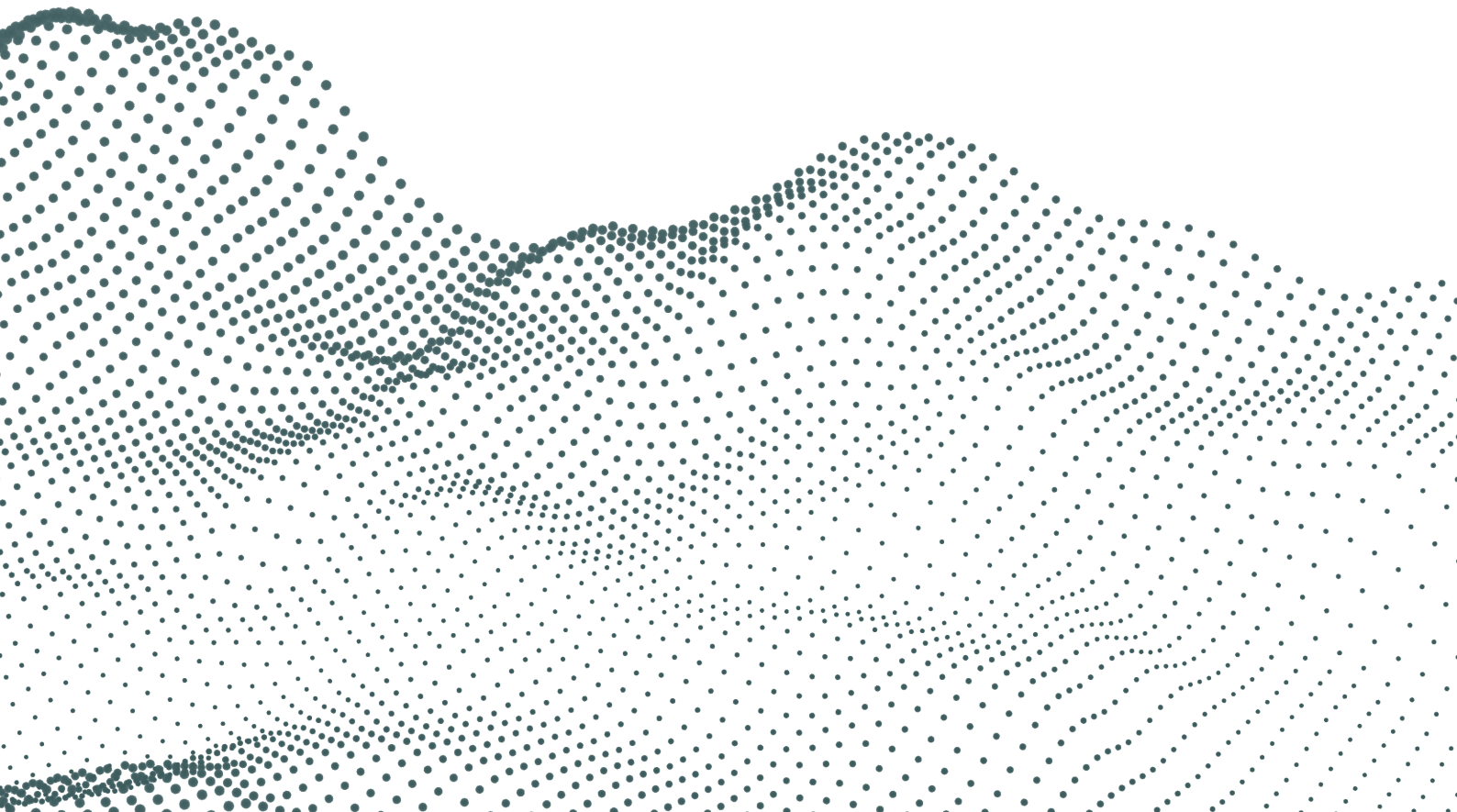
mohr stade

MCP-Server in der Analyse von Analytics-Daten

Infrastruktur und Remote Setup mit Google Cloud Run

Whitepaper



Inhalt

Veröffentlichung dieses Whitepapers: Dezember 2025

1. Einleitung

Self-Service-Analytics galt lange als Hoffnung, Mitarbeitern den eigenständigen Zugriff auf Unternehmensdaten und deren Analyse zu ermöglichen. Ziel war es, mehr Verständnis für Prozesse, Abläufe und Geschäftszusammenhänge zu schaffen. In der Praxis blieb dieser Ansatz jedoch häufig hinter den Erwartungen zurück. Fragmentierte Tools, komplexe Schnittstellen und fehlendes Know-how machten es Fachabteilungen schwer, unabhängig von Data-Teams wertvolle Erkenntnisse zu gewinnen.

Mit dem Aufkommen von Large Language Models (LLMs) wie ChatGPT, Claude, Gemini oder Copilot rückt diese Idee nun wieder in greifbare Nähe. LLMs ermöglichen es, Daten über natürliche Sprache zu analysieren, ohne über technisches Vorwissen zu Tools oder Anwendungen zu verfügen. Damit wird es für Unternehmen denkbar, Analyse- und Performance-Daten direkt über Chat-Interfaces bereitzustellen und so den Self-Service-Gedanken neu zu beleben.

Eine zentrale Rolle spielt dabei das Model Context Protocol (MCP). Das MCP ist ein offener technologischer Standard, über den LLMs sicher und strukturiert mit Datenquellen wie Google Analytics, BigQuery, Azure Data Lake, AWS Redshift oder anderen Cloud-Datenbanken und BI-Systemen interagieren können. MCP fungiert als Brücke zwischen KI-Modellen und Datenquellen und ermöglicht reproduzierbare und kontextbasierte Analysen. Auch Anbieter wie Google integrieren MCP in ihre bestehenden Systeme und stellen damit erstmals standardisierte, LLM-kompatible Schnittstellen für Analyse- und Reporting-Funktionen bereit.

