Model Context Protocol (MCP) – Grundlagen und Einsatz

Werkzeuge und Datenquellen standardisiert für KI-Anwendungen bereitstellen

01 INTRO

Model Context Protocol (MCP) definiert eine standardisierte Schnittstelle, über die KI-Anwendungen auf Werkzeuge, Datenquellen und weitere Fähigkeiten zugreifen können. Statt für jedes Modell und jede Anwendung proprietäre Integrationen zu entwickeln, können Funktionen über MCP-Server beschrieben und unterschiedlichen MCP-fähigen Clients bereitgestellt werden.

Der Kurs vermittelt MCP von der Architektur bis zur praktischen Implementierung. Die Teilnehmer verwenden vorhandene MCP-Server, entwickeln eigene Server und Tools und integrieren diese in KI-Anwendungen. Neben dem Protokoll selbst stehen Schnittstellendesign, Fehlerbehandlung, Berechtigungen, Security und Testbarkeit im Mittelpunkt. Ziel ist nicht lediglich ein funktionierender Demo-Server, sondern eine belastbare Integrationsschicht für reale Anwendungen.

DAUER

2 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Programmiererfahrung, vorzugsweise mit Python

02 TRAININGSPROGRAMM

Inhalte

Warum MCP?

- LLMs und externe Systeme
- proprietäre Tool-Integrationen
- wiederverwendbare Schnittstellen
- Modelle und Anwendungen entkoppeln
- MCP als Integrationsprotokoll
- typische Einsatzszenarien
- lokale und entfernte Systeme
- Grenzen und Nicht-Ziele von MCP

MCP einordnen

- MCP ist kein Agentenframework
- MCP ist kein LLM
- MCP ist keine allgemeine Workflow Engine
- MCP stellt Fähigkeiten und Kontext bereit
- Anwendung entscheidet über deren Nutzung
- Abgrenzung zu Function Calling
- Abgrenzung zu REST APIs
- Zusammenspiel der Komponenten

Architektur

- MCP Host
- MCP Client
- MCP Server
- Modell
- externe Systeme
- Kommunikationsbeziehungen
- Verantwortlichkeiten
- Trust Boundaries

Protokollgrundlagen

- Nachrichten und Operationen
- Requests und Responses
- Notifications
- Initialisierung
- Capability Negotiation
- Lifecycle
- Fehler
- Protokollversionen

Server und Clients

- Aufgaben eines MCP Servers
- Aufgaben eines MCP Clients
- Verbindungsaufbau
- verfügbare Fähigkeiten ermitteln
- mehrere Server
- Server unabhängig entwickeln
- Client-Unterstützung
- Interoperabilität

Tools

- ausführbare Fähigkeiten
- Tool Name
- Beschreibung
- Eingabeparameter
- strukturierte Schemas
- Rückgabewerte
- Fehler
- Seiteneffekte
- Tool Discovery

Gute Tools entwerfen

- klare Verantwortlichkeit
- verständliche Semantik
- geeignete Granularität
- kleine statt universeller Operationen
- strukturierte Parameter
- Validierung
- eindeutige Ergebnisse
- agentenfreundliches API-Design

Tool-Beschreibungen

- Beschreibung als Teil der Schnittstelle
- Zweck des Tools
- geeignete Einsatzfälle
- Parameter verständlich beschreiben
- Einschränkungen
- Auswirkungen
- Mehrdeutigkeiten vermeiden
- Tool-Auswahl durch gute Metadaten unterstützen

Resources

- Informationen statt Aktionen
- adressierbare Ressourcen
- URIs
- Dokumente
- Konfigurationen
- strukturierte Daten
- Resource Templates
- Tools und Resources sinnvoll unterscheiden

Prompts

- wiederverwendbare Prompt-Vorlagen
- Parameter
- vom Server angebotene Prompts
- typische Anwendungsfälle
- Prompts gegenüber Tools und Resources
- projektspezifische Arbeitsweisen
- Client-Unterstützung
- Prompts nicht mit Sicherheitsregeln verwechseln

Kontext über MCP bereitstellen

- externe Informationen
- Dokumentationen
- Projektdaten
- aktuelle Zustände
- strukturierte und unstrukturierte Informationen
- relevante Daten auswählen
- unnötigen Kontext vermeiden
- Datenherkunft nachvollziehbar halten

Transports

- lokale Kommunikation
- stdio
- entfernte MCP-Server
- HTTP-basierte Kommunikation
- Verbindungslebenszyklus
- Einsatzszenarien
- lokale und zentrale Server
- Auswirkungen auf Security und Betrieb

