MCP Database Integrations

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

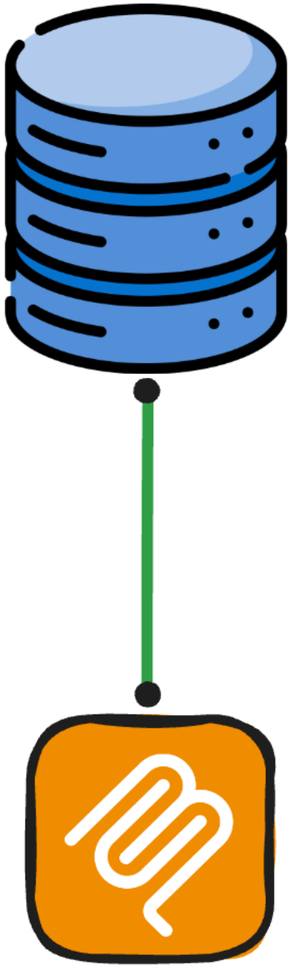


James Chapman
AI Curriculum Manager, DataCamp

datacamp

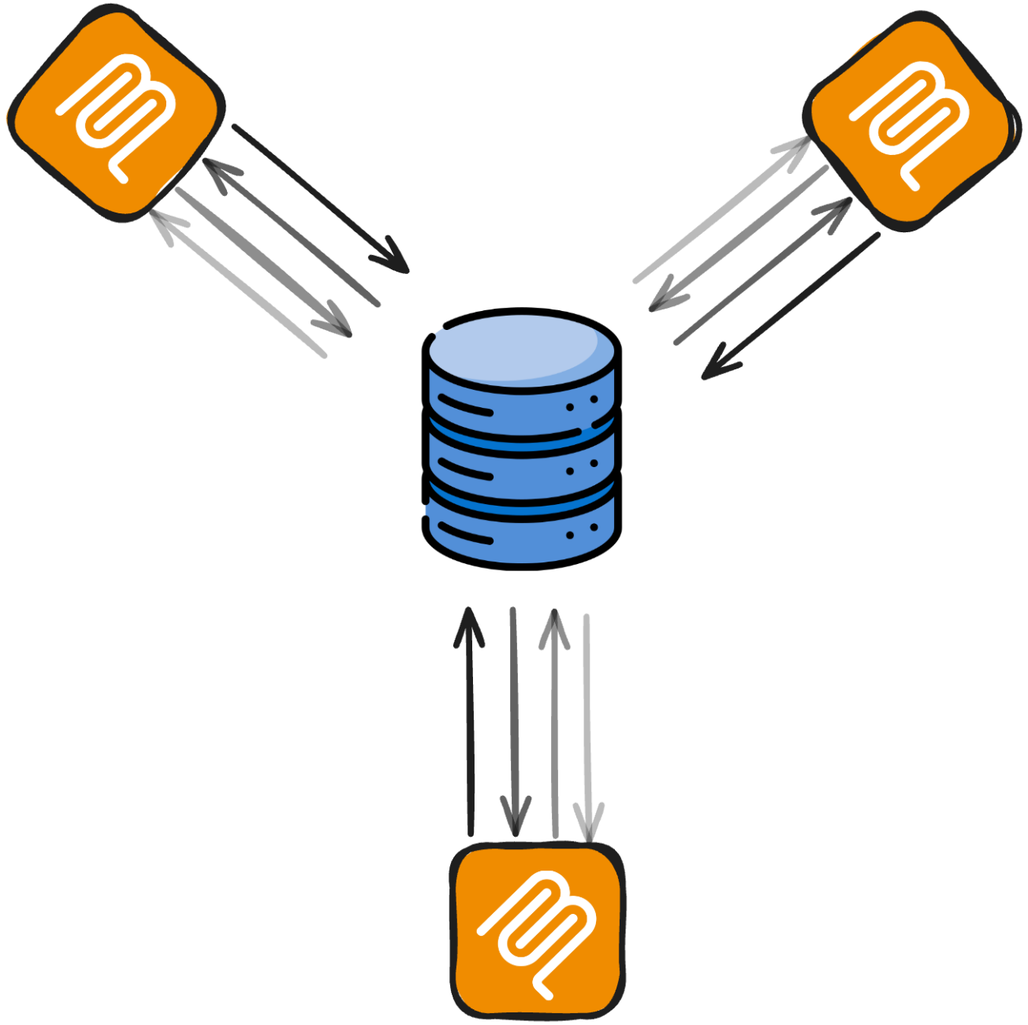
Why Add a Database to the Timezone Server?

- **Query power:** filter, search, and sort without loading the whole dataset
- **Scale:** fast lookups with database *indexes*



Why Add a Database to the Timezone Server?

- **Query power:** filter, search, and sort without loading the whole dataset
- **Scale:** fast lookups with database *indexes*
- **Concurrency:** multiple tool calls or clients hit the DB safely
- **Production-ready:** backups, replication, and a single source of truth