Dieses Whitepaper beschreibt die Grundlagen und Architektur des MCP und veranschaulicht am Beispiel des Google Analytics MCP Servers auf Cloud Run, wie diese Technologie in der Praxis implementiert und genutzt werden kann.

„Zukünftig wird MCP eine zentrale Rolle dabei spielen, Analytics-Daten kontextbewusst in KI-gestützten Anwendungen einzubetten. Mit wachsender Tool-Kompatibilität und neuen Frameworks wie FastMCP wird die Entwicklung eigener MCP-Server und Tools zunehmend einfacher und stabiler. Für Data Engineers, Data Analysts und Architects eröffnet sich damit ein verlässlicher und sicherer Weg, bestehende Analytics-Infrastrukturen in das Zeitalter kontextbasierter Datenverarbeitung mit LLMs zu führen.“

Moritz Bauer
Director Marketing Technology bei mohrstade

2. Definition, Funktionsweise und Grenzen

Das Model Context Protocol (MCP) ist ein offener Standard von Anthropic. Es definiert, wie Modelle (also Large Language Models oder KI-Agenten) sicher mit Tools, APIs und Datenquellen interagieren können. Als serverseitige Verarbeitungsschicht (*Processing Layer*) fungiert MCP dabei als standardisierte Schnittstelle zwischen dem Modell und externen Systemen wie bspw. BigQuery, Azure Data Lake oder Amazon Redshift. Ein MCP-Server implementiert konkrete Tools (z. B. *ga4_get_report, bq_query, realtime_overview*), die über das Protokoll aufgerufen werden können.

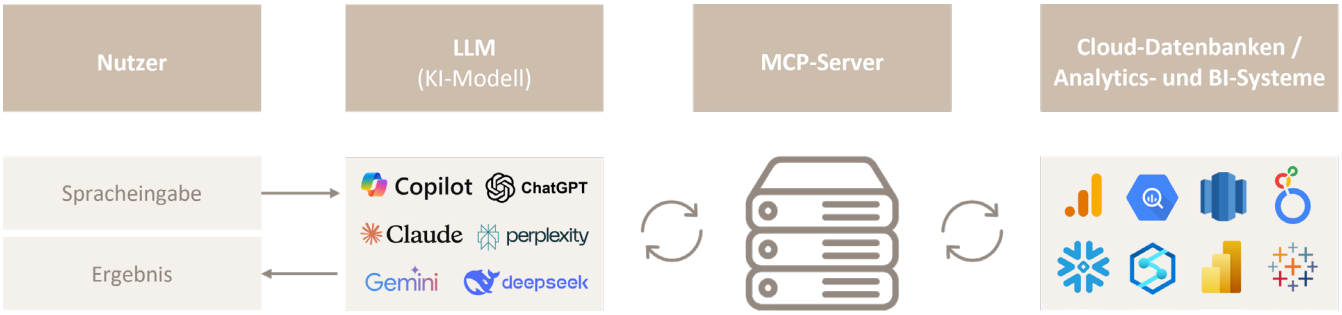


Abbildung 1: Grundlegende Funktionsweise eines MCP-Servers (Quelle: eigene Darstellung)

> „The MCP server exposes tools that allow models to access and analyze Analytics data in a structured, secure and regionally controlled environment.“ (Quelle: Google - MCP GitHub Repository)

Etwas spezifischer bedeutet das:

1. Der Nutzer gibt einen Prompt ein, worauf der MCP-Client (z. B. ChatGPT, Copilot oder Gemini) beim MCP-Server verfügbare Tools abruft.
2. Diese Tools sind definierte Funktionsschnittstellen zu Datenbanken, Analytics- und BI-Systemen (Looker, PowerBI, Tableau etc.) oder verwandten APIs (z. B. BigQuery, Azure Data Lake, AWS Redshift).
3. Der Server verarbeitet die Anfrage, authentifiziert sich über einen Service Account oder über OAuth mit Benutzerrechten und ruft die jeweilige API auf.
4. Die Ergebnisse werden strukturiert (z. B. JSON) an den Client zurückgegeben, ohne dass dieser direkten API-Zugriff benötigt, und dem Nutzer in natürlicher Sprache zur Verfügung gestellt.

Die Kommunikation erfolgt über standardisierte Transportprotokolle. In lokalen Entwicklungsumgebungen wird häufig stdio verwendet, während produktive Setups meist auf HTTP-streamable oder vergleichbare Mechanismen setzen, um parallele Verbindungen und einen cloudbasierten Betrieb zu ermöglichen. Der MCP-Server kann in jeder modernen Cloud-Infrastruktur (z. B. AWS, Azure, Snowflake oder Google Cloud) betrieben werden. Allerdings sollte sichergestellt werden, dass die Datenverarbeitung innerhalb definierter Regionen und Compliance-Richtlinien erfolgt. Der entscheidende Unterschied zu klassischen serverseitigen Tracking-Ansätzen (z. B. Tagging-Servern oder Consent-Proxys) besteht darin, dass MCP keine Tracking-Events oder Cookies verarbeitet. Es stellt lediglich strukturierte Schnittstellen für bestehende Tools und Datenquellen bereit. Large Language Models (LLMs) können diese Tools standardisiert für Abfragen und Analysen nutzen, ohne direkten Zugriff auf die Rohdaten zu benötigen. Der MCP-Server dient somit nicht als "Tracking-Proxy", sondern als neutrale Integrationsschicht zwischen KI-Modellen und bestehenden Analytics- oder Datensystemen.

2.1 Mögliche Vorteile und Chancen der Technologie

Das MCP eröffnet eine neue technologische Ebene für die Integration von Google Analytics 4 (GA4) und BigQuery in datengetriebene Anwendungen und KI-gestützte Analysen.

