ein simples Eingabefeld, ist aber enorm flexibel – und du musst die Knoten gar nicht selbst bauen: Es gibt unzählige fertige Workflows zum Importieren.

Nach dem Start erreichst du ComfyUI unter `http://localhost:8188`.

Installation

Der schnellste Weg auf Windows ist die portable Version, die es direkt auf der GitHub-Seite von ComfyUI als Download gibt (ComfyUI_windows_portable). Du entpackst das Archiv in einen Ordner und startest die enthaltene `run_nvidia_gpu.bat` – fertig. Die portable Version bringt Python, alle Abhängigkeiten und die nötigen Bibliotheken schon mit.

Der klassische Weg über Git funktioniert ebenfalls:

```
git clone https://github.com/comfyanonymous/ComfyUI
cd ComfyUI
```

```
pip install -r requirements.txt
python main.py
```

ComfyUI läuft im Browser, die eigentliche Rechenarbeit übernimmt aber deine Grafikkarte. Beim ersten Start legt ComfyUI die Ordnerstruktur an, die du für Modelle brauchst – am wichtigsten ist `ComfyUI/models/checkpoints`.

Stable-Diffusion-Modelle

Ein „Modell" ist hier kein fertiges Bildprogramm, sondern eine Datei mit trainierten Gewichten – meist mehrere Gigabyte groß. Zwei Varianten sind für den Einstieg entscheidend:

- **SD 1.5:** die ältere, leichte Generation. Läuft flüssig auch auf 8 GB VRAM, riesige Auswahl an angepassten Modellen, sehr schnell. Ideal zum Lernen und für die meisten alltäglichen Bilder.
- **SDXL:** die neuere, größere Generation. Deutlich mehr Detail, bessere Schrift und Komposition – braucht dafür aber mehr VRAM und Rechenzeit.

Du lädst ein Modell (meist im `.safetensors`-Format) von Plattformen wie Civitai oder Hugging Face herunter und legst es in den Ordner `models/checkpoints`. Danach erscheint es im Auswahlfeld des Modell-Knotens.

Ein erster Workflow

Wenn du ComfyUI öffnest, ist ein einfacher Standard-Workflow schon geladen: Ein Knoten für das Modell, ein leerer Bereich für den positiven Prompt (was du sehen willst), ein Bereich für den negativen Prompt (was du *nicht* sehen willst), ein Sampler und ein Ausgabefeld. Du tippst deinen Prompt ein, wählst dein Modell, klickst auf **Queue Prompt** – und nach kurzer Zeit erscheint dein Bild.

Für den Start genügt es, dieses Grundgerüst zu benutzen. Wenn du tiefer einsteigst, lädst du Workflows von anderen Nutzern herunter (oft als JSON-Dateien oder als Bild mit eingebettetem Workflow), ziehst sie einfach ins Browserfenster und hast sofort komplexe Pipelines.

Inpainting: Bilder gezielt verändern

Eine der mächtigsten Fähigkeiten von ComfyUI ist das Inpainting. Damit veränderst du nicht das ganze Bild, sondern nur einen ausgewählten Bereich – zum Beispiel ersetzt du einen störenden Gegenstand oder verpasst einer Person andere Kleidung.

Der Ablauf: Du lädst dein Bild in einen Bild-Knoten, maskierst mit dem Pinsel im Editor den Bereich, den du ändern willst, und beschreibst im Prompt, was dort stattdessen entstehen soll. Stable Diffusion generiert dann nur diesen Bereich neu und fügt ihn nahtlos in den Rest des Bildes ein. Für gute Ergebnisse gibt es spezielle Inpainting-Modelle (oft mit dem Zusatz „inpainting" im Namen), die für genau diese Aufgabe trainiert wurden.

Welche Grafikkarte brauchst du?

Die Grafikkarte ist bei der Bildgenerierung der entscheidende Faktor. Zwei häufige Karten als Beispiele:

- **NVIDIA GTX 1080 Ti (11 GB VRAM):** ein Klassiker, der auch heute noch gut mithalten kann. SD 1.5 läuft damit flott und ohne Einschränkungen. SDXL ist machbar, braucht aber Geduld und gelegentlich den Start mit dem Parameter `--lowvram`, der ComfyUI dazu bringt, Teile des Modells in den normalen Arbeitsspeicher auszulagern. Für den Einstieg völlig ausreichend.
- **AMD RX 6900 XT (16 GB VRAM):** viel Speicher und ordentlich Rechenleistung, aber mit einer Einschränkung: ComfyUI und Stable Diffusion sind historisch stark auf NVIDIA (CUDA) zugeschnitten. Unter Windows läuft die RX 6900 XT über DirectML, was funktioniert, aber langsamer und manchmal hakelig ist. Unter Linux mit ROCm läuft AMD deutlich besser. Wenn du eine AMD-Karte hast, rechne mit mehr Einrichtungsaufwand und etwas niedrigeren Geschwindigkeiten – möglich ist es trotzdem.

