Lianja MCP Server

OpenAI offers a variety of data connectors and integration options to help developers and businesses seamlessly integrate its AI models into their workflows.

OpenAI Connectors in ChatGPT

OpenAI's Connectors feature allows ChatGPT to securely connect to third-party applications like Google Drive, GitHub, and SharePoint. This enables users to search files, pull live data, and reference content directly within the chat interface. Additionally, custom connectors are available for ChatGPT Pro users and ChatGPT Team, Enterprise, and Edu workspaces, allowing integration with custom third-party apps and internal sources using the Model Context Protocol (MCP)

The Model Context Protocol (MCP) is an open standard developed to facilitate secure and standardized communication between AI models and external tools, systems, and data sources. OpenAI has adopted MCP across its products, including the ChatGPT desktop app, OpenAI's Agents SDK, and the Responses API, to enhance interoperability and simplify development in multi-model environments .

These connectors and integration platforms empower developers and organizations to harness OpenAI's advanced AI models, streamlining workflows and enhancing productivity across various applications.

In the context of OpenAI's Model Context Protocol (MCP), the model interacts with your tool via standardized endpoints.

The Lianja 11 Cloud Server provides these MCP Server endpoints, enabling it to be used directly by ChatGPT to pull live Lianja Cloud data as Lianja Apps are running. This functionality can operate in a Lianja Cloud tenancy as well as a private Lianja Cloud Server installation.

Lianja Cloud Server MCP Endpoints

The Lianja Cloud Server exposes these MCP Server endpoints:

1. /tools/list Describes available tools to the model.

Method: GET

Response: a List of tools with their input/output schemas and metadata.

2. /tools/invoke Invoked by the model when it wants to call a tool.

Method: POST

Request Body:

```JSON
{
  "tool_name": "get_weather",
  "tool_input": {
    "location": "London"
  }
}
```

Response:

```JSON
{
  "output": {
    "temperature": "18°C",
    "condition": "Partly cloudy"
  }
}
```

Example Tool Definition in /tools/list

When ChatGPT connects to your server, it fetches tool metadata like this:

```JSON
[
  {
    "name": "get_weather",
    "description": "Get the current weather for a location.",
    "input_schema": {
      "type": "object",
      "properties": {
        "location": {
          "type": "string",
          "description": "The city or place to get the weather for"
        }
      },
      "required": ["location"]
    },
    "output_schema": {
      "type": "object",
      "properties": {
        "temperature": {
          "type": "string"
        },
```

```
        "condition": {
          "type": "string"
        }
      }
    }
  }
]
```

As you can see, this is the TOOL_DEFINITION format created for you when you create an agent in the Lianja App Builder “Agents” workspace.

When an agent is deployed to a Lianja Cloud Server its TOOL_DEFINITION exposes the agent to the MCP Server and hence the /tools/list and the /tools/activate endpoints as described above.

3. /tools/add	Registers an agent with the MCP Server. Upload an agent to the Lianja MCP Server. Use an HTTP(S) POST with an APIKEY. Include the agent (including its TOOL_DEFINITION) in the POST body. POST /tools/add?apikey=xxx&agent=name_of_agent <body contains code of agent>
4. /tools/remove	Unregisters an agent and removes it from the MCP Server. GET /tools/remove%apikey=xxxx&agent=name_of_agent

When you edit agents in the Lianja App Builder “Agents” workspace, you can deploy an agent to an MCP server by clicking the “Upload and Register” icon in the header bar.

Similarly clicking the “Unregister” icon will “Unregister and remove” the agent from the MCP Server.

Both /tools/register and /tools/remove require an apikey specified. This is the APIKEY for the cloud server.

e.g.

```
Unset

GET
https://tenancy.lianjacloud.com/api/mcp/tools/remove?APIKEY=xxxxxxx&agent=myage
nt
```

Define the MCP Tool in Your API Request

When making a request to the OpenAI API, Lianja will include the MCP tools defined in tools.conf in the tools array. tools.conf and agents.conf in the agents workspace describe what tools are exposed to the MCP Server and how they are executed on the server.

JSON

```
{
  "tools": [
    {
      "type": "mcp",
      "server_url":
"https://tenancy.lianjacloud.com/api/mcp&apikey=xxxxxxxxxxxxxxx",
      "server_label": "your_server_label",
      "allowed_tools": ["tool_name_1", "tool_name_2"],
      "require_approval": "auto"
    }
  ]
}
```

ChatGPT integration using data connectors

If you're on ChatGPT Team or Enterprise, you can create custom data connectors using the Model Context Protocol (MCP). This allows ChatGPT to call your internal tools or private APIs externally from your Lianja Apps.

You need to register your custom connector via the ChatGPT admin panel.

Your Lianja Cloud Server implements standard MCP endpoints like /tools/list and /tools/invoke

ChatGPT Team is $30 per month per user or $25 per month if paid annually in advance.

Once your Lianja Cloud Server connector is registered, you can use it in Chat.

Ask ChatGPT things like:

Unset

```
Find the last 10 orders from my Lianja cloud data
```

Lianja AI Assistant: Use third party agents with Function Calling

You can achieve the same effect in the Lianja AI Assistant using local agents that act as a proxy talking to an MCP Server.

Example code:

Python

```python
import asyncio
import openai
from openai import AssistantEventHandler
from openai.agents import Agent
from openai_agents.mcp import MCPTool

# Set your API key
openai.api_key = "your-openai-api-key"

# Define your weather tool via MCP
weather_tool = MCPTool.from_server(
    server_url="https://your-mcp-server.com/api/mcp",
    server_label="weather-server",
    allowed_tools=["get_weather"],
    require_approval="auto"
)

# Define the task/goal
goal = "What is the weather like today in London?"

# Create the agent
agent = Agent(
    model="gpt-4o",  # or gpt-4-turbo
    instructions="You are a helpful assistant that fetches real-time weather
information.",
    tools=[weather_tool],
)

# Define the execution
async def run_agent():
    response = await agent.run(goal)
    print("Agent Response:\n", response)

# Run the agent
if __name__ == "__main__":
    asyncio.run(run_agent())
```