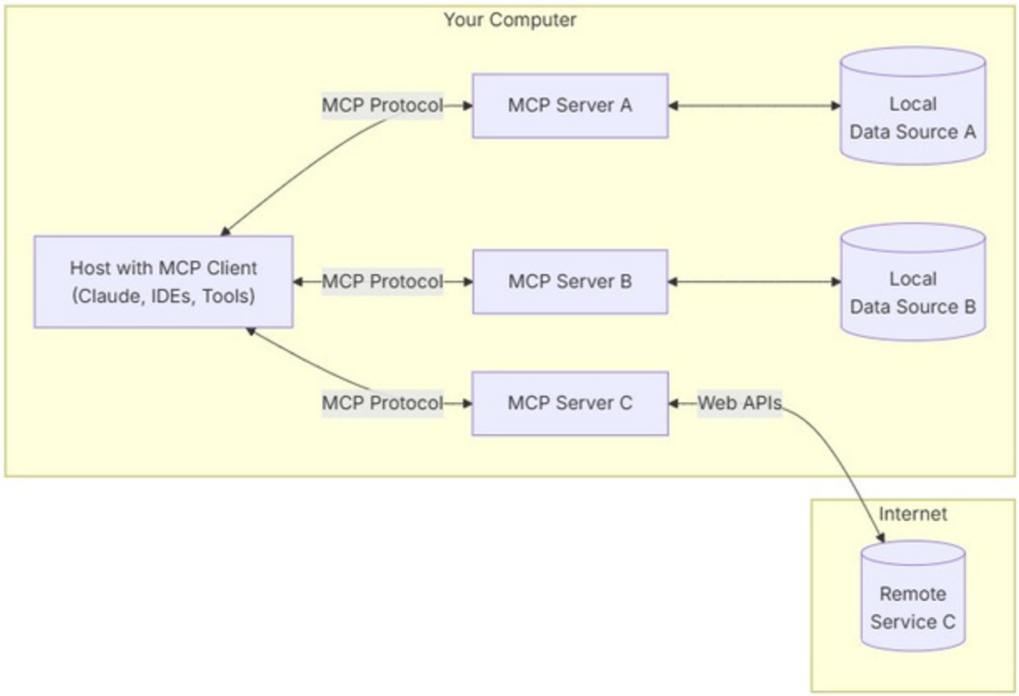


MCP Hosts: Programs like Claude Desktop, IDEs, or AI tools that want to access data through MCP
MCP Clients: Protocol clients that maintain 1:1 connection with servers
MCP Servers: Lightweight programs that each expose specific capabilities through the standardized Model Context Protocol
Local Data Sources: Your computer's files, databases, and services that MCP servers can securely access
Remote Services: External systems available over the internet (e.g., through APIs) that MCP servers can connect to

The Model Context Protocol (MCP) is built on a flexible, extensible architecture that enables seamless communication between LLM applications and integrations

What is MCP?

Model Context Protocol (MCP) is a way to define **structured tools, APIs, or functions** that a language model can call — allowing it to perform real-world actions or access data outside of natural language alone.

MCP is an **orchestration protocol** — a system design pattern or API layer — for organizing how external tools, APIs, or services (like Zoom, Weather, Outlook) are called by an LLM (or Agent) during runtime.

UserPrompt→LLM ()

|

▼

DoesLLMneedexternal data/tools?

|

├——No→LLM generates a response directly

|

└—— Yes →

1.LLMdecides to call a tool (via function/tool-calling)

2.MCPmetadata tells LLM what tool exists and how to call it

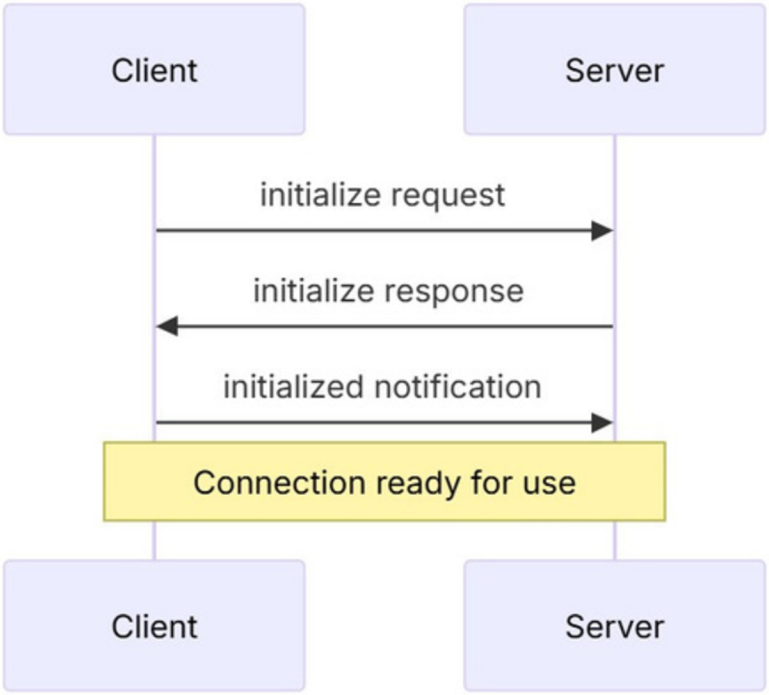
3.YourOrchestrator calls the external API (e.g. weather, zoom)