Durch den standardisierten Tool-Zugriff über MCP können Unternehmen und Entwickler Daten sicher, kontrolliert und automatisiert in modellbasierte Workflows integrieren.

Standardisierung

MCP definiert ein einheitliches Protokoll, über das Modelle (z. B. Gemini) auf Tools und APIs zugreifen können. Dadurch entfällt die bisher notwendige individuelle API-Integration pro Datenquelle.

Sicherheit und Zugriffskontrolle

Der Zugriff auf Analytics- oder BigQuery-Daten erfolgt ausschließlich über den MCP-Server, der in einer EU-Region (z. B. europe-west4) betrieben wird und durch IAM-Policies sowie Secret Manager abgesichert wird. Alternativ kann die Authentifizierung auch über OAuth erfolgen. In diesem Fall erhält der MCP-Server nur die Berechtigung, die der jeweilige Nutzer beim Zugriff explizit freigibt. Dadurch lassen sich Zugriffsrechte granularer verwalten und auf Projektebene einschränken.

Datenschutz und regionales Processing

Der MCP-Server selbst kann innerhalb der EU Data Boundary betrieben werden, sodass die Datenverarbeitung regional erfolgt. Die Abfrage der Analytics-Daten erfolgt hingegen über die globalen GA4-API-Endpunkte (*analyticsdata.googleapis.com*). Damit kann die Verarbeitung zwar regional kontrolliert, jedoch ein vollständiger "EU-Only-Datenfluss" nicht garantiert werden. Demnach reduziert das MCP Übermittlungsrisiken, ersetzt aber keine vertraglichen Datenschutzmaßnahmen gemäß DSGVO.

Schnittstellen für LLMs

Modelle wie Gemini oder Claude können durch MCP verschiedene Tools automatisch entdecken, aufrufen und deren Parameter interpretieren. Das ermöglicht präzise, reproduzierbare und auditierbare Datenabfragen ohne direkten Zugriff auf Rohdaten.

Automatisierung von Analyse- und Reporting-Workflows

Über Tools wie *ga4_get_report* oder *bq_query* lassen sich Berichte generieren, aggregieren und in natürlicher Sprache übersetzen. Das schafft neue Möglichkeiten für die automatisierte Erstellung von Reportings/Dashboards und somit eine Entscheidungsunterstützung durch das LLM.

2.2 Grenzen und Kritik

Trotz seiner technologischen Stärke befindet sich das MCP noch in einer frühen Phase der praktischen Adaption. Insbesondere im Zusammenspiel von LLMs, Analytics-Daten und regulatorischen Anforderungen ergeben sich noch einige Einschränkungen und offene Fragen.

Begrenzte Tool-Verfügbarkeit

Aktuell stellt der Google Analytics MCP-Server nur wenige vordefinierte Tools bereit (z. B. GA4 Data API, BigQuery Export). Komplexe oder kombinierte Abfragen erfordern eigene Tool-Definitionen oder Anpassungen im Code.

Erhöhter technischer Aufwand für Eigenbetrieb

Der Betrieb eines MCP-Servers auf Cloud Run erfordert Kenntnisse in Containern, IAM, Secrets und Logging. Für nicht-technische Teams kann die Implementierung zunächst komplex wirken.

Offene Governance-Fragen

Das Protokoll selbst wird von der Open-Source-Community (Anthropic/Google) weiterentwickelt. Es gibt noch keinen formellen Standardisierungsprozess (z. B. über IETF oder W3C). Langfristig kann sich die Spezifikation verändern, was regelmäßige Wartung erforderlich macht. Obwohl der Server die Datenverarbeitung lokalisiert, bleibt der nachgelagerte Modell-Kontext (z. B. Gemini-Session) ein potenzieller Risikofaktor.

„MCP belebt den Gedanken von Self-Service-Analytics neu. Es verbindet LLMs direkt mit Datenquellen und ermöglicht so auch Mitarbeitenden ohne technisches Know-how, Analysen einfach über natürliche Sprache durchzuführen.“

