1. Warum Claude und Home Assistant gut zusammenpassen

Bevor wir uns in die Technik stürzen, möchte ich dir kurz erklären, warum diese Kombination überhaupt so spannend ist. Du sparst dir damit später eine Menge Ärger und verstehst von Beginn an, was du dort eigentlich aufbaust.

Home Assistant ist eine der mächtigsten Smart-Home-Plattformen, die wir aktuell zur Verfügung haben. Sie bringt unzählige Geräte unter ein Dach und gibt dir die volle Kontrolle über dein Zuhause. Der Preis dafür ist allerdings, dass die Anzahl an Entitäten schnell wächst und du irgendwann den Überblick verlierst. Wo war nochmal der Sensor für die Luftqualität im Schlafzimmer? Welche Automation schaltet abends das Licht im Flur? Bereits ab einigen hundert Entitäten wird das mühsam.

Hier kommt Claude ins Spiel. Claude ist ein Sprachmodell von Anthropic, das natürliche Sprache versteht und sich über das Model Context Protocol mit externen Systemen verbinden kann. Sobald du Claude die richtigen Werkzeuge in die Hand gibst, kann er deinen kompletten Smart-Home-Datenbestand durchsuchen, analysieren und auf Wunsch sogar verändern.

Statt selbst durch Listen zu scrollen, fragst du also einfach: Welche Lichter im Wohnzimmer sind gerade an? Oder du sagst: Erstelle mir eine Automation, die um 22 Uhr alle Außenleuchten ausschaltet. Claude kümmert sich um den Rest.

Damit das funktioniert, brauchen wir aber genau eine Sache: einen Übersetzer. Denn Claude versteht in seiner Grundausstattung nichts von deinem Home Assistant. Dieser Übersetzer heißt MCP-Server, und genau den werden wir im nächsten Kapitel kennenlernen.

> **Kostenloser Account reicht**
>
> Für alles, was wir in diesem Guide tun, benötigst du kein Claude-Abo. Anthropic bietet die MCP-Funktionalität auch im kostenlosen Plan an. Erst wenn du sehr intensiv mit Claude arbeitest und an die Nachrichtengrenzen stößt, ist ein Upgrade sinnvoll.

2. Was ist MCP? – Grundlagen für Einsteiger

Bevor wir mit der Installation loslegen, sollten wir kurz über den Begriff sprechen, der dir in diesem Guide noch oft begegnen wird: MCP. Wenn du diesen Abschnitt verstanden hast, weißt du, warum wir überhaupt eine zusätzliche Software brauchen und was sie genau tut.

2.1 Der Begriff MCP einfach erklärt

MCP steht für Model Context Protocol. Das klingt erst einmal sperrig, ist aber im Kern eine ziemlich elegante Idee. Anthropic, also die Firma hinter Claude, hat im November 2024 einen offenen Standard veröffentlicht, mit dem sich KI-Modelle einheitlich an externe Tools, Dienste und Datenquellen anbinden lassen.

Stell dir MCP wie eine universelle Steckerleiste vor. Vor MCP hatte jede KI-Integration ihren eigenen Stecker. Wer ChatGPT mit Home Assistant verbinden wollte, brauchte eine andere Lösung als jemand, der das mit Gemini tun wollte. Mit MCP haben wir nun einen einheitlichen Standard, an den sich beide Seiten halten.

Auf der einen Seite steht der MCP-Client. Das ist in unserem Fall Claude Desktop. Auf der anderen Seite steht der MCP-Server. Das ist das kleine Programm, das mit deinem Home Assistant spricht. Beide tauschen Daten in einem standardisierten Format aus, und die KI kann über diesen Kanal Werkzeuge aufrufen.

2.2 Warum ein eigener MCP-Server nötig ist

Vielleicht fragst du dich, warum Claude nicht einfach direkt mit deinem Home Assistant spricht. Die Antwort ist simpel: Es fehlt der Kontext. Claude weiß nicht, wie deine Home Assistant API aussieht, welche Entitäten du hast und welche Befehle erlaubt sind. Genau diese Brücke baut der MCP-Server.