Merke dir die Faustregel: VRAM ist wichtiger als reine Rechenleistung. 8 GB sind das Minimum für SD 1.5, für komfortables SDXL sind 12–16 GB sinnvoll.

Tipps für gute Prompts

Das Ergebnis steht und fällt mit deinem Prompt. Diese Regeln helfen:

1. **Sei konkret und beschreibend.** Statt „eine Landschaft" schreibst du „ein verschneiter Kiefernwald bei Sonnenaufgang, Nebel zwischen den Bäumen, Fotografie, 85mm, weiches Licht".

2. **Definiere den Stil.** Begriffe wie „Foto", „Ölgemälde", „3D-Render", „Anime", „Aquarell" oder „Comic" lenken das Ergebnis massiv.

3. **Nutze den negativen Prompt.** Was möchtest du nicht? Typische Füllwörter sind „verschwommen, verzerrt, zusätzliche Finger, schlechte Anatomie, Wasserzeichen".

4. **Reihenfolge ist Gewichtung.** Was vorne im Prompt steht, bekommt mehr Aufmerksamkeit. Das wichtigste Motiv gehört nach vorn.

5. **Baue auf bestehenden Prompts auf.** Viele geteilte Bilder auf Civitai zeigen ihren Prompt direkt mit an. Kopiere sie, passe sie an und lerne so, welche Formulierungen funktionieren.

ComfyUI sieht anfangs einschüchternd aus, aber der Einstieg ist schnell geschafft: Modell laden, Prompt tippen, generieren. Von dort aus kannst du dich Schritt für Schritt in Workflows, Inpainting und eigene Modelle vorarbeiten – und irgendwann Bilder erzeugen, die mit kostenpflichtigen Diensten mithalten.

Kapitel 7: Stromkosten im Griff – was der Server wirklich kostet

Viele Einsteiger unterschätzen, dass ein Homelab-Server rund um die Uhr läuft. 24 Stunden am Tag, 365 Tage im Jahr. Während dein Desktop-PC nach der Arbeit heruntergefahren wird, arbeitet dein Server weiter – und zieht dabei dauerhaft Strom. Bevor du also begeistert Hardware bestellst, lohnt es sich, einmal nachzurechnen, was dich das Gerät im Dauerbetrieb tatsächlich kostet. Die gute Nachricht: Mit ein paar einfachen Entscheidungen hältst du die Kosten klein.

Leistungsaufnahme: Idle versus Last

Der wichtigste Begriff zuerst: die Leistungsaufnahme in Watt. Ein Server verbraucht nicht immer gleich viel Strom. Im Leerlauf („Idle"), wenn keine Aufgabe ansteht, läuft er auf Sparflamme. Sobald du ein großes Sprachmodell abfragst, Video-Streams umwandelst oder Backups schreibst, steigt der Verbrauch – das ist die „Last".

Ein typischer Mini-PC als Homeserver, etwa mit einem sparsamen Intel-Prozessor der N-Serie, liegt im Idle bei ungefähr 8 bis 15 Watt und unter Volllast bei 30 bis 60 Watt. Ein ausgedienter Desktop-PC oder ein älterer Tower-Server ist deutlich durstiger: 40 bis 80 Watt im Idle und 150 bis 300 Watt unter Last sind keine Seltenheit. Ein voll ausgebautes Rack-System mit mehreren Festplatten, zusätzlicher Grafikkarte und redundanten Netzteilen kann dauerhaft 300 bis 500 Watt ziehen – das merkt man dann deutlich auf der Stromrechnung.

Für die Praxis heißt das: Wer nur einen Nextcloud-Speicher, ein paar Docker-Container und ab und zu ein kleines lokales Sprachmodell betreiben will, braucht keinen Server, der 200 Watt Dauerlast frisst. Ein sparsamer Mini-PC tut es oft genauso gut.

Die Beispielrechnung bei 0,40 €/kWh

Wir rechnen mit dem aktuell realistischen deutschen Strompreis von 0,40 € pro Kilowattstunde. Die Formel ist einfach: Leistung in Watt mal Betriebsstunden, geteilt durch 1000 ergibt die Kilowattstunden. Ein Gerät mit 100 Watt verbraucht pro Tag 2,4 Kilowattstunden (100 Watt × 24 Stunden). Das macht im Monat bei 30 Tagen 72 Kilowattstunden und im Jahr 876 Kilowattstunden.

Schauen wir uns drei typische Szenarien an – jeweils im Dauerbetrieb, also 24/7:

100 Watt Dauerlast: 2,4 kWh pro Tag. Im Monat 72 kWh, das kostet 28,80 €. Aufs Jahr gerechnet sind es 876 kWh und damit 350,40 €.

200 Watt Dauerlast: 4,8 kWh pro Tag. Im Monat 144 kWh, das sind 57,60 €. Im Jahr kommen 1752 kWh zusammen, also 700,80 €.