Einen MCP Server entwickeln

- Projektstruktur
- Server initialisieren
- Fähigkeiten registrieren
- erstes Tool
- Eingabeschema
- Implementierung
- Ergebnis zurückgeben
- Server lokal starten und testen

Bestehende Funktionen als Tools bereitstellen

- vorhandene Python-Funktionen
- interne Libraries
- Kommandozeilenwerkzeuge
- APIs
- Datenbankzugriffe
- bestehende Geschäftslogik
- Wrapper statt Neuentwicklung
- Verantwortlichkeiten sauber trennen

Externe APIs anbinden

- API Client und MCP Tool trennen
- Authentifizierung
- Parameter
- Fehler
- Timeouts
- Rate Limits
- externe Antworten validieren
- API-Details vor dem Modell kapseln

Datenbanken anbinden

- lesende Abfragen
- strukturierte Ergebnisse
- parametrisierte Operationen
- Schreibzugriffe
- Transaktionen
- Datenbankrechte
- Query-Tools gegenüber Universal-SQL
- minimale benötigte Fähigkeiten

Dateisystem und Shell

- Dateien lesen
- Dateien schreiben
- Verzeichnisse begrenzen
- Kommandos ausführen
- universelle Shell als mächtige Schnittstelle
- erlaubte Operationen begrenzen
- Pfade validieren
- Auswirkungen auf die Sicherheitsarchitektur

Fehlerbehandlung

- ungültige Eingaben
- erwartbare fachliche Fehler
- technische Fehler
- externe Systeme nicht erreichbar
- Timeouts
- verständliche Fehlermeldungen
- Fehler für Client und Modell nutzbar machen
- interne Details nicht unnötig offenlegen

Strukturierte Ergebnisse

- Text
- strukturierte Daten
- maschinenlesbare Ergebnisse
- Ergebnis-Schemas
- Metadaten
- große Ergebnismengen
- relevante Informationen zurückgeben
- stabile Schnittstellen

Stateful und stateless Tools

- zustandslose Operationen
- Session-Zustand
- externe Zustände
- konkurrierende Aufrufe
- Ressourcenlebensdauer
- Wiederholbarkeit
- idempotente Operationen
- Zustand bewusst gestalten

Berechtigungen

- MCP-Verbindung bedeutet nicht unbegrenzten Zugriff
- Identität
- Authentifizierung
- Autorisierung
- Lese- und Schreibrechte
- Tool-spezifische Rechte
- externe Systeme
- Least Privilege

Secrets und Credentials

- API Keys
- Tokens
- Datenbankzugänge
- Credentials nicht als Modellkontext
- Secret Stores
- Umgebungsvariablen
- getrennte Identitäten
- Rotation und Widerruf

Trust Boundaries

- MCP Client
- MCP Server
- Modell
- Tool-Implementierung
- externe Systeme
- zurückgelieferte Daten
- lokale und entfernte Server
- unterschiedliche Vertrauensniveaus

MCP und Prompt Injection

- Tool-Ergebnisse als untrusted Input
- externe Dokumente
- Webseiten
- Datenbanken
- indirekte Prompt Injection
- Daten und Instruktionen unterscheiden
- Serverbeschreibung nicht als Sicherheitsgrenze
- technische Kontrolle außerhalb des Modells

Gefährliche Tool-Designs

- beliebige Shell-Kommandos
- beliebige SQL-Befehle
- unbeschränkter Dateizugriff
- universelle HTTP Requests
- weitreichende Cloud-Rechte
- kombinierbare Einzelrechte
- unerwartete Seiteneffekte
- Capability Design als Sicherheitsentscheidung

Defensive Tool-Implementierung

- Eingaben validieren
- Pfade begrenzen
- erlaubte Werte
- serverseitige Berechtigungsprüfung
- sichere Defaults
- Ausgaben begrenzen
- kritische Aktionen gesondert behandeln
- Modellinstruktionen nicht als Zugriffskontrolle verwenden

Read und Write trennen

- reine Informationswerkzeuge
- verändernde Operationen
- unterschiedliche Risiken
- getrennte Tools
- explizite Parameter
- Bestätigung kritischer Aktionen
- reversible Operationen
- klare Auswirkungen

MCP Server testen

- Tool-Implementierung unabhängig vom LLM testen
- Unit Tests
- Schema-Validierung
- positive Fälle
- negative Fälle
- Fehlerfälle
- externe Systeme mocken
- reproduzierbare Tests

