USB-C Moment for AI:
Building MCP Servers with FastMCP and Python

Ruslan Korniichuk | 20.07.2026

ver. 1.0.0

ANTONOV AN-124
Ruslan

Source: tinyurl.com/y8pfmm67

TABLE OF CONTENTS

tinyurl.com/EuroSciPy

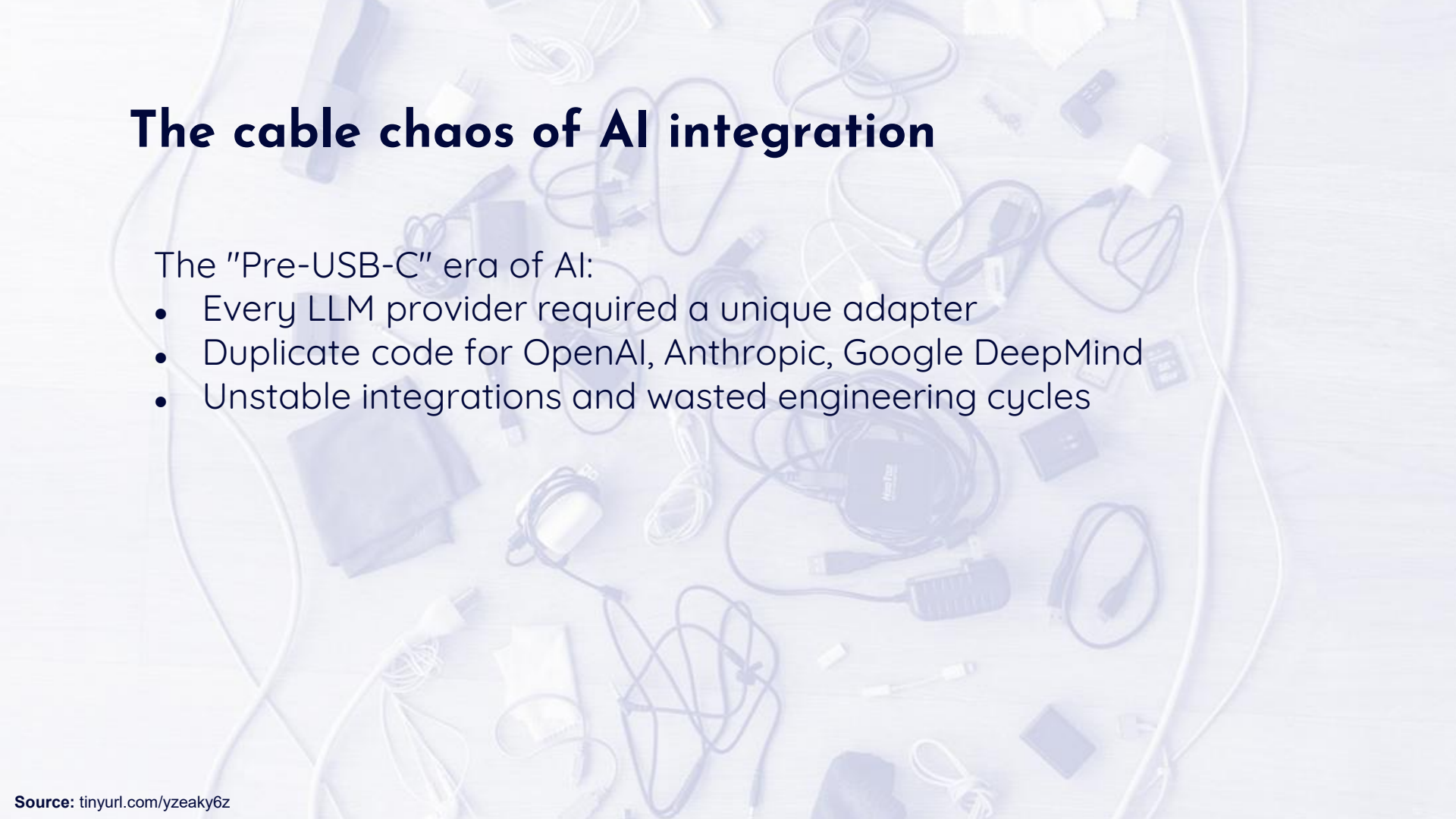
01

AI integration problem

The cable chaos of AI integration

The "Pre-USB-C" era of AI:

- Every LLM provider required a unique adapter
- Duplicate code for OpenAI, Anthropic, Google DeepMind
- Unstable integrations and wasted engineering cycles



Source: tinyurl.com/yzeaky6z

Enter the Model Context Protocol (MCP)

Origin: Created by Anthropic

Milestone: Donated to the Linux Foundation (December 2025)

The Promise: A single, universal way to connect AI models to tools, data, and services