Parick Mohr
Co-Founder und Managing Partner bei mohrstade

3. MCP Use Case: Analyse von GA4-Daten mit Gemini

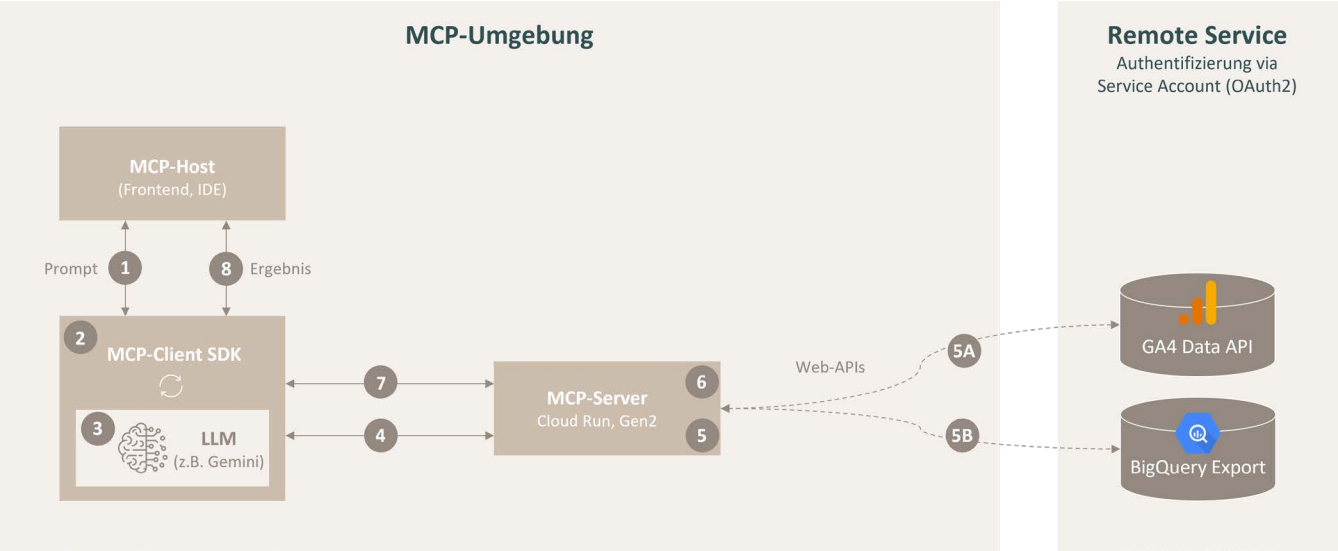


Abbildung 2: Vereinfachte-MCP-Infrastruktur mit Analytics Backend (Quelle: eigene Darstellung)

Die Abbildung zeigt, wie Large Language Models (LLMs) wie Gemini mithilfe des Model Context Protocols (MCP) auf unterschiedliche Datenquellen zugreifen können. In diesem Szenario fungiert MCP als standardisierte Integrationsschicht zwischen dem Modell und externen Systemen wie Google Analytics 4 (GA4) oder BigQuery. Ziel des Modells ist es, dass es in natürlicher Sprache formulierte Fragen (z.B. „Wie viele Sitzungen gab es in der vergangenen Woche?“) in strukturierte API-Abfragen übersetzen und die Ergebnisse anschließend interpretieren kann. Im Folgenden wird dieser Prozess einmal detailliert aufgeschlüsselt:

1. Anfrage durch den Nutzer	Im ersten Schritt gibt der Nutzer in einem LLM-basierten System (bspw. Claude, Gemini etc.) eine natürliche Sprachabfrage wie z. B. „Zeig mir die Conversions der letzten 7 Tage nach Quelle/Medium“ ein. Anschließend übergibt der Host den Prompt an den MCP-Client (client.py).
2. Verarbeitung durch den MCP-Client	Der Client hält die Verbindung zum LLM und kennt die verfügbaren Tools auf den registrierten MCP-Servern (Tool-Discovery). *Wichtig: Der Client enthält keine Secrets für GA4/BigQuery. Er ruft ausschließlich Tools an.*
3. Function Calling	Der Client übermittelt den Prompt, die Tool-Beschreibung (Function-Schema) und optionale Instruktionen an das Modell (z. B. „Nutze GA4-Tool für Metriken, BigQuery-Tool für Event-Daten“). Das Modell (Gemini) erzeugt anschließend einen function_call mit Tool-Name und Parametern (z. B. ga4_get_report, Property-ID, Zeitraum, Metriken, Dimensionen). Dieser strukturierte Call wird anschließend an den MCP-Client zurückgegeben.

4. Übergabe des funtion_call an den MCP-Server	Der Client sendet den Funktionsaufruf über den standardisierten Transport (HTTP-streamable) an den MCP-Server, der auf Google Cloud Run (Gen 2) betrieben wird.
5. Authentifizierung und API-Call	Der MCP-Server interpretiert anschließend diesen Request, prüft das Schema und entscheidet, welche Datenquelle verwendet werden soll. Für aggregierte Kennzahlen (z. B. Sitzungen, Nutzer, Conversions) nutzt er die GA4 Data API. Für detaillierte, eventbasierte Analysen (z.B. Funnel, Attributionspfade, Events) nutzt er den BigQuery Export der jeweiligen GA4-Property. Der MCP-Server kommuniziert über Web-APIs mit Google-Diensten. Dabei erfolgt die Authentifizierung über einen Google Service Account, dessen Zugangsdaten im Secret Manager oder als Umgebungsvariable hinterlegt sind. Der Endpunkt der GA4 Data API (*https://analyticsdata.googleapis.com*) liefert aggregierte Reports nach Metriken und Dimensionen (5A), während der BigQuery Export (*https://bigquery.googleapis.com*) Abfragen via SQL granular auf Event-Level ermöglicht (5B). Beide Endpunkte liegen innerhalb der Google Cloud EU Data Boundary, was für die GDPR-/DSGVO-Konformität entscheidend ist.
6. Datenverarbeitung und Formatierung	Nach Erhalt der Antwort von GA4 oder BigQuery übernimmt der Server: • das Parsing und Normalisieren der Daten (z. B. in JSON- oder Tabellenform) • optional die Aggregation oder Visualisierung (z. B. Diagramm-Spezifikationen für Gemini) • die Vorbereitung für den Rücktransport an den MCP-Client `{ "header": ["channel", "sessions", "conversions"], "rows": [ ["Organic Search", 5321, 94], ["Direct", 2410, 63], ["Paid Social", 1895, 48] ], "metadata": {"source": "GA4 Data API", "fetched_at": "2025-10-09T09:45:00Z"} }` *Beispiel einer vereinfachten GA4-Server-Antwort*
7. Rückgabe an den MCP-Client	Der MCP-Server sendet die verarbeiteten Events über den MCP-Protokollkanal zurück an den MCP-Client. Anschließend fügt der Client die Antwort in das Session-Objekt des LLM ein.
8. Generierung der Antwort durch Gemini	Im letzten Schritt interpretiert Gemini die strukturierten Daten und generiert daraus eine lesbare Antwort in natürlicher Sprache. Bezogen auf das vorherige Beispiel aus Schritt 1 könnte eine Antwort wie folgt aussehen: *„In den letzten 7 Tagen kamen 53% Ihrer Sitzungen über Organic Search, 24% über Direct und 18% über Paid Social. Die höchste Conversion Rate zeigte Paid Social mit 2,5% im Betrachtungszeitraum.“* Auf Wunsch kann Gemini auch automatisch ein Diagramm (z.B. vega-Spec oder Matplotlib-Plot) ausgeben, das direkt im Frontend gerendert werden kann.