Integrationstests

- Client und Server
- Capability Discovery
- Tool-Aufrufe
- Resources
- strukturierte Ergebnisse
- Fehlerbehandlung
- mehrere Tools
- tatsächliches Modell erst als zusätzliche Ebene

Debugging

- Server startet nicht
- Client findet Server nicht
- Tool wird nicht angeboten
- falsche Parameter
- Schema-Probleme
- Tool-Ausführung schlägt fehl
- Logging
- Protokoll- und Anwendungsebene unterscheiden

Observability

- Verbindungen
- Tool-Aufrufe
- Laufzeiten
- Fehler
- externe Zugriffe
- Ergebnisgrößen
- Audit-relevante Aktionen
- sensible Daten nicht unkontrolliert protokollieren

MCP Server im Team

- Tool-Schnittstellen dokumentieren
- Versionierung
- gemeinsame Konfiguration
- Entwicklungs- und Produktionsumgebung
- Ownership
- Änderungen an Tools
- Rückwärtskompatibilität
- Tests als Schnittstellenvertrag

Lokale und zentrale MCP-Server

- Entwicklerwerkzeuge lokal
- gemeinsame Unternehmensdienste
- Remote Server
- unterschiedliche Benutzer
- zentrale Berechtigungen
- Skalierung
- Netzwerkgrenzen
- Auswahl nach Anwendungsfall

MCP in bestehenden Anwendungen

- vorhandene Services wiederverwenden
- MCP als Adapter
- Geschäftslogik nicht in das Protokoll verschieben
- API und MCP parallel anbieten
- Client-spezifische Integration reduzieren
- bestehende Authentifizierung
- Monitoring
- schrittweise Einführung

MCP und Agenten

- Agent verwendet MCP Tools
- Tool Discovery
- Tool-Auswahl durch das Modell
- mehrere MCP-Server
- Agenten erhalten unterschiedliche Toolsets
- Berechtigungen je Agent
- MCP ermöglicht Aktionen, steuert aber nicht den Agenten
- Agentenarchitektur bleibt eigene Verantwortung

MCP und Coding Agents

- Entwicklungswerkzeuge bereitstellen
- Repository-Informationen
- Testautomatisierung
- Browser-Automatisierung
- Datenbanken
- interne Entwicklerwerkzeuge
- projektspezifische Tools
- Coding Agent und MCP Server sauber trennen

MCP-Architektur entwerfen

- benötigte Fähigkeiten identifizieren
- Tools und Resources unterscheiden
- Tool-Grenzen festlegen
- externe Systeme bestimmen
- Berechtigungen definieren
- Trust Boundaries
- Teststrategie
- Betriebsmodell

Vom Prototyp zum produktiven MCP Server

- minimalen Server entwickeln
- Schnittstellen stabilisieren
- Validierung ergänzen
- Fehlerbehandlung
- Berechtigungen
- Tests
- Logging und Monitoring
- Deployment und Betrieb

Praktische Übungen

- vorhandenen MCP-Server einbinden
- angebotene Fähigkeiten untersuchen
- eigenen MCP-Server entwickeln
- Tool mit strukturierten Parametern implementieren
- Resource bereitstellen
- bestehende Funktion als Tool kapseln
- externe API anbinden
- Fehlerbehandlung ergänzen
- Tool gegen ungültige Eingaben absichern
- Server unabhängig vom LLM testen
- MCP-Server mit einem AI-Client verbinden
- Sicherheitsgrenzen einer Tool-Schnittstelle analysieren
- MCP-Architektur für einen realen Anwendungsfall entwerfen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	2 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams
Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit Entwicklungsumgebung, Python und einem MCP-fähigen Client

> **Inhouse-Schulungen**
> Vorhandene APIs, Datenbanken, Entwicklungswerkzeuge oder interne Dienste können als Grundlage für die Entwicklung eigener MCP-Schnittstellen verwendet werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Softwareentwickler, AI Engineers, Softwarearchitekten und technische Consultants, die externe Werkzeuge und Datenquellen standardisiert für KI-Anwendungen und AI Agents bereitstellen möchten.

Er eignet sich sowohl für Entwickler, die vorhandene MCP-Server einsetzen wollen, als auch für Teams, die eigene interne Systeme, APIs oder Entwicklungswerkzeuge über MCP verfügbar machen möchten.