Data Access: Tools vs. Resources



Tools

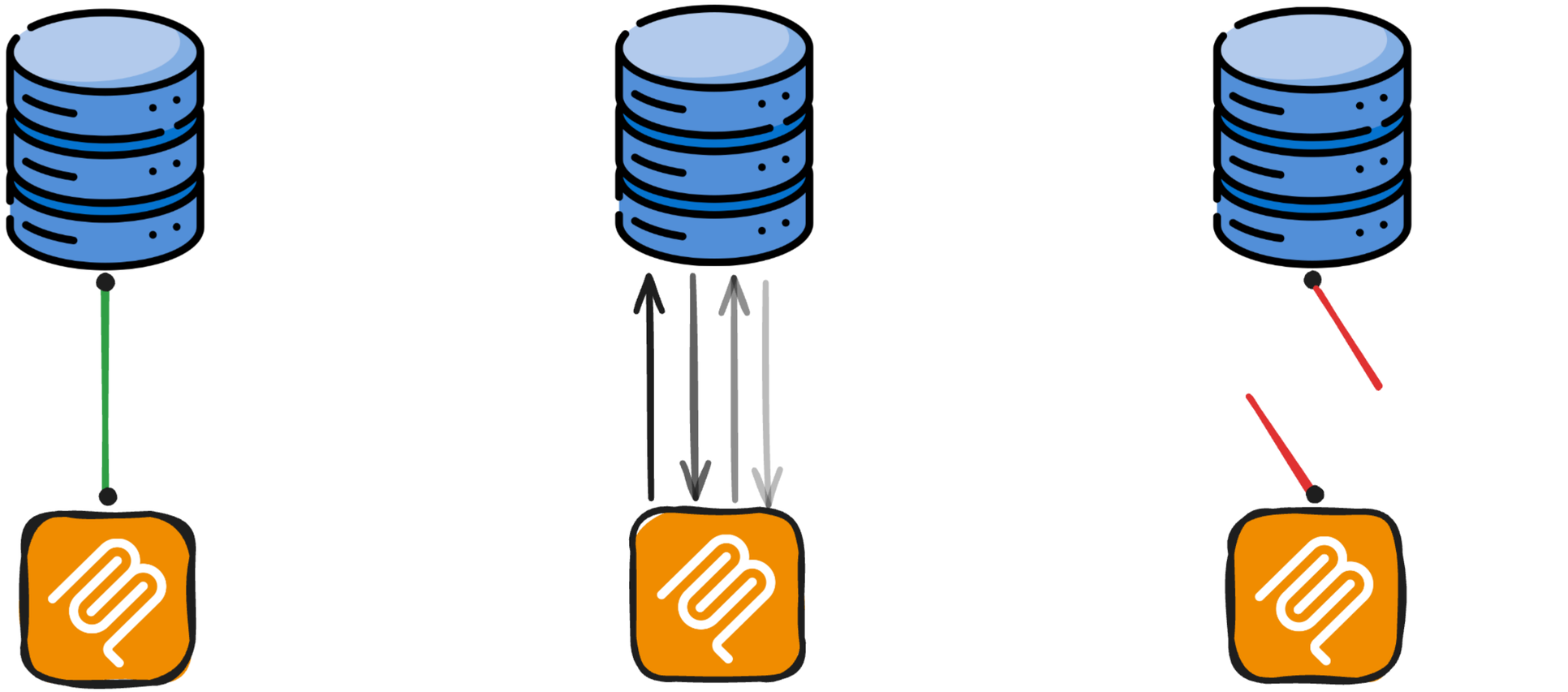
- Dynamic, heavily user-input dependent, writing operations
- **Examples:** Data analytics and operations



Resources

- Read-only, static data, reference information
- **Examples:** Guidelines, docs, policies

Connection Lifecycle



Database Resource: timezone_server.py

```python
from mcp.server.fastmcp import FastMCP
import sqlite3

mcp = FastMCP("Timezone Converter")

# Create connection at startup; reuse in handlers
conn = sqlite3.connect("timezones.db")
conn.row_factory = sqlite3.Row
@mcp.resource("db://timezones")
def get_locations() -> str:
    try:
        cursor = conn.execute("SELECT timezone FROM locations")
        rows = cursor.fetchall()
        return "\n".join(r["timezone"] for r in rows)
    except sqlite3.Error as e: return f"Error: {e}"
```

Database Lookup Tool: timezone_server.py

```
@mcp.tool()
def lookup_locations(prefix: str) -> str:
    """Find timezones that contain the given prefix."""
    try:
        cursor = conn.execute(
            "SELECT timezone FROM locations WHERE timezone LIKE ? LIMIT 50", (f"%{prefix}%",))
        rows = cursor.fetchall()
        return "\n".join(r["timezone"] for r in rows)
    except sqlite3.Error as e: return f"Error: {e}"

if __name__ == "__main__":
    try:
        mcp.run(transport="stdio")
    finally:
        conn.close()
```

Safety and Production Habits

- **Parameterized (`?`) queries:** never string-format user or LLM input into SQL
  - *Prompt injection* → malicious injection of code through prompts

```
SELECT timezone FROM locations WHERE timezone LIKE ? LIMIT 50
```

- Prefer **read-only** and **tightly scoped database access** for the MCP server
- **Apply limits:** max rows and timeouts

Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

MCP API Integrations

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)



James Chapman
AI Curriculum Manager, DataCamp

datacamp

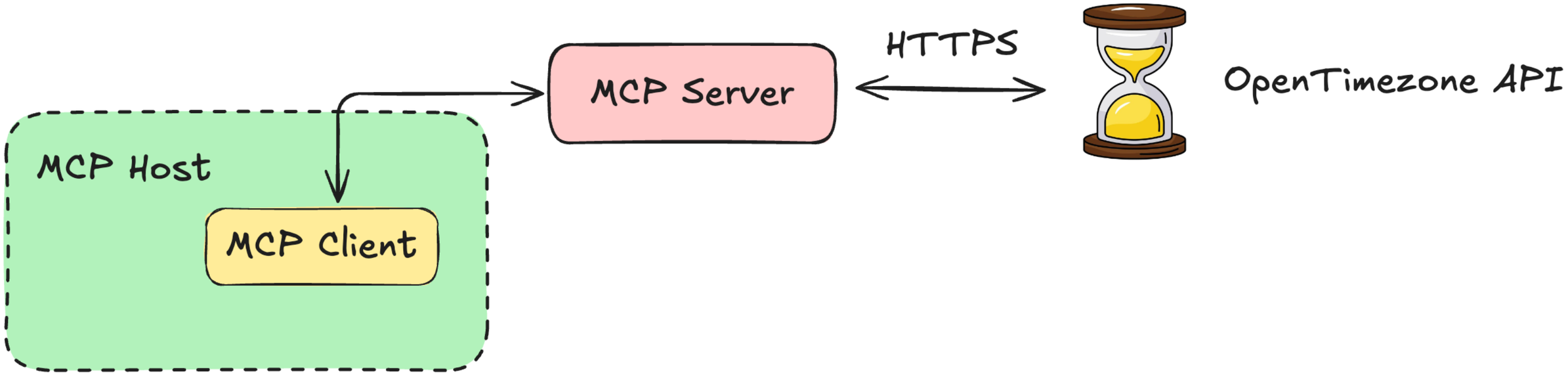
```python
# Create an MCP server instance
mcp = FastMCP("Timezone and Currency Converter")

@mcp.tool()
def convert_timezone(date_time: str, from_timezone: str, to_timezone: str) -> str:
    """
    Convert a datetime from one timezone to another.

    Args:
        date_time: The datetime string in ISO format (e.g., '2025-01-20T14:30:00')
        from_timezone: Source timezone (e.g., 'America/New_York')
        to_timezone: Target timezone (e.g., 'Europe/London')

    Returns:
        A string with the converted datetime and timezone information
    """
    # Define the API endpoint and format input data
    url = "https://api.opentimezone.com/convert"

    payload = {
        "dateTime": date_time,
        "fromTimezone": from_timezone,
        "toTimezone": to_timezone
    }

    # Make the API request
    response = requests.post(url, json=payload)

    data = response.json()
    converted_time = data.get('dateTime', 'N/A')

    return f"Time in {to_timezone}: {converted_time}"
```