3.1 Einrichtung des MCP-Servers

Wie bereits beschrieben, bildet der MCP-Server die technische Kernkomponente der Infrastruktur. Er stellt die Tools bereit, über LLMs wie Gemini auf externe Systeme wie Google Analytics 4 (GA4) oder BigQuery zugreifen können.
Für die Bereitstellung empfiehlt Google eine containerisierte Umgebung auf Cloud Run (Gen 2). Dadurch ist der Server vollständig serverlos skalierbar, versionierbar und über HTTPS erreichbar. Die Implementierung erfolgt typischerweise in Python, basierend auf dem offiziellen Paket *mcp* oder einem vergleichbaren Framework (z. B. FastMCP mit MCP-Integration). Jeder Server bedient dabei ein oder mehrere Tools, die über ein selbstbeschreibendes Schema (JSON) definiert werden.

Für den Betrieb des Servers sind folgende Grundvoraussetzungen zu schaffen:

- Docker Container (Dockerfile und requirements.txt)
- Deployment nach Cloud Run via Cloud Build oder GitHub Actions
- Authentifizierung über einen dedizierten Service Account mit entsprechenden IAM-Rollen
- Speicherung sensibler Daten (API-Keys, Tokens) im Secret Manager
- Aktiviertes Cloud Logging und Cloud Monitoring für Fehler- und Performance-Analysen

3.2 Einrichtung notwendiger Google Cloud Credentials

Damit der MCP-Server auf die Google APIs zugreifen kann, müssen geeignete Credentials eingerichtet werden:

1. Service Account

Der Server authentifiziert sich mit Application Default Credentials (ADC) über den Service Account. Folgende Rollen müssen hierfür über die Google Cloud Console im Service Account (z. B. *mcp-ga4-service@<projekt>.iam.gserviceaccount.com*) erstellt und zugewiesen werden:

Rolle	Beschreibung
roles/viewer	Grundlegende Leseberechtigung für API-Nutzung
roles/bigquery.dataViewer	Lesezugriff auf BigQuery-Tabellen (GA4-Export)
roles/secretmanager.secretAccessor	Zugriff auf Secrets im Secret Manager
roles/logging.logWriter	Schreiben von Logs in Cloud Logging
roles/run.invoker	Aufrufberechtigung für Cloud Run Service

2. Aktivierung der API

Anschließend erfolgt die Aktivierung über die Google Cloud Console unter API & Dienste > Bibliothek durch folgende APIs:

- Google Analytics Data API (analyticsdata.googleapis.com)
- BigQuery API (bigquery.googleapis.com)
- Secret Manager API (secretmanager.googleapis.com)
- Cloud Logging & Monitoring (optional für Auditing)

3. Secrets anlegen

Im nächsten Schritt müssen folgende API-Keys oder Token in Secret Manager gespeichert werden:

Secret Name	Beschreibung
GA4_SERVICE_KEY	JSON-Key des Service Accounts
MCP_CLIENT_TOKEN	Authentifizierungs-Token für den MCP-Client
BQ_PROJECT_ID	*Optional: Projekt-ID für BigQuery-Zugriffe anar*

4. Umgebungsvariablen (Environment Variables)

Beim Deployment des MCP-Servers müssen folgende Credentials als Umgebungsvariablen zur Verfügung stehen:

```
export GOOGLE_APPLICATION_CREDENTIALS="/secrets/ga4-service.json"
export GA4_PROPERTY_ID="123456789"
export BQ_PROJECT_ID="analytics-project"
```

In Cloud Run kann die Variable über den Secret Manager automatisch eingebunden werden:

```
gcloud run deploy mcp-server \
  --set-secrets "GA4_SERVICE_KEY=projects/123/secrets/GA4_SERVICE_KEY:latest" \
  --region=europe-west4 \
  --service-account=mcp-ga4-service@<projekt>.iam.gserviceaccount.com
```

5. Zugriffsbeschränkungen (Org Policies)

Zur Einhaltung der Datenschutz- und Compliance-Vorgaben sollten auf Organisationsebene folgende Policies aktiviert werden:

Policy	Zweck
constraints/gcp.resourceLocations	beschränkt Ressourcen auf EU-Regionen (z. B. europe-west4, europe-north1).
constraints/iam.allowedPolicyMemberDomains	limitiert IAM-Mitglieder auf vertrauenswürdige Domains.
constraints/run.allowedIngress	beschränkt eingehenden Traffic auf interne oder authentifizierte Quellen.
constraints/secretmanager.allowedLocations	stellt sicher, dass Secrets nur in EU-Regionen gespeichert werden.

6. Überwachung und Logging

Zur Nachvollziehbarkeit der Zugriffe sollte Cloud Logging aktiviert sein. Die Standardkonfiguration protokolliert Requests, Latenzen, Statuscodes und Tool-Aufrufe (ohne personenbezogene Daten).

3.3 Lokales Setup

Für Entwicklungs- und Testzwecke kann der MCP-Server auch lokal ausgeführt werden. Dies ermöglicht ein unkompliziertes Debugging mit Live-Logs und vermeidet Kosten für Cloud-Ressourcen. Daher ist das Setup ideal für Integrationstests vor dem Deployment auf Cloud Run.