400 Watt Dauerlast: 9,6 kWh pro Tag. Im Monat 288 kWh für 115,20 €. Aufs Jahr sind das 3504 kWh – satte 1401,60 €.

Die Faustregel: Jedes Watt Dauerleistung kostet dich bei 0,40 €/kWh im Jahr etwa 3,50 €. Ein Server, der dauerhaft 50 Watt mehr zieht als nötig, macht also rund 175 € Jahreskosten aus. Genau deshalb lohnt es sich, beim Kauf auf den Idle-Verbrauch zu schauen – nicht auf die auf dem Papier tolle Rechenleistung.

Wichtig: Diese Rechnung gilt nur für den Dauerbetrieb. Viele Einsteiger fahren ihren Server anfangs gar nicht durchgehend, sondern nur abends oder am Wochenende. Wer den Server täglich nur vier Stunden laufen lässt, zahlt nur ein Sechstel – aus den 350 € Jahreskosten bei 100 Watt werden dann rund 58 €.

Energiespartipps für dein Homelab

Der größte Hebel ist die Hardwarewahl. Ein moderner Mini-PC mit effizientem Prozessor ist fast immer die richtige erste Wahl – klein, leise und im Idle oft unter 15 Watt. Vermeide alte Gaming-Rechner oder ausgemusterte Server, die mit üppigen Netzteilen und vielen rotierenden Festplatten dauerhaft Strom ziehen.

Der zweite Hebel ist die Laufzeit. Überlege ehrlich, ob dein Server wirklich 24/7 laufen muss. Ein täglicher automatischer Start und Stopp per Zeitplan („Wake-on-LAN" oder ein einfacher Cron-Job zum Herunterfahren) spart sofort. Viele Aufgaben wie Backups oder Downloads lassen sich in die Nacht legen – oder eben nur dann laufen lassen, wenn du den Dienst brauchst.

Auch die Festplatten spielen eine Rolle. Klassische 3,5-Zoll-Festplatten brauchen pro Stück 5 bis 10 Watt, im Dauerbetrieb summiert sich das. Wenn möglich, setze auf eine oder zwei große Festplatten statt vieler kleiner – oder auf sparsame SSDs, die im Idle fast nichts verbrauchen. Und

aktiviere den Energiesparmodus: Viele Systeme können die Festplatten bei Inaktivität in den Schlaf schicken.

Zu guter Letzt: Miss nach. Ein günstiges Energiemessgerät für die Steckdose (ab 15 €) zeigt dir sofort, was dein Aufbau wirklich zieht – im Idle und unter Last. Nur so erkennst du, ob dein Server tatsächlich die erhofften 20 Watt zieht oder doch heimlich 60.

Wann sich ein Server lohnt

Die Gegenrechnung: Was kostet dich die Alternative? Ein kleines Cloud-VPS mit 2 Kernen und 4 GB RAM liegt bei 5 bis 10 € im Monat, also 60 bis 120 € im Jahr. Ein Server, der bei 15 Watt Idle nur 131 kWh im Jahr verbraucht, kostet dich rund 52 € Strom – und kann deutlich mehr speichern und rechnen als so ein Mini-VPS. Spätestens wenn du mehrere Dienste, viel Speicher oder ein lokales KI-Modell betreiben willst, schlägt der eigene Server die Cloud oft schon nach ein bis zwei Jahren.

Die klare Empfehlung: Ein sparsamer Mini-PC für 150 bis 400 € ist der wirtschaftlichste Einstieg. Er rechnet sich bei 24/7-Betrieb gegen Cloud-Dienste meist innerhalb von ein bis zwei Jahren. Teure, stromhungrige Server lohnen sich erst, wenn du wirklich viel Rechenleistung brauchst – und dann solltest du die Stromkosten bewusst einkalkulieren. Rechne also vorher, nicht nachher.

Kapitel 8: Sicherheit – dein Server, deine Regeln

Ein eigener Server ist ein bisschen wie eine eigene Haustür: Du bestimmst, wer hereinkommt. Aber sobald dein Server am Netzwerk hängt und im Idealfall sogar aus dem Internet erreichbar ist, klopfen auch ungebetene Gäste an. Automatisierte Bots scannen das Internet pausenlos nach offenen Türen. Die gute Nachricht: Mit ein paar Grundregeln machst du deinen Server so unattraktiv, dass die Angreifer lieber weiterziehen.

Die Firewall: deine erste Verteidigungslinie

Eine Firewall entscheidet, welcher Datenverkehr deinen Server erreichen darf und welcher nicht. Auf Linux-Servern ist UFW (Uncomplicated Firewall) der einfachste Einstieg. Die Grundregel lautet: erst einmal alles zu, dann gezielt öffnen, was du wirklich brauchst.