Instantiate MCP Server

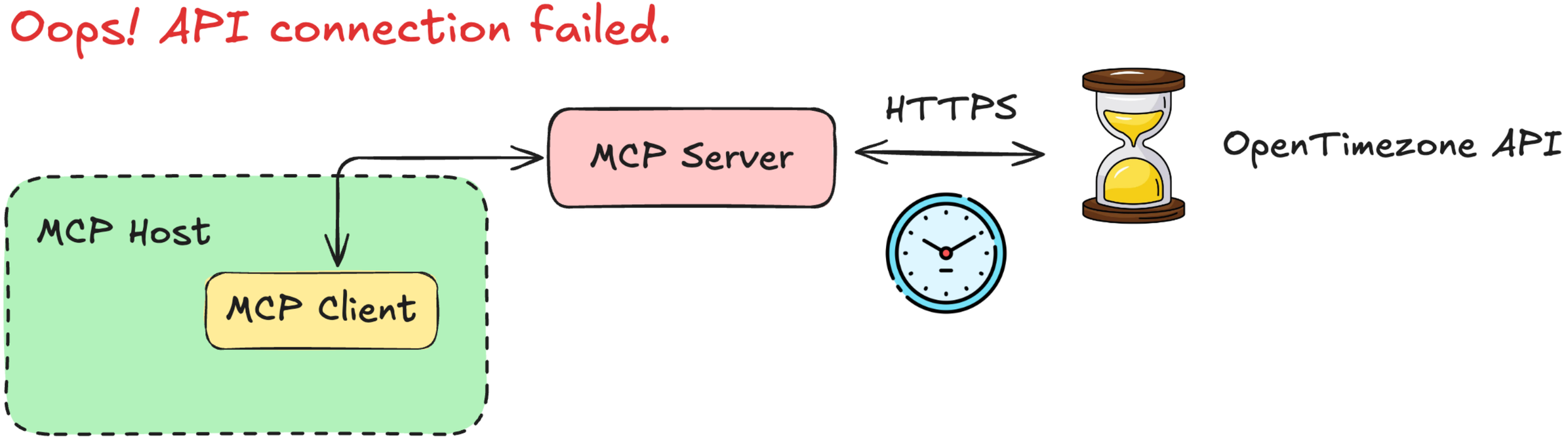
Define Tools

Doctrings and Typing

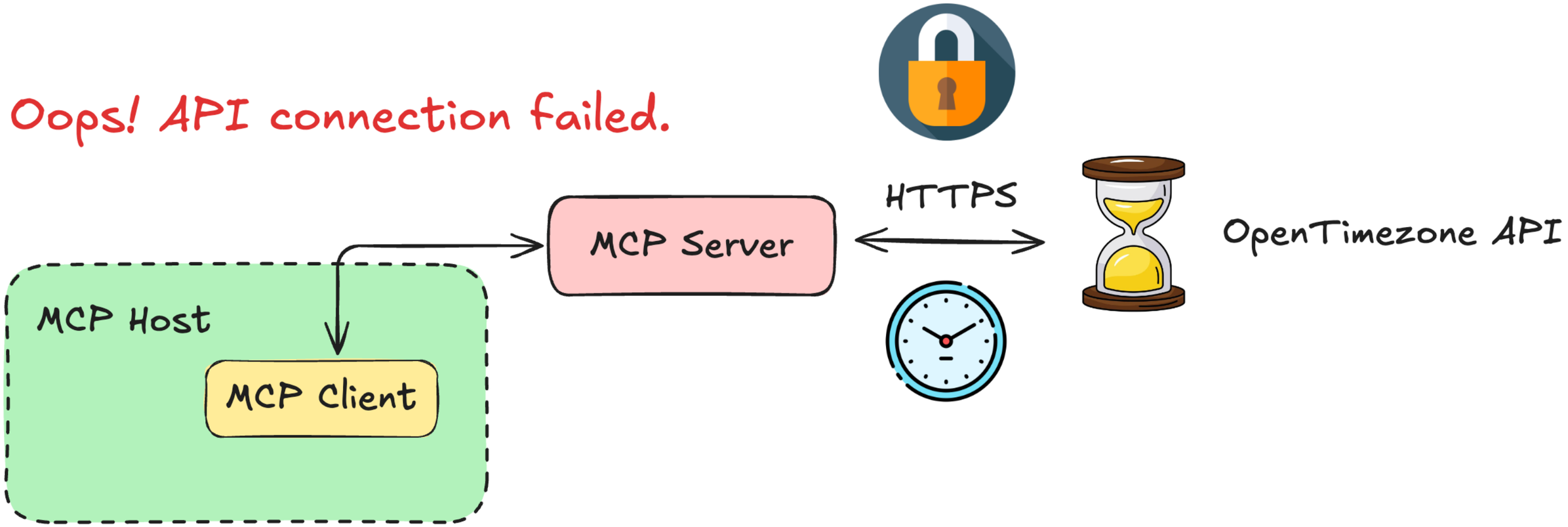
API Improvements



API Improvements



API Improvements

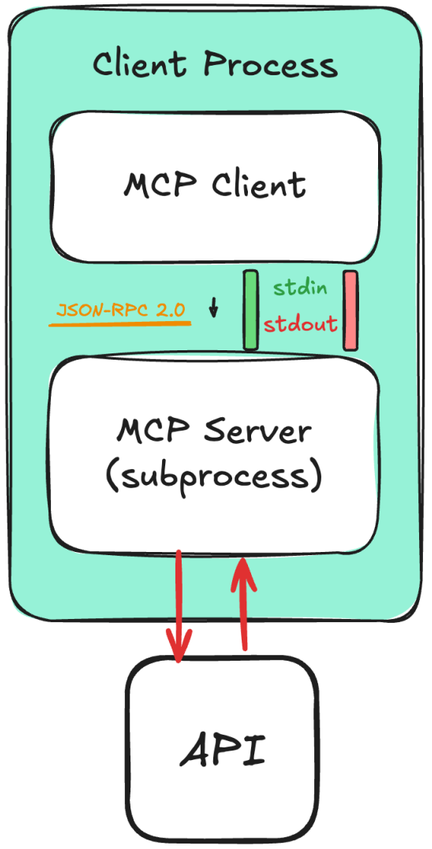


Timeout and Error Handling

```python
import requests

@mcp.tool()
def convert_timezone(date_time: str, from_timezone: str, to_timezone: str) -> str:
    payload = {"dateTime": date_time, "fromTimezone": from_timezone, "toTimezone": to_timezone}

    try:
        r = requests.post("https://api.opentimezone.com/convert", json=payload, timeout=10)