4.Response is sent back to LLM

5.LLMcontinues generation with the new data

Modelcontextprotocol is an open protocol that standardizes how applications providecontexttoLLMS

- MCPismedian between LLM and MCP servers



Prompts in MCP are predefined templates that can:

Accept dynamic arguments
Include context from resources
Chain multiple interactions
Guide specific workflows
Surface as UI elements (like slash commands)

MCP servers
We're not just talking about MCP — we're **building it.**

Weather MCP Server Description

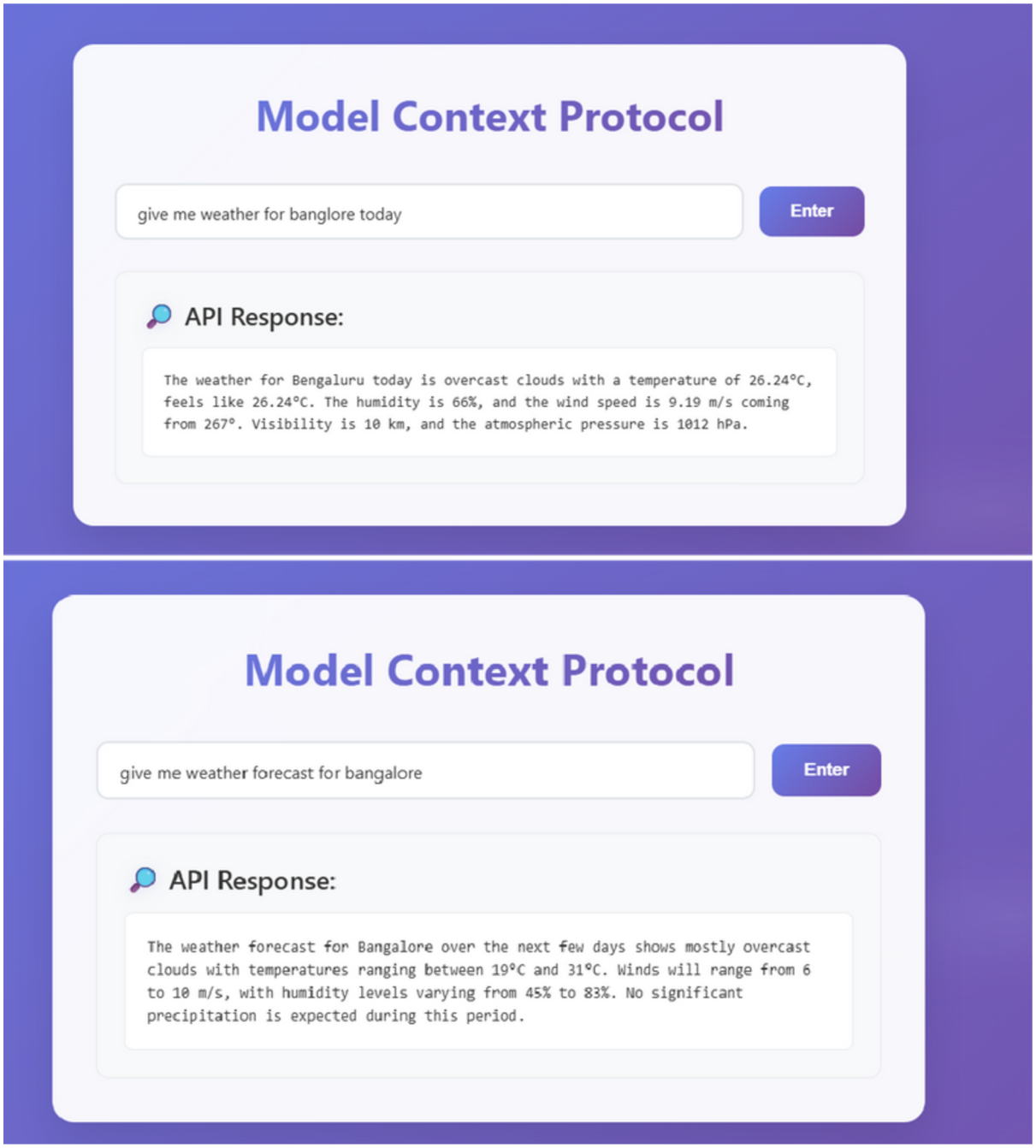
The Weather MCP Server is a Node.js-based microservice built with Express.js that serves as an MCP (Model Context Protocol) server to provide weather data. It acts as an external API endpoint that can be integrated with MCP clients, such as LLM-powered agents, to fetch real-time weather information.

Key features include:

- **Current Weather by City Name:** Accepts a POST request with a city name and returns the current weather conditions using the OpenWeatherMap API. If the API key is missing, it returns mock data for testing.
- **5-Day Weather Forecast by City:** Provides a 5-day weather forecast for a specified city, with the same fallback to mock data when the API key is absent.
- **Current Weather by Coordinates:** Allows clients to fetch current weather by specifying latitude and longitude coordinates, with validation of input ranges.
- **Health Check Endpoint:** Returns the health status of the service and lists available endpoints.
- **Root Endpoint:** Provides metadata about the service, including version, usage instructions, and available endpoints.