In der Praxis sieht das so aus: Du aktivierst UFW, erlaubst eingehende Verbindungen nur für die Dienste, die du nutzt – etwa SSH (Port 22), einen Webserver (Port 80 und 443) oder deinen VPN

(WireGuard). Alles andere bleibt standardmäßig gesperrt. Drei Befehle genügen für den Start: `sudo ufw enable` schaltet die Firewall ein, `sudo ufw allow 22/tcp` öffnet SSH, und `sudo ufw status` zeigt dir den aktuellen Zustand. Was du nicht ausdrücklich erlaubst, kommt nicht rein.

SSH-Schlüssel statt Passwort

SSH ist dein Werkzeug, um dich auf dem Server anzumelden. Wer dabei nur ein Passwort nutzt, spielt den Bots in die Hände: Die versuchen rund um die Uhr, gängige Passwörter durchzuprobieren. Die Lösung ist ein SSH-Schlüsselpaar – ein privater Schlüssel, der nur auf deinem Rechner liegt, und ein öffentlicher, der auf dem Server hinterlegt wird.

Das Vorgehen: Auf deinem Rechner erzeugst du mit `ssh-keygen` ein Schlüsselpaar, dann kopierst du den öffentlichen Teil mit `ssh-copy-id` auf den Server. Anschließend stellst du die Passwort-Anmeldung ab, indem du in der SSH-Konfiguration `PasswordAuthentication no` setzt. Ab jetzt klappt die Anmeldung nur noch mit deinem Schlüssel – wer ihn nicht hat, kommt nicht rein. Den privaten Schlüssel sicherst du zusätzlich mit einer Passphrase und verwahrst eine Kopie sicher, denn wer den Schlüssel verliert, kann sich ausgesperrt haben.

Updates: die unspektakuläre Pflicht

Viele Sicherheitslücken entstehen nicht durch geniale Angriffe, sondern durch veraltete Software. Sobald eine Lücke bekannt wird, veröffentlichen die Hersteller ein Update – und die Angreifer wissen genau, welche Systeme noch ungepatcht sind. Deshalb gilt: Updates sind keine lästige Pflicht, sondern dein wichtigstes Sicherheitswerkzeug.

Am einfachsten automatisierst du das. Auf Debian- und Ubuntu-Systemen installiert `unattended-upgrades` Sicherheitsupdates automatisch im Hintergrund. Du aktivierst es einmal, und ab dann hält sich dein System selbst aktuell. Zusätzlich lohnt sich ein regelmäßiger Blick, ob alle installierten Dienste – gerade Docker-Container – auf dem neuesten Stand sind. Ein Container, den du vor einem Jahr gestartet hast und nie aktualisiert hast, ist ein offenes Scheunentor.

Kein offener Port nach außen

Die sicherste Regel ist simpel: Öffne gar nichts nach außen, solange es nicht absolut nötig ist. Ein Homeserver muss für den Anfang überhaupt nicht aus dem Internet erreichbar sein. Solange alles nur in deinem Heimnetz läuft, kannst du auf die meisten Sicherheitsmaßnahmen entspannter verzichten.

Wenn du doch von unterwegs auf deinen Server zugreifen willst, öffne nicht einfach Ports am Router und leite sie zum Server weiter. Der deutlich sicherere Weg ist ein VPN: Du baust einen

verschlüsselten Tunnel zu deinem Heimnetz auf, und erst durch diesen Tunnel erreichst du deine Dienste – als wärst du zu Hause. Nach außen ist dann nur ein einziger Port für den VPN offen, alles andere bleibt verborgen.

VPN mit WireGuard: der sichere Tunnel

WireGuard ist der moderne Standard für genau diesen Zweck. Es ist schnell, schlank und vor allem einfach einzurichten – die Konfigurationsdatei besteht im Kern aus wenigen Zeilen. Du installierst WireGuard auf deinem Server (oder direkt auf deinem Router, falls der es unterstützt), erzeugst für jedes deiner Geräte ein Schlüsselpaar und trägst die Geräte in die Konfiguration ein.

Auf deinem Smartphone oder Laptop installierst du die WireGuard-App, importierst die Konfigurationsdatei und aktivierst den Tunnel. Ab jetzt läuft dein gesamter Datenverkehr zu deinem Heimnetz verschlüsselt. Der Clou: WireGuard antwortet nur, wenn das anklopfende Gerät den richtigen Schlüssel vorweist. Für jeden automatisierten Scanner im Internet wirkt der VPN-Port wie eine verschlossene Tür ohne Türklinke – es gibt schlicht nichts zu knacken.

Fail2ban: aufdringliche Bots abwimmeln

Auch wenn du alles richtig machst, versuchen Bots weiterhin, sich über SSH anzumelden. Fail2ban beobachtet deine Logdateien und sperrt IP-Adressen, die wiederholt mit falschen Zugangsdaten scheitern. Nach ein paar Fehlversuchen wird die Adresse für eine bestimmte Zeit blockiert.