        r.raise_for_status()
        data = r.json()
        converted_time = data.get("dateTime", "N/A")
        return f"Time in {to_timezone}: {converted_time}"
    except requests.exceptions.RequestException as e:
        return f"Error converting timezone: {e}"
```

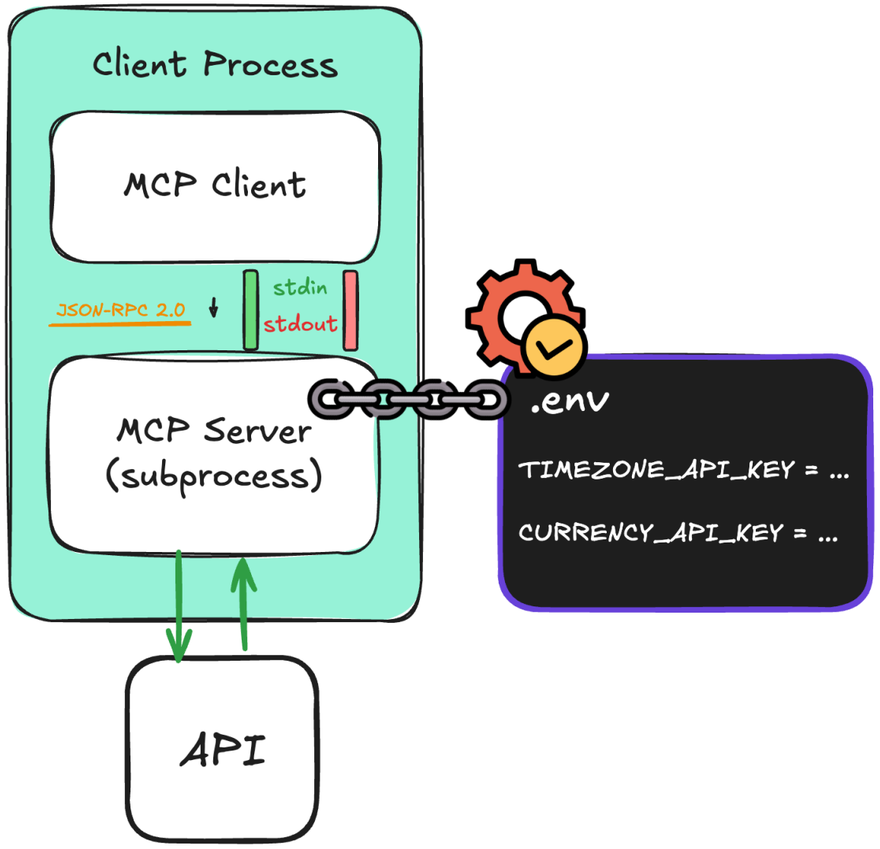
API Authentication: The Do's and Don'ts

- **API keys** → always accessed *server-side*



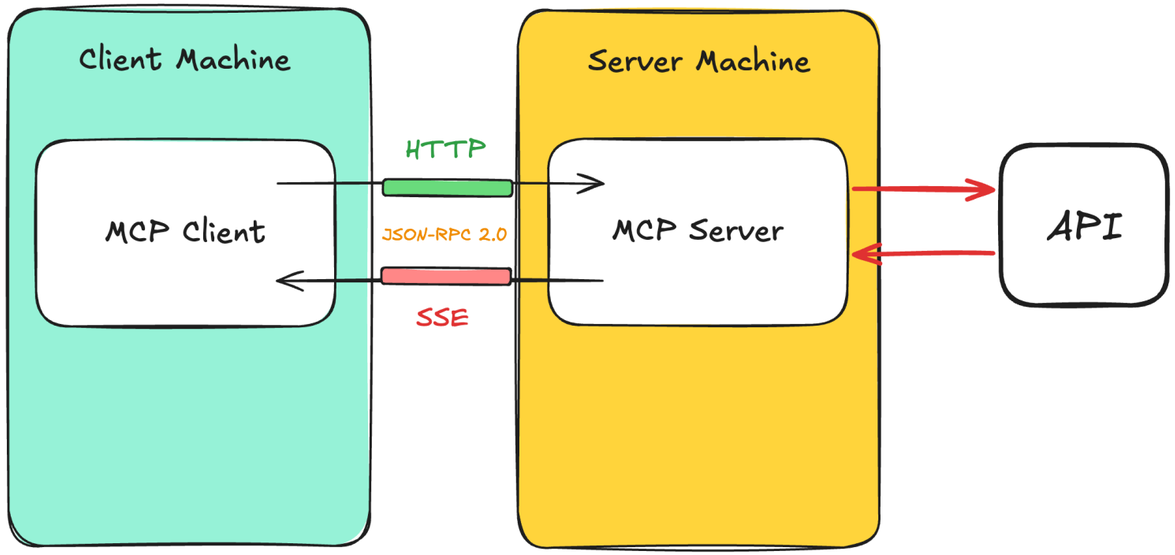
API Authentication: The Do's and Don'ts

- **API keys** → always accessed *server-side*
  - Local Host: Environment variables, `.env`