Konkret übersetzt der MCP-Server zwei Welten ineinander. Auf der einen Seite spricht er das MCP-Protokoll, das Claude versteht. Auf der anderen Seite spricht er die REST-API von Home Assistant. Du musst dich um diese Übersetzung nicht kümmern – sie passiert vollkommen automatisch im Hintergrund.

Zusätzlich gibt der MCP-Server Claude eine Liste an verfügbaren Werkzeugen mit. Diese Liste enthält nicht nur die Namen der Tools, sondern auch eine genaue Beschreibung dessen, was sie tun. Genau das macht den Unterschied. Erst durch diese Beschreibung weiß Claude, wann er welches Werkzeug verwenden soll.

2.3 ha-mcp im Überblick

Es gibt mehrere MCP-Server für Home Assistant. Für diesen Guide habe ich mich bewusst für ha-mcp entschieden, da es aktuell der umfangreichste und am besten gepflegte Server ist. Er stammt von der Community-Organisation homeassistant-ai auf GitHub und steht unter einer offenen Lizenz.

Anders als der eingebaute MCP-Server von Home Assistant deckt ha-mcp deutlich mehr als nur die Gerätesteuerung ab. Wir reden hier von mehr als 80 verschiedenen Werkzeugen, die Claude zur Verfügung stehen. Dazu gehören neben dem klassischen Schalten von Lichtern unter anderem das Erstellen und Bearbeiten von Automationen, das Auslesen der Historie, das Verwalten von Helfern und Szenen und sogar das Auslesen von Logdateien zur Fehleranalyse.

> **Eingebaut vs. ha-mcp**
>
> Home Assistant hat seit der Version 2025.2 einen eigenen MCP-Server eingebaut. Dieser greift jedoch nur auf die Assist-API zu und fokussiert sich auf die reine Steuerung deiner exponierten Entitäten. ha-mcp bietet dagegen über 80 Werkzeuge und richtet sich an Power-User, die ihre komplette Verwaltung mit der KI erledigen wollen.

Damit du verstehst, was dahintersteckt: Jedes dieser Werkzeuge ist im Grunde nichts anderes als eine kleine Funktion, die der MCP-Server bei Bedarf an die API von Home Assistant weitergibt. So ist beispielsweise smart_entity_search ein Werkzeug, das nach Entitäten in deinem System sucht und sogar Tippfehler verzeiht. Oder ha_config_set_automation, mit dem Claude eine bestehende Automation umschreiben oder eine neue anlegen kann.

Du musst dir diese Werkzeuge nicht im Detail merken. Wichtig ist nur, dass du den Mechanismus verstehst: Sobald ha-mcp läuft, hat Claude einen riesigen Werkzeugkasten zur Hand, mit dem er praktisch jeden Aspekt deines Smart Homes anfassen kann. Das macht das System mächtig – und genau deshalb müssen wir später besonders auf die Sicherheit achten.

3. Was ist uvx? – Der Werkzeugkasten der modernen Python-Welt

uvx ist der Befehl, mit dem wir später ha-mcp tatsächlich starten werden. Bevor wir ihn in unsere Konfiguration eintragen, möchte ich dir erklären, was sich hinter diesem unscheinbaren Drei-Buchstaben-Befehl verbirgt. Wenn du das verstanden hast, wird dir das Setup an mehreren Stellen plötzlich logisch erscheinen.

3.1 uv und uvx in einfachen Worten

uv ist ein vergleichsweise neuer Paketmanager für die Programmiersprache Python. Entwickelt wurde er von der Firma Astral, die auch für den beliebten Linter Ruff verantwortlich ist. Geschrieben ist uv in Rust, was ihn extrem schnell macht. In Tests installiert uv Python-Pakete oftmals zehn- bis hundertmal schneller als das altbekannte pip.

uvx wiederum ist ein Werkzeug, das mit uv geliefert wird. Es ist eine Kurzform für uv tool run und dient genau einem Zweck: Es führt ein Python-Programm aus, ohne dass du es vorher klassisch installieren musst. Du sagst uvx einfach, welches Paket du ausführen willst, und der Rest passiert automatisch im Hintergrund.