Die Einrichtung ist denkbar einfach: Fail2ban installieren, den Dienst starten, fertig – die Standardregeln für SSH sind bereits enthalten. Du kannst die Sperrzeit und die Anzahl der erlaubten Fehlversuche anpassen. Zusammen mit SSH-Schlüsseln hast du damit einen doppelten Schutz: Selbst wer deinen Port findet, kommt nicht durch, und wer es zu oft versucht, fliegt raus.

Backups als Teil der Sicherheit

Sicherheit endet nicht an der Firewall. Die größte Gefahr für deine Daten ist oft nicht der Hacker, sondern der Festplattenausfall, ein fehlerhaftes Update oder ein Bedienfehler – und nicht zuletzt Ransomware. Genau deshalb gehört das Backup untrennbar zur Sicherheit dazu: Es ist deine Versicherung gegen den schlimmsten Fall.

Die bewährte 3-2-1-Regel hilft dir beim Planen: drei Kopien deiner Daten, auf zwei verschiedenen Speichermedien, eine davon an einem anderen Ort. Konkret heißt das: Die Originaldaten liegen auf dem Server, eine Kopie auf einer externen Festplatte im Haus und eine weitere Kopie außerhalb – etwa bei einem zweiten Standort oder einem verschlüsselten Cloud-Backup.

Wichtig ist vor allem, dass Backups automatisch laufen und du sie regelmäßig testest. Ein Backup, von dem du nicht weißt, ob es sich zurückspielen lässt, ist kein Backup, sondern ein gutes Gefühl. Mit Tools wie BorgBackup oder restic erstellst du verschlüsselte, versionierte Sicherungen, die du einfach auf eine zweite Platte oder einen entfernten Speicher schieben kannst. Plane von Anfang an ein automatisches Backup ein – nicht erst, wenn die erste Festplatte schon verdächtig klickt.

Das Fundament steht

Mehr braucht es für den Einstieg nicht: Firewall an, SSH-Schlüssel statt Passwort, automatische Updates, keine offenen Ports nach außen, WireGuard für den Fernzugriff, Fail2ban gegen aufdringliche Bots und ein funktionierendes Backup. Damit ist dein Server deutlich besser geschützt als die meisten Geräte im Internet – und du hast die Kontrolle über deine eigene kleine Infrastruktur.

Kapitel 9: Backups – damit nichts verloren geht

Ein Homelab lebt von seinen Daten. Die Fotos der Familie, wichtige Dokumente, die Konfiguration deiner virtuellen Maschinen, die Datenbank deiner Nextcloud – all das existiert oft nur ein einziges Mal. Ein Festplattenausfall, ein fehlgeschlagenes Update oder schlicht ein Fehler beim Kopieren kann binnen Sekunden Monate Arbeit vernichten. Deshalb gilt im Homelab eine eiserne Grundregel: Was nicht mindestens einmal gesichert ist, existiert nicht. In diesem Kapitel baust du eine Backup-Strategie auf, die auch dem schlimmsten Fall standhält – vom Prinzip der 3-2-1-Regel über den Proxmox-eigenen Sicherungsmechanismus vzdump bis hin zur vollautomatischen Sicherung per systemd-Timer.

Warum Backups überhaupt?

Viele Einsteiger verschieben das Thema Backup auf später, weil „bisher ja nichts passiert ist". Genau das ist der Fehler. Datenverlust ist kein seltenes Ereignis, sondern eine Frage der Zeit: Festplatten haben eine begrenzte Lebensdauer, Netzteile können andere Komponenten mit in den Tod reißen, und auch Ransomware oder ein falsch ausgeführter Befehl mit `rm -rf` richten echten Schaden an. Ein Backup ist keine Versicherung gegen den Ausfall selbst, sondern gegen dessen Folgen. Und die Regel Nummer eins lautet: Ein Backup, das du noch nie zurückgespielt hast, ist keines. Nur was sich nachweislich wiederherstellen lässt, zählt.

Die 3-2-1-Regel

Die 3-2-1-Regel ist der Goldstandard der Datensicherung und lässt sich in drei Zahlen zusammenfassen:

- **3 Kopien** deiner Daten – das Original und zwei Sicherungen.
- **2 verschiedene Speichermedien** – zum Beispiel eine interne Festplatte und eine externe USB-Platte oder ein NAS.
- **1 Kopie außerhalb des Hauses** (offsite) – gegen Brand, Diebstahl oder Wasserschaden.

Warum zwei Medien? Wenn alle Kopien auf demselben Plattenmodell liegen, das einen Serienfehler hat, sind im Zweifel alle gleichzeitig tot. Und warum offsite? Gegen die eigene Wohnung als Single Point of Failure hilft keine noch so gute lokale Kopie. Für den Einstieg reicht es oft schon, eine verschlüsselte externe Festplatte regelmäßig zu befüllen und sie bei Freunden oder Verwandten zu deponieren. Wer mehr Komfort will, greift später zu einem kleinen NAS an einem zweiten Standort oder einem Cloud-Backup.