The server uses environment variables to securely manage the OpenWeatherMap API key (OPENWEATHER_API_KEY). When the key is not present, the server defaults to providing mock weather data, facilitating development and testing without external dependencies.

This MCP server exposes standardized REST API endpoints, allowing LLM-based clients to query weather data efficiently within the MCP framework. The endpoints are designed to be simple and consistent with MCP principles, enabling seamless integration in multi-server ecosystems.



Zoom MCP Server Description

The Zoom MCP Server is a Node.js-based microservice built with Express.js that provides an MCP-compliant interface for interacting with Zoom's API. It serves as an MCP server, enabling clients to programmatically manage Zoom meetings and retrieve meeting information in a standardized way.

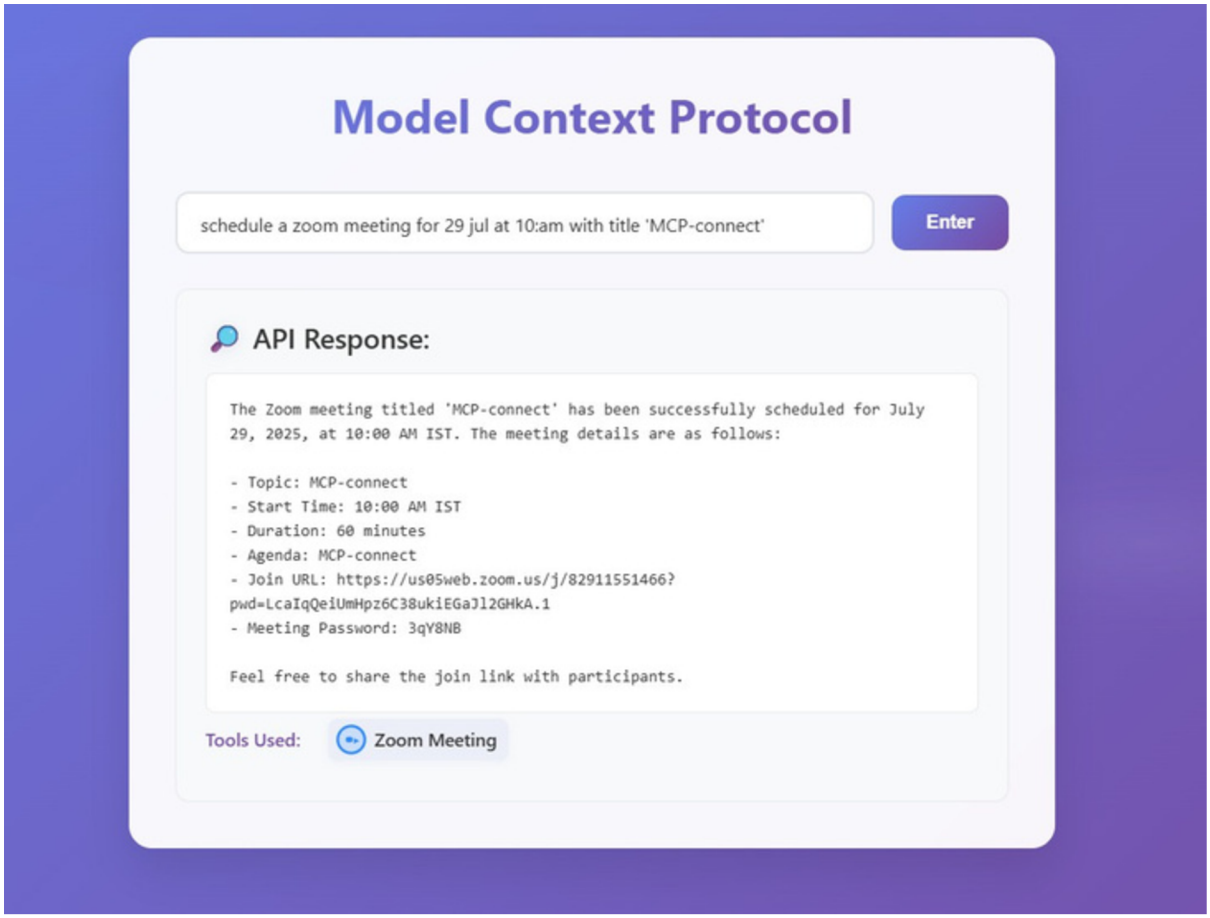
Key functionalities exposed via REST endpoints include:

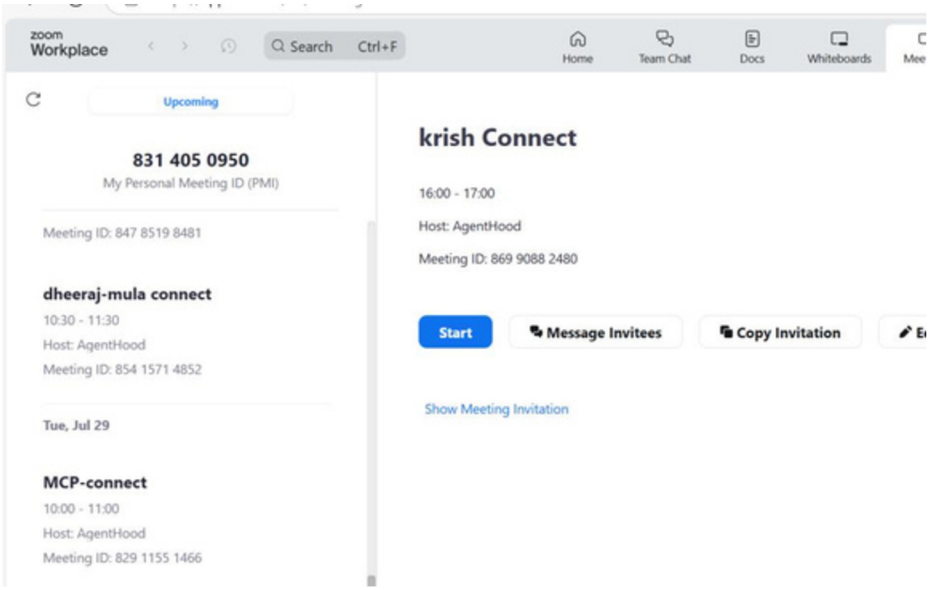
- **List Meetings:** Fetches all meetings for the authenticated Zoom user.
- **Get Meeting Details:** Retrieves detailed information about a specific meeting by its meetingId.
- **Create Meeting:** Allows clients to create a new Zoom meeting with configurable meeting parameters.
- **Delete Meeting:** Deletes an existing Zoom meeting by meetingId.
- **List Past Meeting Participants:** Provides a list of participants for a past Zoom meeting, identified by its meetingUUID.

The server handles OAuth 2.0 authentication using Zoom's Account Credentials grant type to securely obtain access tokens, which are used to authorize all API requests.

A unified /tools/call endpoint provides a flexible interface to invoke any supported tool by specifying the tool name and parameters in the request body. Additionally, the server exposes a plugin manifest endpoint (/.well-known/ai-plugin.json) detailing available tools, their descriptions, and parameter schemas, enabling seamless discovery and integration by MCP clients or AI agents.

The Zoom MCP Server relies on environment variables (ZOOM_CLIENT_ID, ZOOM_CLIENT_SECRET, ZOOM_ACCOUNT_ID) to configure authentication securely. This design facilitates integration into larger MCP ecosystems where AI agents or services can manage Zoom meetings as part of automated workflows.


Model Context Protocol
schedule a zoom meeting for 29 jul at 10:am with title 'MCP-connect'
Enter
API Response:
The Zoom meeting titled 'MCP-connect' has been successfully scheduled for July 29, 2025, at 10:00 AM IST. The meeting details are as follows:
- Topic: MCP-connect
- Start Time: 10:00 AM IST
- Duration: 60 minutes
- Agenda: MCP-connect
- Join URL: https://us05web.zoom.us/j/82911551466?pwd=LcaIqQeiUmHpz6C38ukiEGaJl2GHkA.1
- Meeting Password: 3qY8NB
Feel free to share the join link with participants.
Tools Used:
Zoom Meeting


zoom Workplace
Search Ctrl+F
Home
Team Chat
Docs
Whiteboards
Upcoming
831 405 0950
My Personal Meeting ID (PMI)
Meeting ID: 847 8519 8481
dheeraj-mula connect
10:30 - 11:30
Host: AgentHood
Meeting ID: 854 1571 4852
Tue, Jul 29
MCP-connect
10:00 - 11:00
Host: AgentHood
Meeting ID: 829 1155 1466
krish Connect
16:00 - 17:00
Host: AgentHood
Meeting ID: 869 9088 2480
Start
Message Invitees
Copy Invitation
Show Meeting Invitation