Konkret bedeutet das: Wenn wir in unserer Konfiguration später uvx ha-mcp@latest schreiben, sucht uvx das Paket ha-mcp in der öffentlichen Python-Paketdatenbank PyPI, lädt es in der neuesten Version herunter, packt es in eine eigene kleine Umgebung und startet es. Du selbst musst nichts davon tun. Es läuft komplett automatisch ab.

3.2 Wie uvx den MCP-Server startet

Lass uns das Ganze einmal Schritt für Schritt nachvollziehen. Wenn Claude Desktop seinen MCP-Server hochfahren will, schaut die Anwendung zuerst in die Konfigurationsdatei. Dort findet sie den Eintrag command: uvx und args: ["ha-mcp@latest"]. Das ist die Anweisung.

Im nächsten Schritt ruft Claude Desktop also den Befehl uvx ha-mcp@latest auf. uvx prüft daraufhin, ob das Paket ha-mcp in der neuesten Version bereits in seinem Cache liegt. Ist das nicht der Fall, lädt uvx das Paket von PyPI nach und richtet eine isolierte Umgebung dafür ein. Anschließend startet es das Programm und gibt die Kontrolle an Claude Desktop zurück.

Damit wird auch klar, warum der erste Start manchmal ein paar Sekunden länger dauert. Beim allerersten Aufruf muss uvx das Paket herunterladen, das ein paar Megabyte groß ist. Bei jedem weiteren Start zieht uvx das Paket einfach aus dem Cache und der Start geht in Sekundenbruchteilen.

> **Was bedeutet @latest?**
>
> Das Suffix @latest sagt uvx, dass es immer die neueste verfügbare Version des Pakets verwenden soll. Wenn du eine bestimmte Version festnageln willst, kannst du sie auch direkt angeben, zum

Beispiel ha-mcp@7.0.0. Für den Anfang ist @latest aber genau richtig, weil wir ohne extra Aufwand stets die aktuellste Version bekommen.

3.3 Warum uvx besser ist als pip install

Für alle, die schon mal mit Python gearbeitet haben: Wir hätten ha-mcp theoretisch auch über pip install ha-mcp klassisch installieren können. Warum gehen wir den Umweg über uvx? Dafür gibt es mehrere gute Gründe.

Erstens: uvx kapselt das Paket in einer eigenen Umgebung. Es kommt also nicht zu Konflikten mit anderen Python-Paketen, die du möglicherweise auf deinem System hast. Gerade Einsteiger wissen oft nicht, dass die globale Python-Installation auf Windows ein heikles Konstrukt sein kann. Mit uvx umgehen wir dieses Problem komplett.

Zweitens: uvx kann verschiedene Python-Versionen automatisch verwalten. ha-mcp benötigt mindestens Python 3.13. Wenn dein System eine ältere Version installiert hat, kümmert sich uvx im Hintergrund darum, die richtige Python-Version bereitzustellen. Du musst dich um diesen Aspekt nicht selbst kümmern.

Drittens: uvx hält das System sauber. Da nichts dauerhaft installiert wird, kannst du das Paket jederzeit wieder vergessen. Es liegen keine verstreuten Dateien herum und keine Registry-Einträge, die du eines Tages mühsam suchen musst.

Viertens: uvx ist deutlich schneller als pip. Für den alltäglichen Einsatz mag das egal sein. Bei der Installation größerer Pakete spürst du den Unterschied jedoch deutlich.

uvx zusammengefasst

uvx ist ein Werkzeug aus dem Hause Astral, das Python-Programme ausführt, ohne sie klassisch zu installieren. Es lädt das Paket in den Cache, kapselt es in einer eigenen Umgebung und startet das Programm. Für unseren Anwendungsfall ist es die sauberste und einfachste Möglichkeit, ha-mcp zu betreiben.

3.4 uvx und npx im Vergleich

Wenn du dich in der MCP-Welt umsiehst, wirst du schnell auf eine Frage stoßen, die viele Einsteiger irritiert. Manche MCP-Server starten mit uvx, andere mit npx, und wieder andere brauchen einen ganz anderen Befehl. Dahinter steckt aber kein Geheimnis, sondern eine sehr simple Logik. Lass uns diese gemeinsam durchgehen, damit du in Zukunft jeden MCP-Server sofort einordnen kannst.