API Authentication: The Do's and Don'ts

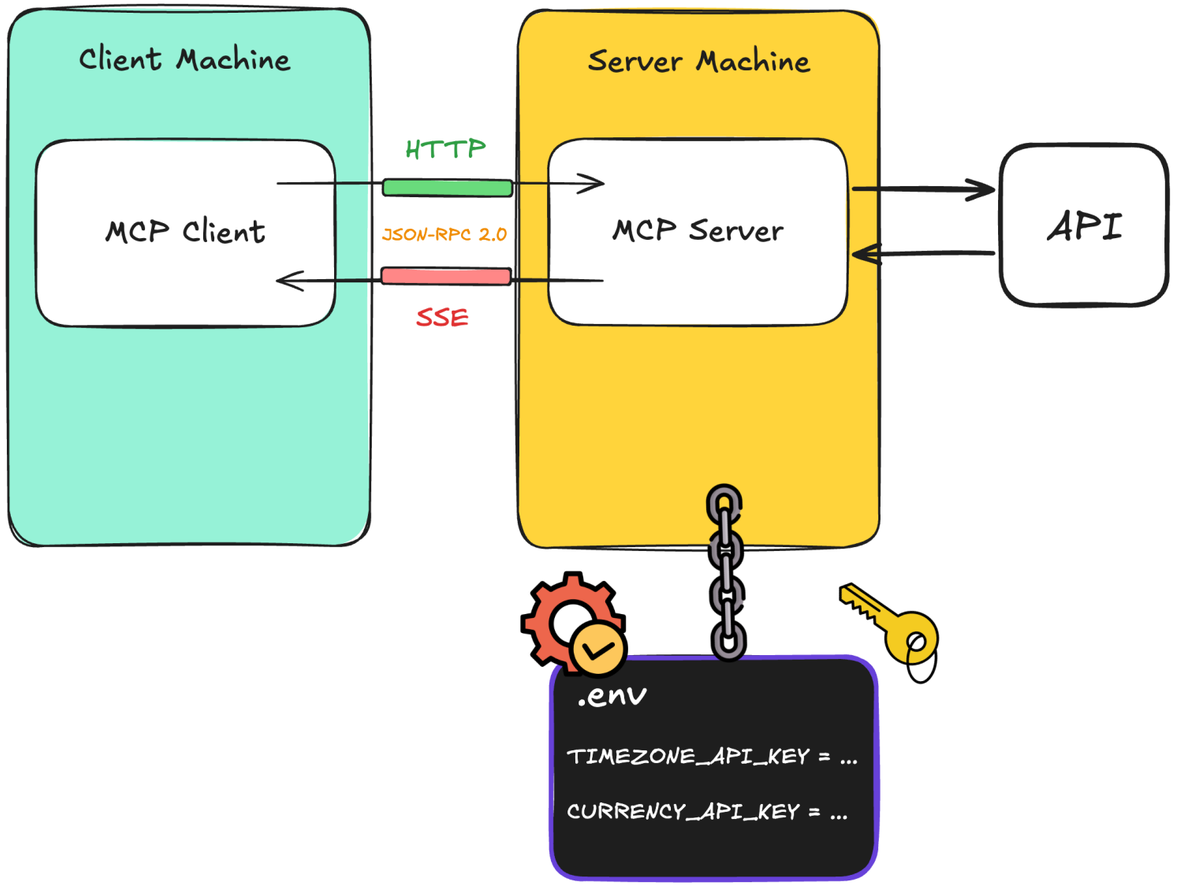
- **API keys** → always accessed *server-side*
  - Local Host: Environment variables, `.env`
  - Remote Host: Environment variables, managed secrets



API Authentication: The Do's and Don'ts

- **API keys** → always accessed *server-side*
  - Local Host: Environment variables, `.env`
  - Remote Host: Environment variables, managed secrets

The client never sends, receives, or accesses the credentials



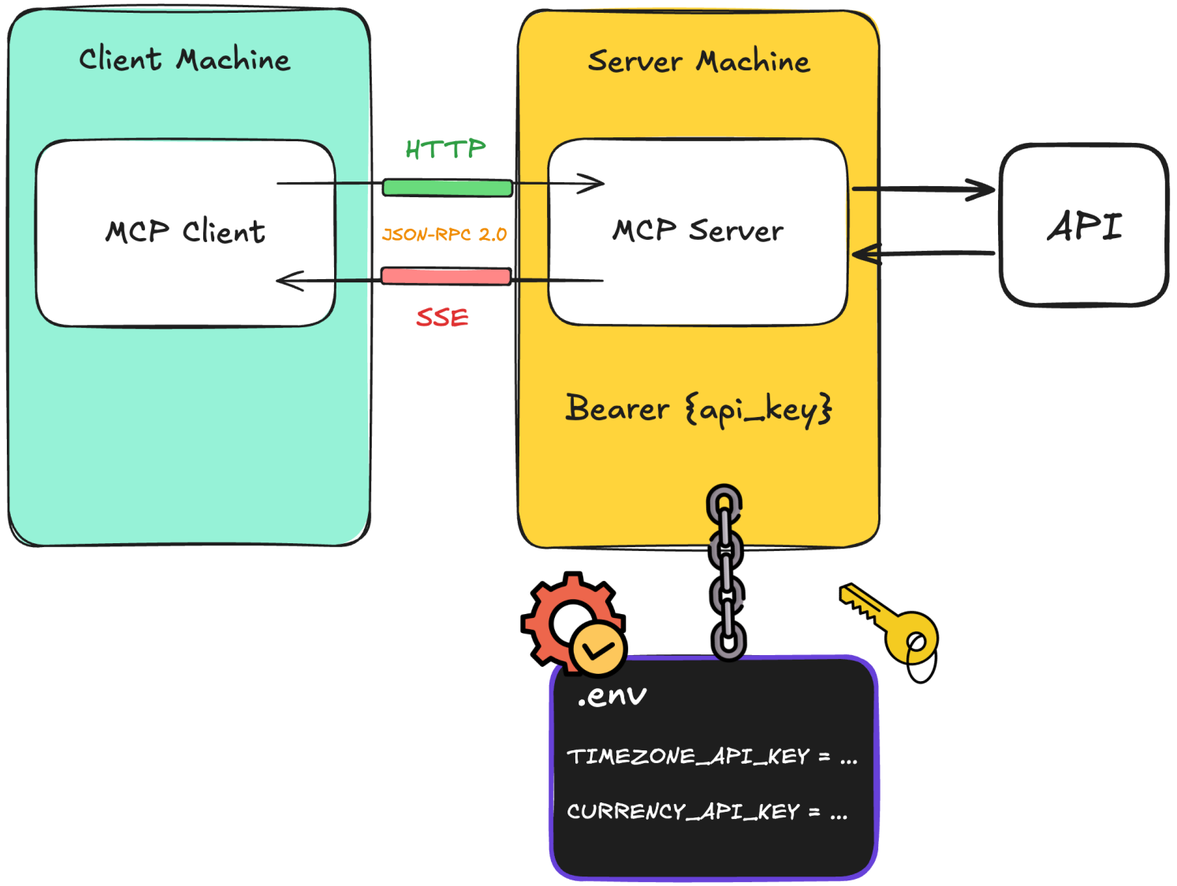
API Authentication: The Do's and Don'ts

- **API keys** → always accessed *server-side*
  - Local Host: Environment variables, `.env`
  - Remote Host: Environment variables, managed secrets