Outlook MCP Server Tool Description

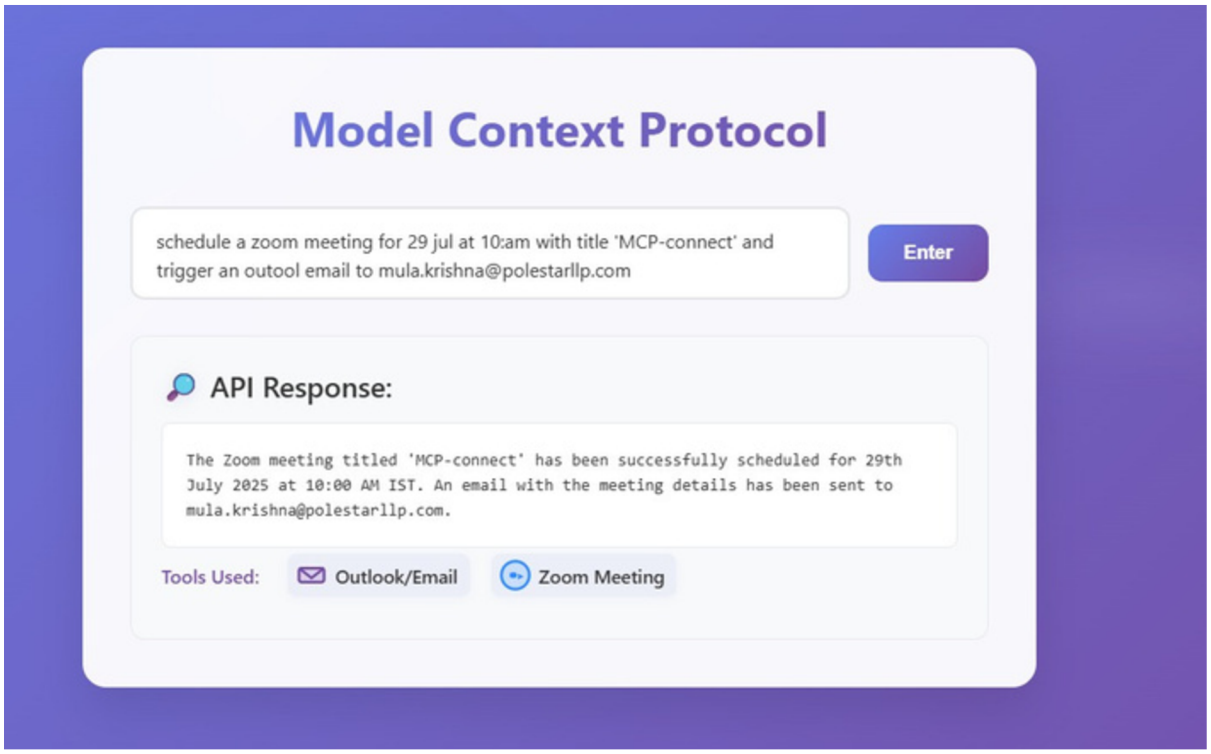
Send richly formatted emails via Outlook SMTP. Supports plain text and HTML content, multiple recipients (To, CC), and validates email addresses before sending. Automatically formats markdown-style tables into responsive HTML tables with enhanced styling, and applies professional email formatting for greetings, closings, and paragraphs.

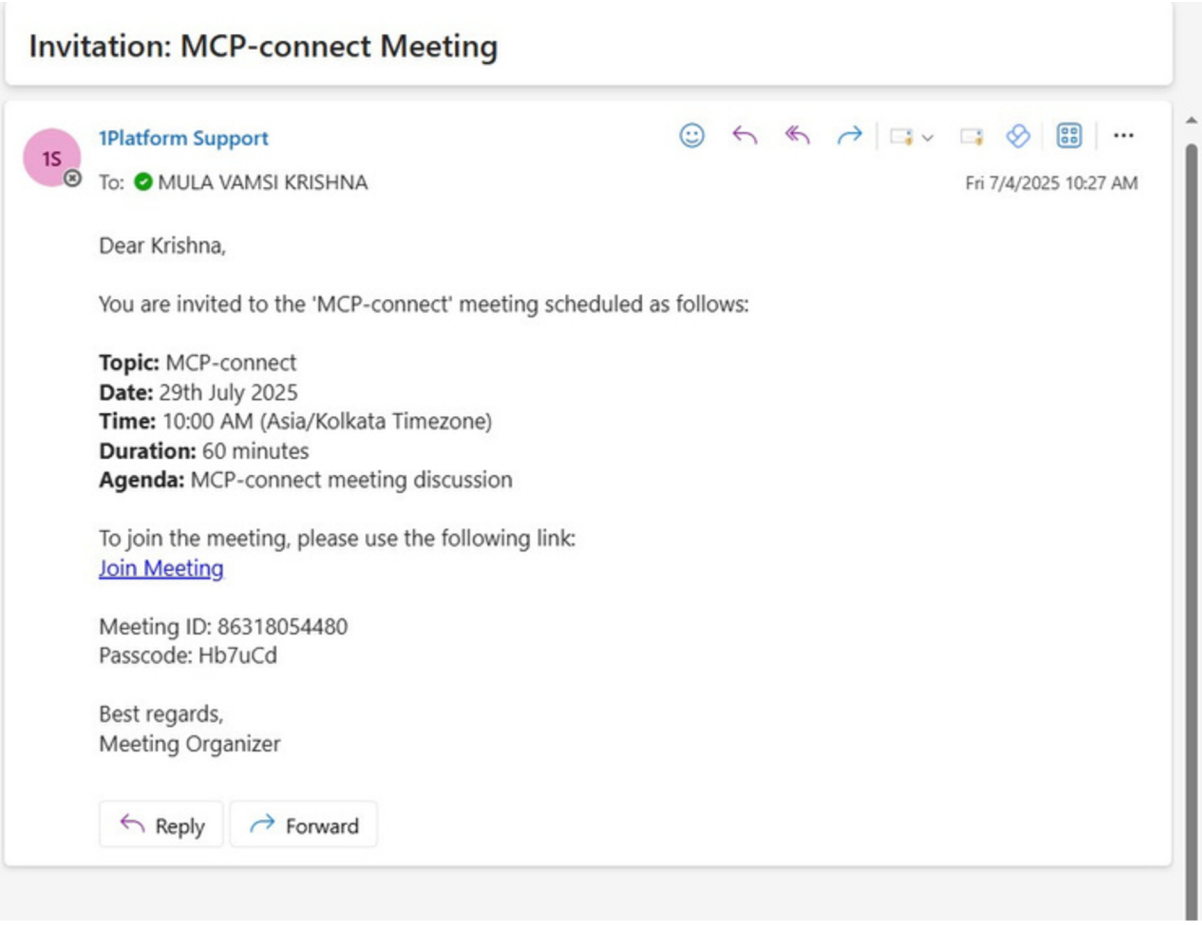
Parameters:

- subject (string) — Subject line of the email.
- body (string) — Email content, supports markdown tables and formatting.
- to (string or array of strings) — One or more recipient email addresses.
- cc (optional string or array of strings) — One or more CC email addresses.
- isHtml (optional boolean, default true) — Whether the body content is HTML formatted.