Der Grund liegt schlichtweg in der Programmiersprache, in der ein MCP-Server geschrieben wurde. Genau wie es verschiedene Sprachen für Menschen gibt, gibt es auch verschiedene Programmiersprachen für Computer. Jede dieser Sprachen bringt ihr eigenes Ökosystem mit, also ihre eigenen Werkzeuge, Bibliotheken und Paketverwalter. Wer einen MCP-Server schreiben will, sucht sich die Sprache aus, mit der er am liebsten arbeitet. Und je nach Sprache ergibt sich dann automatisch das passende Werkzeug, um den Server zu starten.

Die beiden mit Abstand verbreitetsten Welten sind Python und Node.js. Python ist seit Jahrzehnten ein Liebling in Wissenschaft, Datenanalyse und neuerdings KI. Node.js ist die Plattform, auf der JavaScript außerhalb des Browsers läuft, und vor allem im Web-Umfeld zu Hause. Beide Welten haben ihre Stärken, und beide Welten bringen einen eigenen Werkzeugkasten mit, um Programme einfach zu starten.

In der Python-Welt heißt dieses Werkzeug uvx. Es lädt sich ein Paket von der offiziellen Python-Paketdatenbank PyPI, packt es in eine isolierte Umgebung und führt es aus. Genau das haben wir uns in den vorherigen Abschnitten angesehen.

In der Node.js-Welt heißt das vergleichbare Werkzeug npx. Es macht im Grunde dasselbe, nur eben für JavaScript-Programme. npx lädt ein Paket aus der npm Registry, der offiziellen Node.js-Paketdatenbank, und führt es ohne dauerhafte Installation aus. Auch hier passiert alles im Hintergrund, ohne dass du dich um Pfade oder Abhängigkeiten kümmern musst.

Wenn du in der Konfiguration eines MCP-Servers also command: uvx siehst, weißt du jetzt: Dieser Server ist in Python geschrieben. Steht dort command: npx, ist er in JavaScript geschrieben. Es gibt vereinzelt auch andere Varianten, etwa wenn jemand seinen Server in Go oder Rust geschrieben hat. Dann rufst du einfach eine ausführbare Datei direkt auf. Das sind aber Ausnahmen.

Eine kleine Übersicht

Python-MCPs werden mit uvx gestartet und nutzen PyPI als Paketquelle. Beispiele sind ha-mcp, das wir in diesem Guide verwenden, oder zahlreiche Datenanalyse-MCPs.

JavaScript-MCPs werden mit npx gestartet und nutzen die npm Registry. Beispiele sind der offizielle Filesystem-Server von Anthropic, der GitHub-MCP-Server oder verschiedene Slack-Integrationen.

Beide Tools verfolgen denselben Ansatz: kein dauerhaftes Installieren, kein Verschmutzen deines Systems, einfach nur Ausführen auf Abruf.

Für dich als Nutzer hat das vor allem eine praktische Konsequenz: Wenn du einen neuen MCP-Server einbinden willst, der mit npx startet, brauchst du dafür Node.js auf deinem System. Genauso wie du für uvx das Werkzeug uv installiert hast. Für unseren Guide ist das nicht relevant, weil ha-mcp ausschließlich auf uvx setzt. Sobald du aber deinen Werkzeugkasten erweitern willst, ist Node.js ein häufig benötigter Begleiter.

Node.js installierst du übrigens ähnlich einfach wie uv. Unter Windows reicht der Befehl winget install -e --id OpenJS.NodeJS.LTS in PowerShell. Damit bekommst du sowohl node selbst als auch npm und npx in einem Aufwasch. Nach einem Neustart von PowerShell sind alle drei Befehle verfügbar.

Warum gibt es nicht einen universellen Runner?

Diese Frage ist berechtigt. Tatsächlich gibt es einige Bestrebungen, MCP-Server unabhängig von der Programmiersprache zu starten, etwa über Docker. Solange MCP aber noch jung ist, hält sich jede Sprache an ihre eigenen Konventionen. Mit der Zeit wird sich das vermutlich angleichen. Bis dahin musst du nur wissen: uvx ist Python, npx ist JavaScript, alles andere ist die Ausnahme.