1. Virtuelle Umgebung anlegen

```
python3 -m venv venv
source venv/bin/activate
pip install mcp google-analytics-data google-cloud-bigquery
```

2. Authentifizierung

Der lokale MCP-Server verwendet Application Default Credentials (ADC). Hierfür ist eine Anmeldung im jeweiligen Google-Konto oder Service Account notwendig:

```
gcloud auth application-default login
```

Alternativ kann ein Service Account JSON über eine Umgebungsvariable gesetzt werden:

```
export GOOGLE_APPLICATION_CREDENTIALS="/pfad/zum/service-account.json"
```

3. Lokale Konfiguration

Je nach gewünschtem Transport stehen zwei Varianten zur Verfügung.

```
python -m mcp.server --connection_type stdio
```

Variante A – stdio (empfohlen für lokale Tests)

```
uvicorn main:app --host 0.0.0.0 --port 8080
```

Variante B – HTTP (z. B. mit FastAPI + Uvicorn)

4.
Testen der Tools

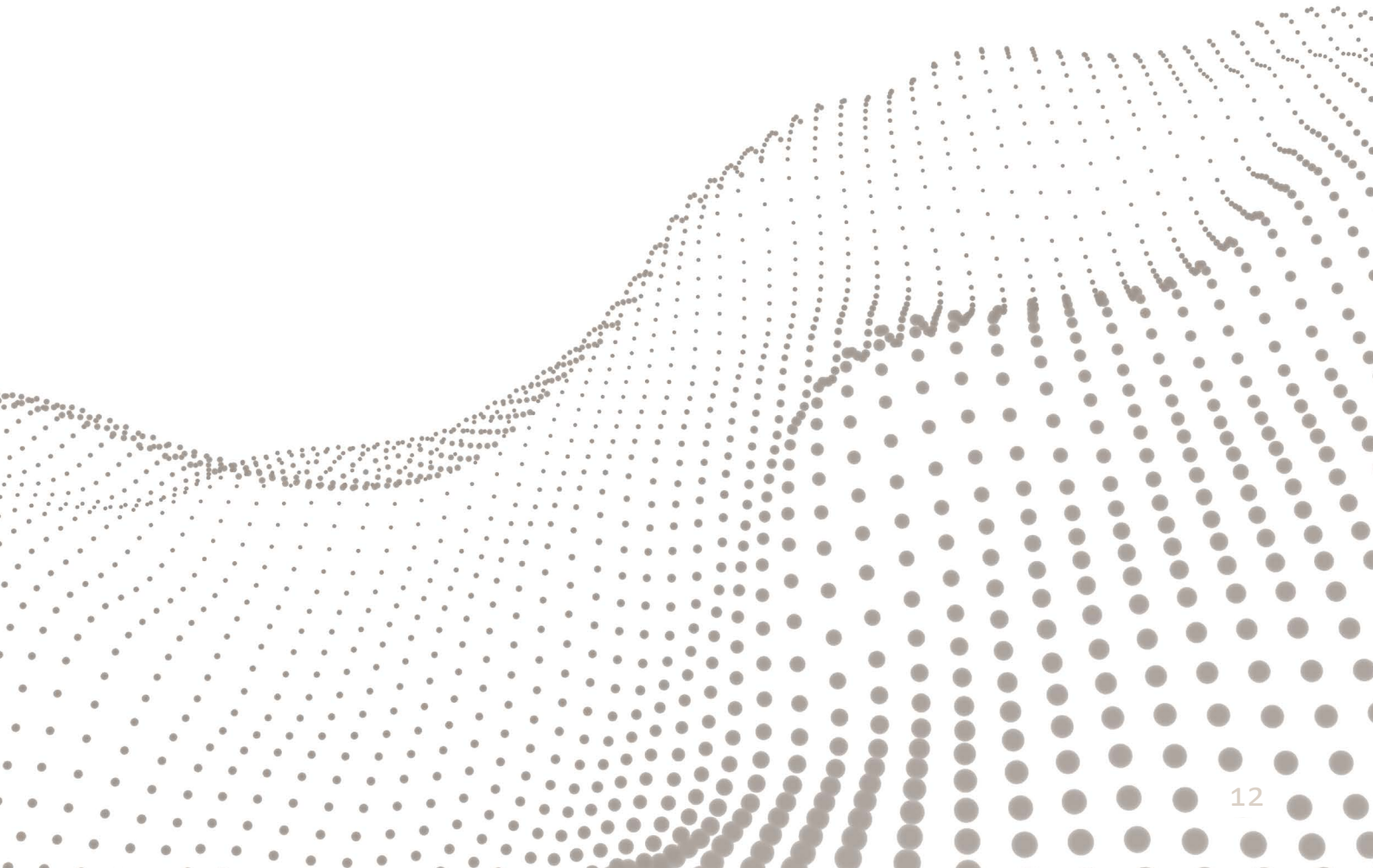
Über eine lokale Client-Session erfolgt eine Überprüfung der registrierten Tools:

```
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
import asyncio

async def test_tools():
      server_params = StdioServerParameters(command="python", args=["-m", "mcp.
server", "--connection_type", "stdio"])
    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await session.list_tools()
            print([t.name for t in tools.tools])

asyncio.run(test_tools())
```

Daraufhin sollte die Konsole die verfügbaren Tools ausgeben, z. B.:

```
['ga4_get_report', 'bq_query', 'realtime_overview']
```

Damit ist der lokale MCP-Server einsatzbereit und kann Requests von Gemini simulieren. Nach erfolgreichem Test empfiehlt sich die Bereitstellung auf Cloud Run (Gen 2) für produktive Umgebungen.



3.4 Remote Setup mit Google Cloud Run

Das Remote Setup ist der Kern der produktiven Bereitstellung eines MCP-Servers. Durch den Betrieb auf Google Cloud Run (Gen 2) erhält der Dienst eine serverlose Architektur mit automatischer Skalierbarkeit, ein sicheres Laufzeitumfeld und eine native Integration in IAM, Cloud Logging und Secret Manager.