Response:

- error (boolean) — Indicates if sending failed.
- message (string) — Status message or error details.
- info (string, if success) — SMTP server response.





PostgreSQL MCP Integration for Agenthood

The PostgreSQL MCP module abstracts database operations and exposes the following capabilities via internal tooling:

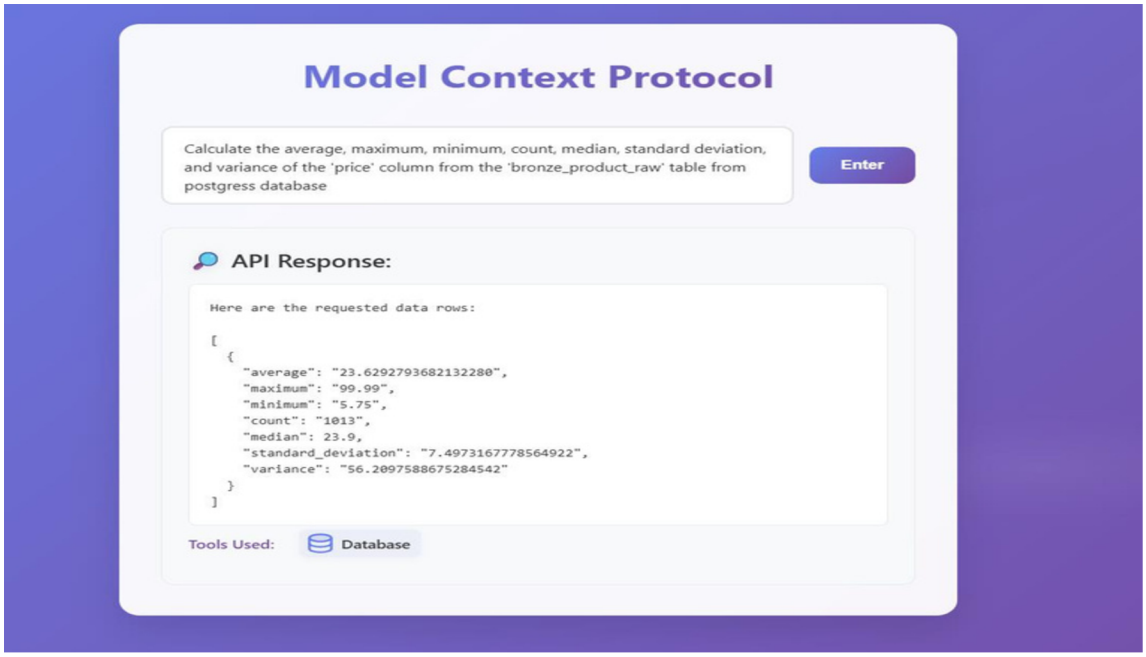
- Query execution
- Table schema introspection
- Health checks and usage statistics

This module can be invoked as a tool by the Agenthood orchestrator, enabling LLM-based agents to fetch and manipulate data programmatically.

⚠️ **Important Security Restriction**:

The **LLM cannot perform** INSERT, UPDATE, **or** DELETE **operations via natural language phrasing**.
These operations are strictly disabled through phrasing-based tool calls to **prevent accidental or unauthorized data mutation**.

Only **read-only access is** allowed in the default LLM-driven workflow to maintain data integrity and safety.



Lakehouse MCP Integration for Agenthood

The **Lakehouse MCP (Modular Control Point)** module enables secure, programmatic access to Microsoft Fabric Lakehouse data via an API-driven interface. This integration facilitates **LLM-based agents** and **Node.js services** to perform analytical operations without exposing the data layer directly.