Backups in Proxmox: vzdump

Proxmox VE bringt mit **vzdump** ein mächtiges Bordmittel mit, das ganze virtuelle Maschinen und Container konsistent sichert. Der Befehl erzeugt standardmäßig ein komprimiertes Archiv, das du später eins zu eins wiederherstellen kannst.

Das einfachste Beispiel sichert die VM mit der ID 100 in das Standard-Backup-Verzeichnis:

```
vzdump 100
```

Sinnvoller ist es, Ziel, Kompression und Modus explizit anzugeben:

```
vzdump 100 --mode snapshot --compress zstd --dumpdir /mnt/backup
```

Der Modus `snapshot` nutzt die Snapshot-Funktion von ZFS oder LVM-Thin und erlaubt Backups bei laufender VM. Bei Containern ist der Modus standardmäßig `suspend` (kurzer Stopp) oder – bei unprivilegierten LXC – ebenfalls `snapshot`. Mit `--compress zstd` erreichst du eine sehr gute Kompression bei überschaubarer CPU-Last; alternativ stehen `gzip` oder `lzo` zur Verfügung.

Noch komfortabler ist die grafische Oberfläche: Unter **Datacenter** → **Backup** richtest du geplante Backups ein. Du wählst den Speicher (Storage), den Wochentag, die Uhrzeit und den

gewünschten Aufbewahrungsmodus. Mit **Retention** legst du fest, wie viele alte Backups aufgehoben werden, zum Beispiel „Letzte 7 tägliche und 4 wöchentliche". So verhinderst du, dass dir das Backup-Laufwerk volläuft, während du trotzdem mehrere Wiederherstellungspunkte behältst.

Ziel: externe Platte oder NAS

Ein Backup auf dieselbe Festplatte, auf der die VM liegt, schützt nur gegen logische Fehler, nicht gegen den Plattenausfall. Deshalb gehört das Ziel auf ein separates Medium. Zwei klassische Wege:

Externe USB-Platte: Sie ist günstig und schnell eingerichtet. Mounte sie zum Beispiel nach `/mnt/backup` und trage sie in Proxmox als Storage vom Typ **Directory** ein. Achte darauf, dass du die Platte nach dem Backup sauber aushängen kannst, wenn du sie für die Offsite-Ablage mitnimmst.

NAS (Netzwerk-Speicher): Ein NAS wie eine Synology oder ein selbstgebauter Server mit TrueNAS wird per NFS oder SMB/CIFS eingebunden. In Proxmox legst du dafür unter **Datacenter** → **Storage** → **Add** → **NFS** einen neuen Speicher an und gibst Server-IP sowie Exportpfad an. Der Vorteil: Backups laufen automatisch über Nacht, ohne dass du physisch eine Platte anstöpseln musst. Der Nachteil: Wenn das NAS im selben Haus steht, ist die Offsite-Komponente der 3-2-1-Regel noch offen – kombiniere es also mit einer zusätzlichen externen oder Cloud-Kopie.

Für die Einbindung einer externen Platte per Kommandozeile hilft ein Blick auf `lsblk`, um die Gerätebezeichnung zu finden, gefolgt von:

```
mount /dev/sdb1 /mnt/backup
```

Ob ein Storage in Proxmox als Backup-Ziel taugt, siehst du an der Spalte „Content", in der der Eintrag **VZDump backup file** aktiviert sein muss.

Wiederherstellung testen

Der wichtigste Teil eines Backups ist die Wiederherstellung – und sie muss geübt werden. In der Proxmox-Oberfläche klickst du auf die gesicherte VM, dann auf **Backup** und wählst das Archiv aus. Über **Restore** legst du fest, in welche Storage-Pools die virtuelle Festplatte zurückgespielt wird und unter welcher neuen VM-ID. Wichtig: Für einen Test stellst du die VM mit einer **anderen ID** wieder her, damit die laufende Original-VM nicht überschrieben wird. Läuft die

wiederhergestellte Kopie und die Daten sind vollständig lesbar, weißt du: Das Backup funktioniert wirklich.

Plane diesen Test regelmäßig ein, mindestens einmal im Quartal. Ein guter Rhythmus ist, sich den Termin in den Kalender zu legen – dann entdeckst du Fehler, bevor du auf das Backup angewiesen bist.

Automatisierung mit systemd-Timer

Proxmox selbst läuft auf Debian und bringt systemd mit. Zwar deckt der eingebaute Backup-Planer die meisten Fälle ab, aber für Aufgaben außerhalb von vzdump – etwa das Synchronisieren einer Nextcloud-Datenbank oder das Kopieren der Backups auf die Offsite-Platte – sind **systemd-Timer** das richtige Werkzeug. Sie funktionieren wie Cron, sind aber transparenter und besser protokolliert.