The client never sends, receives, or accesses the credentials

Hard-coded, inserted into URLs, logged

Tools argument or tool results



```python
import os
import requests

@mcp.tool()
def convert_timezone(date_time: str, from_timezone: str, to_timezone: str) -> str:
    headers = {"Content-Type": "application/json"}
    api_key = os.environ.get("TIMEZONE_API_KEY")
    if api_key:
        headers["Authorization"] = f"Bearer {api_key}"
    payload = {"dateTime": date_time, "fromTimezone": from_timezone, "toTimezone": to_timezone}
    try:
        r = requests.post("https://api.opentimezone.com/convert", json=payload, headers=headers, timeout=
        r.raise_for_status()
        data = r.json()
        converted_time = data.get("dateTime", "N/A")
        return f"Time in {to_timezone}: {converted_time}"
    except requests.exceptions.RequestException as e:
        return f"Error converting timezone: {e}"
```

Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

Third-Party MCP Servers

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

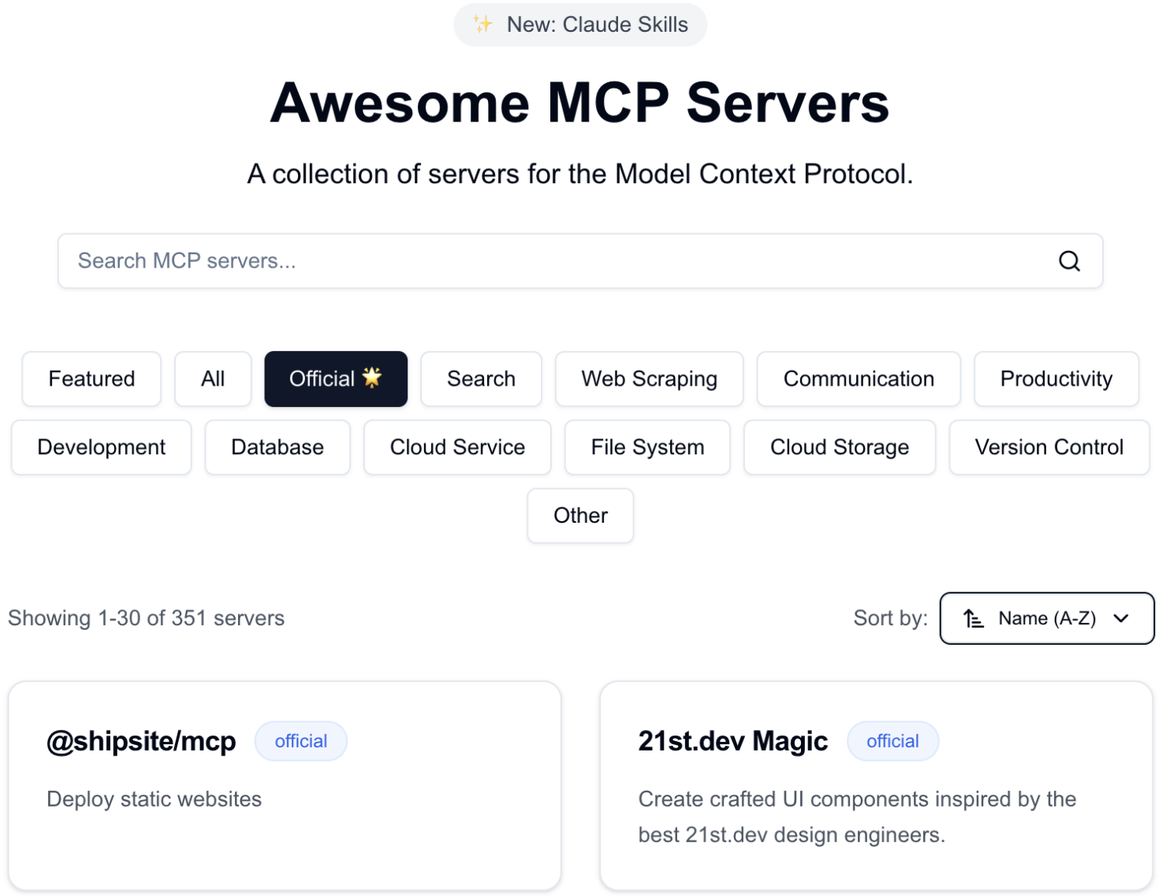


James Chapman
AI Curriculum Manager, DataCamp

datacamp

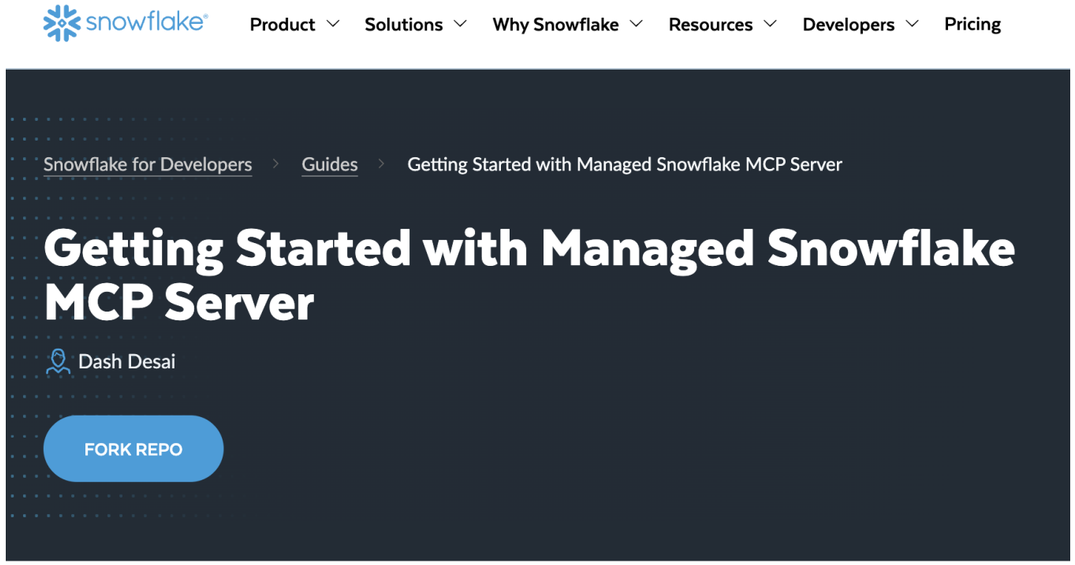
Why Use Third-Party MCP Servers?