> **Voraussetzungen**
> Praktische Programmiererfahrung wird vorausgesetzt; für die Übungen sind Python-Grundkenntnisse sinnvoll. Grundkenntnisse über generative KI und Sprachmodelle sind hilfreich. Erfahrung mit AI Agents oder bestimmten MCP-fähigen Produkten ist nicht erforderlich.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- Zweck und Einsatzbereich von MCP erklären
- MCP von Agentenframeworks, Function Calling und klassischen APIs abgrenzen
- Host, Client und Server unterscheiden
- Tools, Resources und Prompts sinnvoll einsetzen
- eigene MCP-Server entwickeln
- vorhandene Funktionen und APIs als MCP Tools bereitstellen
- agentenfreundliche Tool-Schnittstellen entwerfen
- strukturierte Ein- und Ausgaben definieren
- Fehler robust behandeln
- lokale und entfernte MCP-Szenarien einordnen
- Berechtigungen nach dem Least-Privilege-Prinzip gestalten
- Secrets und Credentials vom Modellkontext trennen
- Trust Boundaries einer MCP-Architektur bestimmen
- Prompt-Injection- und Tool-Risiken berücksichtigen
- MCP-Server unabhängig vom Sprachmodell testen
- Integrationstests und Observability aufbauen
- MCP in vorhandene Anwendungen und Architekturen integrieren
- einen MCP-Server vom Prototyp zu einer belastbaren Integration weiterentwickeln

06 FAQ

Häufige Fragen

Was unterscheidet diesen Kurs von AI Agents?

Der Agentenkurs behandelt die Architektur vollständiger agentischer Systeme. MCP ist dort eine von mehreren Möglichkeiten, Agenten mit externen Fähigkeiten auszustatten.

Dieser Kurs behandelt MCP selbst. Protokollarchitektur, Server, Clients, Tools, Resources, Transports, Schnittstellendesign, Security, Testing und Betrieb stehen hier im Mittelpunkt.

Ist MCP ein Agentenframework?

Nein. MCP stellt standardisierte Schnittstellen für Kontext und Fähigkeiten bereit. Wie ein Agent plant, entscheidet, seinen Zustand verwaltet oder mehrere Arbeitsschritte orchestriert, wird durch MCP nicht festgelegt.

Ersetzt MCP REST APIs?

Nein. Häufig kapselt ein MCP-Server gerade vorhandene APIs und stellt daraus geeignete Fähigkeiten für AI-Anwendungen bereit. Die bestehende Geschäftslogik muss deshalb nicht für MCP neu entwickelt werden.

Werden eigene MCP-Server programmiert?

Ja. Die Teilnehmer entwickeln im Kurs eigene Server und stellen mehrere Fähigkeiten darüber bereit.

Werden Tools und Resources behandelt?

Ja. Ein wichtiger Teil des Kurses ist gerade die Entscheidung, ob etwas als ausführbare Operation, als Informationsquelle oder auf andere Weise bereitgestellt werden sollte.

Wird Security behandelt?

Ja. MCP kann einem Modell Zugriff auf reale Systeme und Aktionen ermöglichen. Authentifizierung, Autorisierung, Least Privilege, Secrets, Trust Boundaries und sichere Tool-Schnittstellen gehören deshalb zum Kern des Kurses.

Werden MCP-Server getestet?

Ja. Ein MCP-Tool ist letztlich eine Softwareschnittstelle und sollte auch so behandelt werden. Tool-Implementierungen, Schemas und Fehlerfälle werden deshalb zunächst deterministisch getestet, bevor ein LLM in den Test einbezogen wird.

Wird ein bestimmter AI-Client verwendet?

Ein MCP-fähiger Client kann für die praktischen Übungen verwendet werden. Die Bedienung eines bestimmten Produkts ist jedoch nicht Gegenstand dieses Kurses; dafür sind die jeweiligen Produktschulungen vorgesehen.

Muss ich den Agentenkurs vorher besucht haben?

Nein. Grundlegende Kenntnisse zu generativer KI und Programmiererfahrung sind ausreichend. Agentenarchitektur wird nur so weit erläutert, wie sie zum Verständnis der Rolle von MCP erforderlich ist.

Warum dauert der Kurs zwei Tage?

Der Kurs konzentriert sich auf MCP als Integrationsschicht und nicht auf die Entwicklung vollständiger Agentensysteme. Zwei Tage bieten ausreichend Raum, die Architektur zu verstehen, eigene Server und Tools praktisch zu entwickeln und die wesentlichen Aspekte von Security und Testing zu behandeln.

Kann der Kurs auf unsere eigenen Systeme zugeschnitten werden?

Ja. Bei Inhouse-Schulungen können vorhandene APIs, Datenbanken, Entwicklungswerkzeuge oder interne Dienste als Grundlage für die Entwicklung eigener MCP-Schnittstellen verwendet werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/model-context-protocol/