✅ *Key Capabilities:*

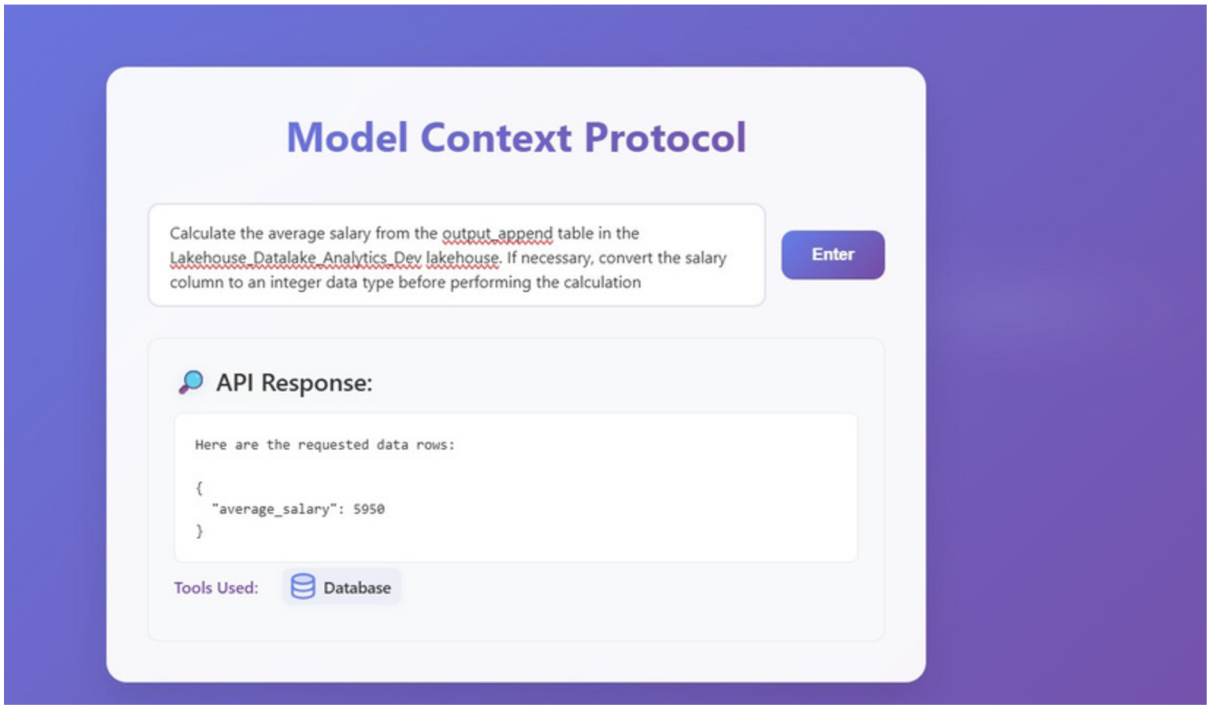
- **Dynamic Table Data Retrieval:**
  Fetch complete or filtered table data from a specified Lakehouse using parameterized queries.
- **Custom Aggregation via SQL:**
  Execute aggregation or analytical SQL queries dynamically and retrieve structured results.
- **Python-Powered Data Access:**
  Backend logic written in **Python (FastAPI)** interacts with Fabric Lakehouse using secure OAuth-based access tokens and SQLAlchemy for query execution.
- **Node.js Integration:**
  The API endpoints are consumable by Node.js or other front-end clients, enabling easy orchestration and dashboard connectivity.

⚠️ *Security Considerations*

- **Read-Only Access:**
  The system restricts all write operations (INSERT, UPDATE, DELETE) to prevent accidental or unauthorized changes to data.
- **OAuth2-Based Authentication:**
  Access tokens are retrieved using **Azure AD credentials**, ensuring secure, authenticated interactions with the Fabric SQL endpoint.

⚠️ *Supported Use Cases*

- Executing analytical queries like `AVG`, `SUM`, `COUNT` or complex joins



Dummy json APIS

`get_product_by_id`
Fetches detailedproduct information by product ID from the DummyJSON API. Accepts a singleinteger parameter id and returns product data including title, description, price,and more.

`get_user_by_id`
Fetches detaileduser information by user ID from the DummyJSON API. Accepts a

single integer parameter `id` and returns user data including name, email, address, and other profile info.

MCP servers

DummyJson

weather

HubSpot API reference | HubSpot

Amazon

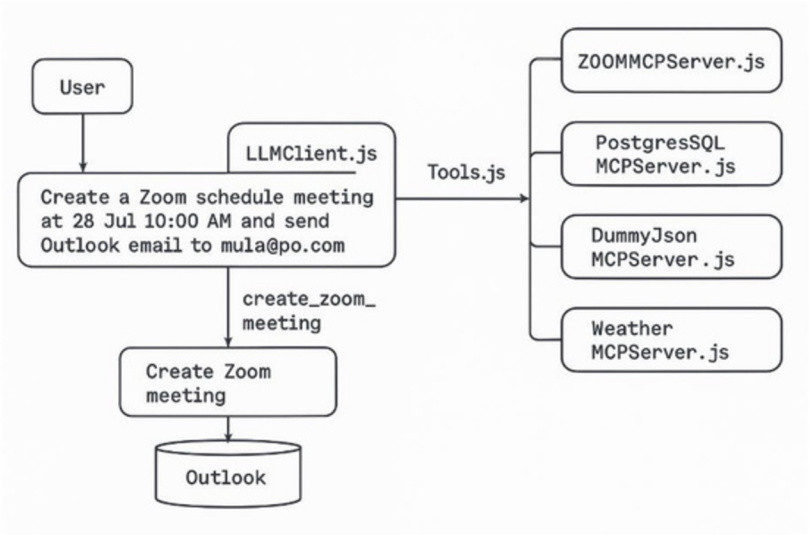
salesforce

Zoom

Azure devops

Chained tool invocations

Like fetching data -> scheduling calls with that data ->triggering an email



How can AgentHood become an MCP Client?

AgentHood can become an MCP client by integrating with an LLM (like OpenAI) and defining a clear protocol in the prompt that instructs the model how to interact with MCP servers, while also providing detailed information about the available MCP servers, their capabilities, and the expected tool schema, in accordance with the MCP specification.

How can data pass to the Tool via External MCP server be secured?