- **Speed:** filesystem and database access without writing a server



Why Use Third-Party MCP Servers?

- **Speed:** filesystem and database access without writing a server
- **Maintained by others:** you focus on the client and application
- **Integration:** client code can be shared and reused for different MCP servers



Overview

The Snowflake MCP Server allows AI agents to securely retrieve data from Snowflake accounts without needing to deploy separate infrastructure. MCP clients discover and invoke tools, and retrieve data required for the application. The Snowflake MCP Server includes Cortex Analyst and Cortex Search as tools on the standards-based interface. It is now available with Model Context Protocol (MCP) so that AI Agents can discover and invoke tools (Cortex Analyst, Cortex Search) via a unified and standard based interface.

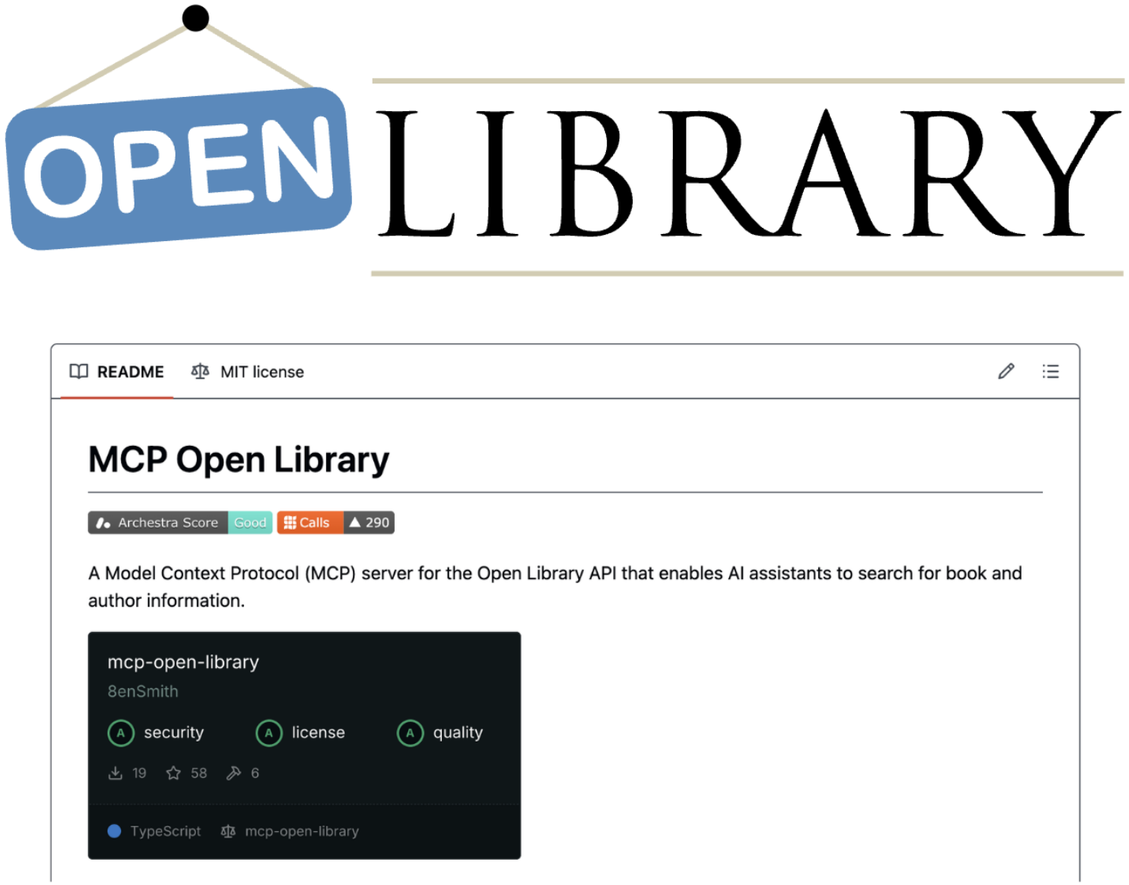
[1] https://www.snowflake.com/en/developers/guides/getting-started-with-snowflake-mcp-server/

Considerations Before Using a Third-Party Server

- **Trust:** the server runs code and may call external APIs → check these sources
- **Security:** requires careful management of private data and credentials
- **Surface area:** more tools mean more the LLM can do; be aware of what each server exposes



Open Library MCP



- **mcp-open-library:** MCP server for the Internet Archive Open Library API
- **Tools:** `get_book_by_title`, `get_authors_by_name`, `get_book_cover`, `get_book_by_id`, ...
- **No API key**

[1] https://github.com/8enSmith/mcp-open-library

Accessing the Open Library MCP

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


params = StdioServerParameters(
    command="open-library-server",
    args=[]
)
```

[1] https://github.com/8enSmith/mcp-open-library?tab=readme-ov-file#installation

Using the Open Library MCP

```
params = StdioServerParameters(command="open-library-server", args=[])

async with stdio_client(params) as (reader, writer):
    async with ClientSession(reader, writer) as session:
        await session.initialize()
        response = await session.list_tools()
        print("Tools:", [t.name for t in response.tools])
        result = await session.call_tool("get_book_by_title", {"title": "Dune"})
        text = result.content[0].text
        print("Tool Call Result:", text)
```

```
Tools: ['get_book_by_title', 'get_authors_by_name', 'get_author_info',
'get_author_photo', 'get_book_cover', 'get_book_by_id']