Ein Timer besteht aus zwei Dateien: einem Service, der die eigentliche Arbeit beschreibt, und einem Timer, der den Zeitplan festlegt. Ein Beispiel für einen Service, der täglich um 03:00 Uhr die Backups auf ein NAS spiegelt:

```
# /etc/systemd/system/backup-sync.service
[Unit]
Description=Backups auf das NAS synchronisieren

[Service]
Type=oneshot
ExecStart=/usr/bin/rsync -av --delete /var/lib/vz/dump/ user@192.168.1.50:/backup/

# /etc/systemd/system/backup-sync.timer
[Unit]
Description=Tägliche Backup-Synchronisation

[Timer]
OnCalendar=*-*-* 03:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

`Persistent=true` bewirkt, dass ein versäumter Lauf nachgeholt wird, wenn der Server zur geplanten Zeit ausgeschaltet war. Aktivieren tust du den Timer mit:

```
systemctl enable --now backup-sync.timer
```

Mit `systemctl list-timers` siehst du alle aktiven Timer samt nächstem Ausführungszeitpunkt, und `journalctl -u backup-sync.service` zeigt dir, ob der letzte Lauf fehlerfrei war.

Damit hast du die Grundlagen beisammen: ein durchdachtes Konzept nach der 3-2-1-Regel, zuverlässige VM-Sicherungen per vzdump, ein separates Ziel auf Platte oder NAS, einen regelmäßigen Wiederherstellungstest und die Automatisierung per systemd-Timer. Deine Daten sind jetzt so sicher, wie es in einem Homelab realistisch möglich ist.

Kapitel 10: Ausbau & Automatisierung – dein Server wächst mit

Dein Server läuft, die ersten Dienste sind eingerichtet, Backups funktionieren. Jetzt beginnt der Teil, der ein Homelab so faszinierend macht: Du baust deine eigene digitale Infrastruktur aus und automatisierst, was sich automatisieren lässt. In diesem Kapitel lernst du weitere Dienste kennen, die fast jedes Homelab bereichern, steigst in Docker als Standard-Verpackungsformat für Anwendungen ein und bringst mit n8n deine Automatisierungen auf ein neues Level. Am Ende werfen wir einen Blick darauf, wie du dein Setup skalierst, wenn dir eine Maschine nicht mehr reicht.

Weitere Dienste für dein Homelab

Vier Dienste sind bei Homelab-Betreibern besonders beliebt, weil sie sofort sichtbaren Nutzen bringen:

Pi-hole blockiert Werbung und Tracker auf Netzwerkebene, bevor sie überhaupt deine Geräte erreichen. Du installierst Pi-hole als Container oder direkt auf einem kleinen Linux-System und trägst seine IP als DNS-Server in deinem Router ein. Danach profitieren alle Geräte im Netz – Smartphones, Fernseher, Rechner – automatisch vom Werbeblocker. Pi-hole zeigt dir in einer Weboberfläche, welche Domains blockiert werden, und lässt dich eigene Listen pflegen.

Home Assistant ist die zentrale Schaltstelle für Smart Home. Es vereint Geräte verschiedenster Hersteller unter einer Oberfläche: Lichter, Thermostate, Sensoren, Rolläden. Weil Home Assistant

lokal läuft, bleiben deine Daten bei dir und funktionieren auch ohne Cloud. Es läuft hervorragend als VM in Proxmox oder als Docker-Container; für die volle Funktionsvielfalt inklusive Add-ons empfehlen viele die eigene VM-Variante.

Nextcloud ist deine private Cloud: Dateiablage, Kalender, Kontakte, Foto-Synchronisierung und kollaboratives Arbeiten in einem. Du erreichst sie von überall über einen Browser oder die App und behältst die Hoheit über deine Daten. Nextcloud lässt sich als LXC-Container in Proxmox oder per Docker betreiben und lässt sich mit Let's Encrypt absichern.

Jellyfin ist dein eigener Streaming-Dienst für Filme, Serien und Musik. Du legst deine Medien auf dem Server ab, Jellyfin organisiert sie, holt Metadaten und Cover und streamt sie auf Fernseher, Tablet oder Smartphone – ganz ohne Abo. Besonders schön ist die Kombination mit Hardware-Transkodierung: Wenn deine GPU das Umwandeln der Videoformate übernimmt, läuft das Streaming auch auf schwächeren Geräten flüssig.

Docker als Standard-Verpackungsformat

Viele dieser Dienste werden heute als **Docker-Container** verteilt. Ein Container bündelt eine Anwendung mit all ihren Abhängigkeiten, sodass sie überall gleich läuft – unabhängig vom Betriebssystem darunter. Das macht Installationen reproduzierbar und das Aufräumen einfach: Container wegwerfen, neu starten, fertig.

Der schnellste Einstieg ist **Docker Compose**. Du beschreibst deine Dienste in einer `docker-compose.yml`-Datei und startest sie mit einem Befehl. Ein minimales Beispiel für Pi-hole:

```
services:
  pihole:
    image: pihole/pihole:latest
    container_name: pihole
    environment:
      TZ: "Europe/Berlin"
      WEBPASSWORD: "ein-sicheres-passwort"
    ports:
      - "53:53/tcp"
      - "53:53/udp"
      - "80:80/tcp"
    volumes:
      - ./pihole:/etc/pihole
```

```
      - ./dnsmasq:/etc/dnsmasq.d
    restart: unless-stopped
```

Mit `docker compose up -d` startest du den Dienst im Hintergrund. Die `volumes`-Angaben sorgen dafür, dass die Konfiguration dauerhaft gespeichert wird, auch wenn du den Container neu erstellst. `restart: unless-stopped` bringt den Dienst nach einem Neustart automatisch zurück.

Für dein Proxmox-Setup gibt es zwei sinnvolle Wege, Docker zu nutzen: Du richtest eine eigene VM ein, die als Docker-Host dient (sauber und gut isoliert), oder du betreibst Docker in einem unprivilegierten LXC-Container. Für den Einstieg ist die VM der robustere Weg. So oder so gilt: Aktualisiere deine Images regelmäßig mit `docker compose pull && docker compose up -d`, damit du nicht mit veralteten Versionen und bekannten Sicherheitslücken arbeitest.

Automatisierung mit n8n

n8n ist ein Werkzeug zur Workflow-Automatisierung, das du selbst hostest – eine Open-Source-Alternative zu Diensten wie Zapier oder Make. Du baust Abläufe grafisch aus Bausteinen zusammen, die Daten von einem Dienst zum nächsten reichen. Weil n8n auf deinem eigenen Server läuft, bleiben deine Daten in deiner Hand.

Typische Szenarien fürs Homelab:

- **Benachrichtigungen:** Dein Server sendet dir eine Telegram- oder Matrix-Nachricht, wenn ein Backup fertig ist oder ein Dienst ausfällt.
- **Smart-Home-Logik:** Ein n8n-Workflow wertet Sensordaten aus Home Assistant aus und schaltet bei Abwesenheit Lichter und Heizung herunter.
- **Datensammeln:** n8n fragt regelmäßig APIs ab – Wetter, Börsenkurse, Bahnverbindungen – und schreibt die Ergebnisse in eine Datenbank oder eine Tabelle.
- **Dokumentenablage:** Neue Dateien in einem Ordner werden automatisch verschlagwortet, in Nextcloud einsortiert oder per E-Mail verschickt.

Der Einstieg ist denkbar einfach: n8n läuft als Docker-Container, du erreichst es über die Weboberfläche und ziehst dir die benötigten Knoten auf die Arbeitsfläche. Jeder Workflow hat einen **Trigger** – etwa einen Zeitplan oder ein eingehendes Webhook-Ereignis – und beliebig viele Folge-Schritte. Ein einfacher Zeitplan-Trigger, der dir jeden Morgen eine Zusammenfassung per Telegram schickt, ist in wenigen Minuten gebaut und ein guter erster Erfolg.

Skalierung: mehr GPUs, mehr Server

Irgendwann reicht die eine Maschine nicht mehr – etwa weil du lokale KI-Modelle betreiben willst und dafür mehr GPU-Leistung brauchst, oder weil du Dienste trennen möchtest. Dann skaliert dein Homelab in zwei Richtungen:

Mehr GPUs: Für lokale KI-Inferenz – zum Beispiel mit llama.cpp oder einem lokalen Sprachmodell – lohnt sich eine leistungsfähige Grafikkarte. Achte beim Kauf auf ausreichend VRAM, denn der bestimmt, wie große Modelle du laden kannst. Unter Proxmox kannst du die GPU per **PCIe-Passthrough** direkt an eine VM durchreichen, die dann die volle Leistung nutzt. Das erfordert etwas Einarbeitung (IOMMU im BIOS aktivieren, Treiber einrichten), ist aber gut dokumentiert.

Mehr Server: Ab zwei Maschinen kannst du einen Proxmox-Cluster bilden. Mehrere Nodes werden unter **Datacenter** → **Cluster** → **Create Cluster** zusammengeschlossen, und du verwaltest alle VMs zentral über eine Oberfläche. Ein Cluster ermöglicht Live-Migration: Du schiebst eine laufende VM per Drag-and-drop auf einen anderen Node, etwa vor Wartungsarbeiten – ohne Ausfallzeit. Für echte Hochverfügbarkeit mit automatischem Neustart ausgefallener VMs brauchst du zusätzlich geteilten Speicher, zum Beispiel ein NAS mit ausreichend Platz.

Wichtiger als jede einzelne Technik ist aber die innere Haltung: Baue dein Homelab Schritt für Schritt aus, dokumentiere, was du änderst, und sichere weiterhin alles nach der 3-2-1-Regel aus Kapitel 9. Dann wächst dein Server mit deinen Anforderungen – und du behältst dabei jederzeit die Kontrolle.