1. **Secure Transmission: Use HTTPS Always**

Ensure that **all API requests and responses** are made over **HTTPS (TLS/SSL)**.

HTTPS encrypts data in transit, protecting it from interception or tampering (man-in-the-middle attacks).

2. **Secure Storage of Client ID and Secret**

Storing all secret credentials in .env file

3. **Token Handling Best Practices (OAuth2)**

Use Short-Lived Access Tokens

Never Expose Tokens to the Client (Frontend/Browser)

Store Refresh Tokens Securely & Encrypted

4. Data Minimization

Only collect, process, and share **the minimum amount of data** required for a specific task or API call.

"Just enough data — and no more."

5. Logging and Auditing

1. Audit Logs for API Calls and Responses

What to Log:

- **Timestamp** of the request
- **API endpoint** accessed
- **HTTP method** (GET, POST, etc.)
- **User or service identity** (user ID, API key ID, etc.)
- **Source IP address**
- **Outcome** (success, error code, response time)

Authentication & Authorization:

1. Token Management (OAuth2 / JWT flows)
we have this already for 1platform application

2. **Credential Storage**

We are using .env file

3. Fine-Grained Permissions for Tool Access (Per User, Per Tool)

- Assign permissions/roles scoped to tools, e.g.:

```
{
  "userId123": {
    "permissions": {
      "zoom.create_meeting": true,
      "postgres.execute_query": false,
      "outlook.send_email": true
    }
  }
}
```

✅Implementation Plan

Here's a more production-grade flow:

Step	Action
⚠️ Step 1	Let users register their API keys via a frontend securely

⚠️ Step 2	Store those credentials in an encrypted DB table (or secure store)
⚠️ Step 3	When the agent runs, fetch ***that user's*** keys dynamically
⚠️ Step 4	Pass those secrets to the MCP tool handler (e.g., ZoomMCPServer)
⚠️ Step 5	Never log or expose secrets. Only use them in memory during the API call.

Each user can securely add and manage their own credentials(Zoom, DB, etc.) through a secure interface. All secrets are encryptedand stored per user — never shared or stored in flat files. The Agenthood system dynamically loads these credentials per userto perform actions safely.

Cost & Licensing

Zoom

Limited API usage (e.g., create meetings, list participants) under free/personal plans.

Paid accounts (Pro or higher) required for full API access.

Rate limits: 100 requests/day/user (can vary).

Outlook (Microsoft Graph)

Free for personal accounts, but enterprise apps (Azure AD tenants) may require Microsoft 365 licensing.

Rate limits exist (10k+ calls per day for basic access).

Weather API

Many providers (e.g., OpenWeatherMap) offer free tiers with rate limits.

Advanced forecasts/history may require paid subscriptions.

PostgreSQL
Self-hosted = no license fees.

If using managed DB (e.g., Azure Database for PostgreSQL), you pay per compute/storage/queries.

Scalability

What Happens When You Add Salesforce?

You'd do the following:

1. **Create anewfile:** `SalesforceMCPServer.js` withallSalesforce API logic.
2. **Addtooldefinitions**in tool.js, like:

```
salesforce: [
  {
    name: "get_salesforce_lead",
    description: "Get a lead from Salesforce by ID",
    endpoint: "POST /tools/salesforce_get_lead",
    parameters: { leadId: "string" },
    ...
  }
]
```

3. **Add Salesforce credentials**
4. Optionally: update the prompt to include this tool in the available context.

9. Circuit Breaker Pattern – In-Depth

The Circuit BreakerPattern isafault tolerancemechanism used to detect failures and prevent the system from repeatedly trying a failing service, thus allowing it time to recover and protecting the system from resource exhaustion.

9.1 Real-WorldAnalogy

Just like an electrical circuit breaker in a house trips to prevent overloading the system, a software circuit breaker prevents continued requests to a failing service to avoid cascading failures and high latency.

9.2 Circuit BreakerStates

- Closed – Normal operation, requests are sent to the tool.
- Open – The tool has failed too many times; requests are blocked or redirected to fallback.

• Half-Open – After a cooldown, a few requests are allowed through to test recovery.

9.3 When to Use

• Downstream tools are intermittently failing (timeouts, 5xx errors)

• APIs are under rate limits (429 errors)

• Services throw unhandled exceptions frequently

9.4 Basic Flow

1. MCP Server sends requests to tool.

2. Tool fails repeatedly (e.g., 5 times).

3. Circuit enters Open state and blocks requests temporarily.

4. After a cooldown period, enters Half-Open and tests a few requests.

5. If those succeed → circuit resets. If not → stays open.

9.5 SampleError ResponseDuring Circuit Open

```
{
  "tool_call_id": "abc123",
  "status": "error",
  "error": {
    "type": "circuit_open",
    "message": "The tool is temporarily unavailable due to repeated failures.",
    "retry_after": 60
  }
}
```

9.6 Tools and Libraries

• Node.js – opossum

• Python – pybreaker

• Java – resilience4j, Hystrix

• Go – sony/gobreaker

9.7 Best Practices

• Log all circuit transitions (open, half-open, closed)

• Combine with retry and fallback for full fault tolerance

• Use exponential backoff before retrying

• Track circuit metrics (open time, error rate, fallback usage)