Tool Call Result: [
  {
    "title": "Dune",
    "authors": [
      "Frank Herbert"
    ],
    "first_publish_year": 1965,
    ...
  },
  ...
```

Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

Congratulations!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

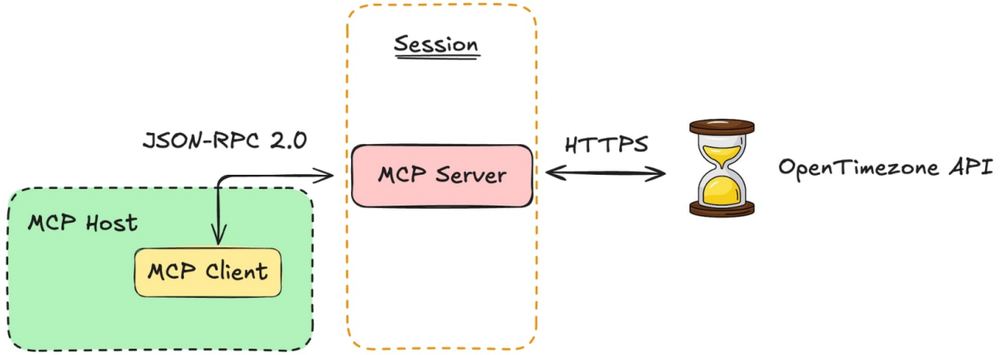


James Chapman
AI Curriculum Manager, DataCamp

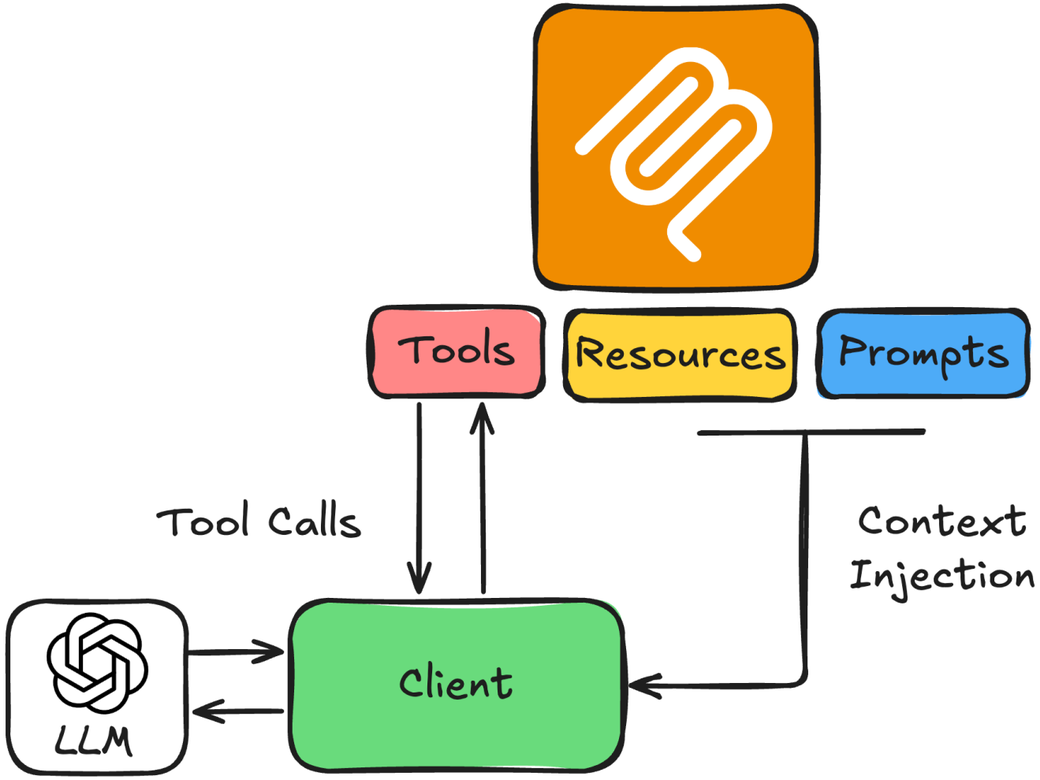
Chapter 1



- **MCP** → the "USB-C for AI" standard
- **Tools** → defined with `mcp.tool()`

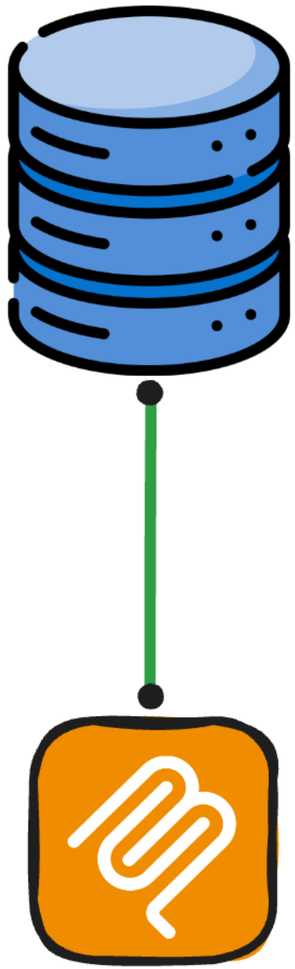


Chapter 2



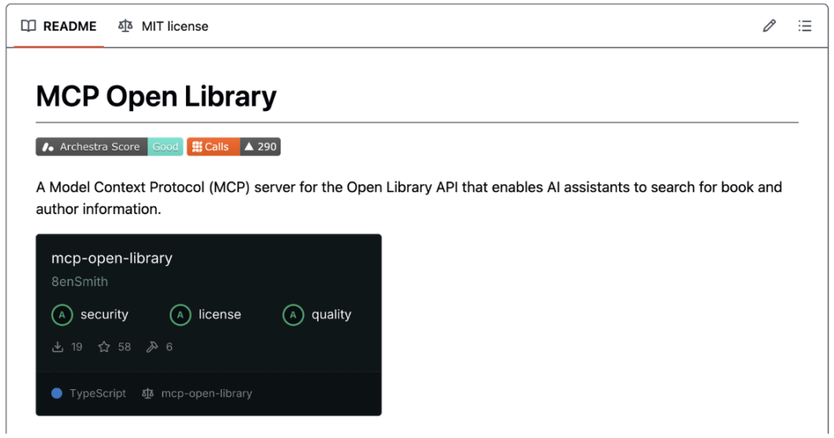
- **Resources** → read-only data for context
- **Prompts** → reusable instruction templates

Chapter 3



- **APIs** → timeouts, error handling, auth
- **Databases** → parameterized queries, security





Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)