1. Aufbau und Bereitstellung des Containers

Der MCP-Server wird als Docker Container Image verpackt und in der Artifact Registry gespeichert. Der Aufbau erfolgt mit Cloud Build:

```
gcloud builds submit --tag
europe-west4-docker.pkg.dev/$PROJECT_ID/mcp/mcp-server:v1
```

Anschließend wird das erstellte Image als Cloud Run Service deployed:

```
gcloud run deploy mcp-server \
  --image=europe-west4-docker.pkg.dev/$PROJECT_ID/mcp/mcp-server:v1 \
  --region=europe-west4 \
  --service-account=mcp-server-sa@$PROJECT_ID.iam.gserviceaccount.com \
  --no-allow-unauthenticated \
  --set-secrets="MCP_CLIENT_TOKEN=MCP_CLIENT_TOKEN:latest" \
  --cpu=1 --memory=512Mi --timeout=300
```

Das Ergebnis ist ein serverloser MCP-Endpunkt (*/mcp*), der eingehende Tool Calls authentifiziert, entgegennimmt und auf die Google APIs zugreift.

2. Authentifizierung zwischen LLM und MCP Server

Ebene	Beschreibung
Bearer-Token-Verfahren	Der Client übermittelt ein Secret (Authorization: Bearer …), das im Secret Manager verwaltet wird.
IAM "Authenticated Invoker"	Optional wird der Dienst nur für autorisierte Service Accounts innerhalb des Projekts zugänglich gemacht.
VPC Connector + Private Google Access	stellt sicher, dass API-Aufrufe an GA4 oder BigQuery innerhalb des Google-Backbones bleiben (kein externer Traffic).
Logging & Monitoring	Alle Requests werden automatisch in Cloud Logging protokolliert, Fehler in Error Reporting erfasst.

Der Zugriff auf den MCP-Server darf ausschließlich über eine Authentifizierung erfolgen. Dabei gibt es je nach Typ des Clients zwei empfohlene Varianten:

(a) Bearer-Token-Verfahren (empfohlen für LLMs wie Gemini, Claude etc.)
Beim Deployment wird im Secret Manager ein Token gespeichert (MCP_CLIENT_TOKEN). Das LLM bzw. der Host-Client sendet dieses Token bei jedem Request im Header mit:

```
Authorization: Bearer <MCP_CLIENT_TOKEN>
```

2. Authentifizierung zwischen LLM und MCP Server	Anschließend prüft der MCP-Server das Token serverseitig (z. B. über den FastAPI-Header-Validator):
	`from fastapi import Request, HTTPException`
	
	`async def verify_token(request: Request):`
	`  auth = request.headers.get("Authorization")`
	`  if auth != f"Bearer {os.environ['MCP_CLIENT_TOKEN']}":`
	`  raise HTTPException(status_code=401, detail="Unauthorized")`
	
	(b) IAM "Authenticated Invoker" (empfohlen für interne Services oder Service Accounts)
	Der Cloud Run Service wird mit eingeschränkter IAM-Policy betrieben. Das bedeutet, dass nur bestimmte Service Accounts den Dienst aufrufen dürfen:
	`gcloud run services add-iam-policy-binding mcp-server \`
	`  --member="serviceAccount:gemini-client@$PROJECT_ID.iam.gserviceaccount.com" \`
	`  --role="roles/run.invoker"`
	
	Damit kann nur der registrierte Client (z. B. Gemini über Vertex AI oder ein interner Proxy) den Endpoint aufrufen.

3. Netzwerk und Compliance-Konfiguration	Alle Requests werden in Cloud Logging erfasst, einschließlich: • Invoker-ID (Service Account OAuth User) • Request-Zeitstempel • Too-Name • Response Code Fehlgeschlagene Authentifizierungen werden automatisch in Cloud Error Reporting protokolliert.
4. Datenschutz und Compliance	Der MCP-Server kann innerhalb einer EU-Region (z. B. europe-west4) betrieben werden. Dadurch verbleiben alle Datenverarbeitungsvorgänge im Rahmen der EU Data Boundary, während Authentifizierung und API-Aufrufe über das Google-interne Netzwerk abgewickelt werden.

3.5 Anbindung des LLM Gemini (MCP-Clients)

Sobald der MCP-Server aktiv ist, können LLM-basierte Clients wie Gemini 2.5 Pro über das MCP-Protokoll auf definierte Tools zugreifen. Die Kommunikation läuft über standardisierte JSON-Nachrichten, die Funktionsaufrufe (Function Calls) und Ergebnisse strukturieren. Der Ablauf besteht im Wesentlichen aus drei Phasen:

1. Initialisierung des Clients und Tool-Discovery
2. Erzeugung des Function Calls durch das LLM
3. Weiterleitung des Calls an den MCP-Server und Rückgabe der Antwort

1. Initialisierung des Clients

Beispiel für die Initialisierung des MCP Python Clients und den Zugriff auf verfügbare Tools:

```
import asyncio
from mcp import ClientSession
from mcp.client.http import HttpTransport

MCP_URL = "https://mcp-server-ew4.a.run.app/mcp"
TOKEN = "super-secret-client-token"

async def list_tools():
    async with HttpTransport(
        MCP_URL,
        headers={"Authorization": f"Bearer {TOKEN}"}
    ) as transport:
        async with ClientSession(transport) as session:
            await session.initialize()
            tools = await session.list_tools()
            print("Verfügbare Tools:", [t.name for t in tools.tools])

asyncio.run(list_tools())
```

Als Ergebnis wird folgende Nachricht ausgeliefert, womit die Tool-Discovery abgeschlossen ist und das LLM nun Funktionsaufrufe an diese Tools generieren kann.

```
Verfügbare Tools: ['ga4_get_report', 'bq_query', 'realtime_overview']
```

2. Aufruf des Tools über Gemini (Function Calling)

Gemini nutzt die Function-Calling-Funktion, um basierend auf natürlicher Sprache strukturierte Funktionsaufrufe zu erzeugen. Die generierte Funktionsdefinition wird anschließend vom Host an den MCP-Client weitergegeben, der die tatsächliche Ausführung am Server vornimmt.

```
from google import genai
from google.genai import types
client = genai.Client(api_key=os.getenv("GEMINI_API_KEY"))

response = client.models.generate_content(
    model="gemini-2.5-pro",
    contents="Zeig mir die Sitzungen der letzten 7 Tage in Property 123456",
    config=types.GenerateContentConfig(
        temperature=0,
        tools=[
            types.Tool(function_declarations=[{
                "name": "ga4_get_report",
                "description": "Abfrage der GA4 Data API",
                "parameters": {
                    "property_id": "string", "start": "string",
                    "end": "string", "metrics": "array", "dimensions": "array"
                }
            }])
        ]
    )
)
```

Daraufhin gibt Gemini einen strukturierten *function_call* zurück, zum Beispiel:

```
{
  "function_call": {
    "name": "ga4_get_report",
    "args": {
      "property_id": "123456",
      "start": "2025-10-01",
      "end": "2025-10-07",
      "metrics": ["sessions"],
      "dimensions": ["source", "medium"]
    }
  }
}
```

Der Host-Prozess nimmt diesen Aufruf entgegen und übergibt ihn an den MCP-Client (siehe 1. Schritt), der den Request an den Cloud-Run-Server weiterleitet. Der Server führt das Tool aus (z. B. GA4 Data API), formatiert das Ergebnis und sendet es strukturiert zurück.

3. Antwortverarbeitung	Im letzten Schritt wird das Ergebnis vom MCP-Server als JSON-Objekt an den Client zurückgegeben:

```
{
  "header": ["source", "sessions"],
  "rows": [
    ["Organic Search", 5321],
    ["Direct", 2410],
    ["Paid Social", 1825]
  ],
  "metadata": {
    "source": "GA4 Data API",
    "region": "europe-west4"
  }
}
```

Gemini interpretiert dieses Ergebnis und generiert daraus eine natürliche Sprachausgabe, die in etwa wie folgt aussehen könnt:

> *„In den letzten 7 Tagen kamen 53 % der Sitzungen über Organic Search, 24 % über Direct und 18 % über Paid Social.“*

4. Abschließende Worte

Mit dem Model Context Protocol (MCP) entsteht eine neue Möglichkeit, den ursprünglichen Gedanken von Self-Service-Analytics neu zu beleben. MCP verbindet Large Language Models (LLMs) direkt mit Datenbanken, APIs oder Analytics-Systemen und überwindet damit die technische Hürde, die Self-Service-Ansätze bisher oft scheitern ließ. Daten können nun über natürliche Sprache abgefragt, interpretiert und bereitgestellt werden, was einen entscheidenden Schritt darstellt, um analytisches Wissen im Unternehmen breiter zugänglich zu machen.

MCP bildet die technische Brücke zwischen KI-Modellen und Datenquellen verschiedener Anbieter. Ob BigQuery, Redshift, Azure Synapse, Snowflake oder BI-Systeme wie Power BI und Tableau – MCP schafft eine einheitliche Zugriffsschicht, über die Modelle wie Gemini, Claude oder ChatGPT sicher und nachvollziehbar mit bestehenden Unternehmensdaten interagieren können.

Zukünftig wird MCP eine zentrale Rolle dabei spielen, kontextbasierte und KI-gestützte Analysen in Unternehmen zu verankern. Mit wachsender Tool-Kompatibilität und Frameworks wie FastMCP wird die Entwicklung eigener MCP-Server zunehmend einfacher und stabiler. Für Data Engineers, Data Analysts und Architects eröffnet sich damit ein verlässlicher Weg, bestehende Analytics-Infrastrukturen in das Zeitalter der nutzerorientierten und sprachbasierten Datenanalyse zu führen.

Über mohrstade

Unternehmen

mohrstade ist eine Beratung für Marketing-Technologie in München, Hamburg und Wien. mohrstade ist spezialisiert auf Projekte in den Bereichen Data Collection, Data Management, Analytics, Marketing Activation und Data Visualization. Diese Services bietet mohrstade in zertifizierten Partnerschaften mit Marketing-Software-Herstellern an.

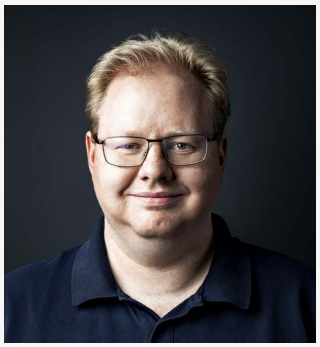
Managing Partner

Patrick Mohr

Co-Founder & Managing Partner

Patrick ist Gründer und Geschäftführer von mohrstade. Bereits während seines Studiums für BWL, Finance und Information (MSc) sammelte er Erfahrungen im Management Consulting. Später arbeitet er als SEA Manager, Data Scientist und Analytics Consultant bei Rocket Internet, Group M und UDG. 2017 baute er schließlich den Münchner Standort von Trakken auf. Parallel arbeitet er als Dozent an Universitäten. Darüber hinaus ist er Co-Organisator von Analytics Pioneers, der größten Analytics Community im DACH-Raum.

patrick@mohrstade.de



Marcus Stade

Co-Founder & Head of Analytics

Marcus ist Gründer von mohrstade und Head of Analytics. Darüber hinaus ist er Co-Organisator von Analytics Pioneers, der größten Analytics Community im DACH-Raum. Zuvor hat er im Bereich Web-Development und Online-Marketing gearbeitet. Auf seinem Blog www.marcusstade.de schreibt er regelmäßig zu Themen der Digitalen Analyse.

marcus@mohrstade.de

mohr
stade

Mohr & Stade GmbH
Friedrichstraße 1A
80801 München

www.mohrstade.de