The Industry Standard

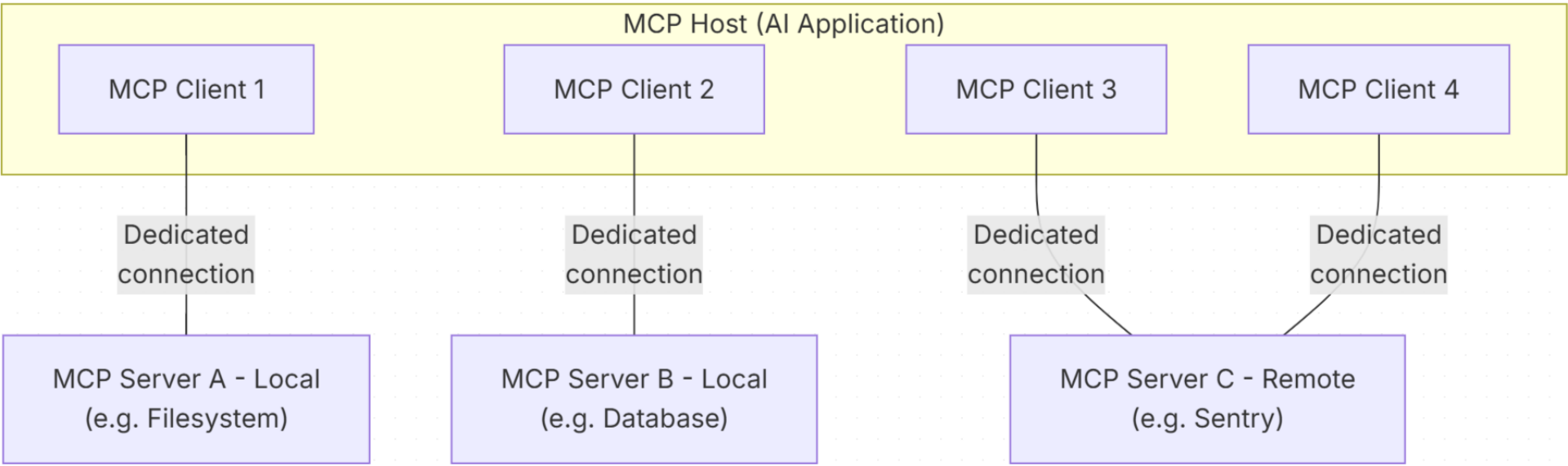
- Adopted by OpenAI, Google DeepMind, and Anthropic
- Supported by major IDEs (Google Antigravity, Cursor, VS Code)
- Thousands of developers in the open-source community
- Write once, connect anywhere

02

Protocol architecture

MCP Architecture

- **MCP Host**: AI application that coordinates and manages one or multiple MCP clients (e.g., Google Antigravity)
- **MCP Client:** Component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use (e.g., The AI Agent)
- **MCP Server:** Program that provides context to MCP clients (e.g., Python application)
- **Transport Layers:** stdio (local) or Streamable HTTP (remote)


MCP Host (AI Application)
MCP Client 1
MCP Client 2
MCP Client 3
MCP Client 4
Dedicated connection
Dedicated connection
Dedicated connection
Dedicated connection
MCP Server A - Local (e.g. Filesystem)
MCP Server B - Local (e.g. Database)
MCP Server C - Remote (e.g. Sentry)


User
AI Agent in Antigravity
MCP
FastMCP stocks server
HTTP
Yahoo Finance via yfinance

Protocol anatomy: the three primitives

- **Resources:** "Here is some data." Read-only context (e.g., file contents, database records, API responses)

- **Prompts:** "Here is how to ask." Reusable templates that help structure interactions with language models (e.g., system prompts, few-shot examples)

- **Tools:** "Take this action." Executable functions (e.g., API calls, Bash commands, file operations, database queries)

Protocol anatomy: the three primitives

- **Resources:** "Here is some data." Read-only context (e.g., file contents, database records, API responses).
  Example: `stocks://watchlist`

- **Prompts:** "Here is how to ask." Reusable templates that help structure interactions with language models (e.g., system prompts, few-shot examples).
  Example: `analyze_stock("AAPL")`

- **Tools:** "Take this action." Executable functions (e.g., API calls, Bash commands, file operations, database queries).
  Example: `get_stock_price("AAPL")`

03

FastMCP

Get Started

Welcome to FastMCP

Copy page

The fast, Pythonic way to build MCP servers, clients, and applications.

FastMCP is the standard framework for building MCP applications. The Model Context Protocol (MCP) connects LLMs to tools and data. FastMCP gives you everything you need to go from prototype to production — build servers that expose capabilities, connect clients to any MCP service, and give your tools interactive UIs:

Why FastMCP for Python?

- The FastAPI of the MCP ecosystem
- Declarative and Pythonic
- Relies on standard type hints (`int`, `str`, Pydantic models)
- Zero boilerplate: automatic JSON schema generation and auto-discovery

Create a stocks_server_min.py Python file:

```python
from fastmcp import FastMCP

mcp = FastMCP("stocks")


@mcp.tool
def ping() -> str:
    """Return a simple health-check message."""
    return "stocks MCP server is alive"


if __name__ == "__main__":
    mcp.run()
```

Run the server

```
python stocks_server_min.py
```

Run the server

```
python stocks_server_min.py
```

Create a stocks_server.py Python file:

```
"""Stock market MCP server built with FastMCP and
yfinance."""

import os

import yfinance as yf
from fastmcp import FastMCP

mcp = FastMCP("stocks")

WATCHLIST = ["AAPL", "MSFT", "GOOGL", "NVDA", "TSLA"]
```

Tool 1: get_company_info

```
@mcp.tool
def get_company_info(ticker: str) -> dict:
    """Return basic company information for the given ticker
symbol."""
    info = yf.Ticker(ticker).info
    return {
        "ticker": ticker.upper(),
        "name": info.get("longName") or
info.get("shortName"),
        "sector": info.get("sector"),
        "industry": info.get("industry"),
        "country": info.get("country"),
        "website": info.get("website"),
        "market_cap": info.get("marketCap"),
        "summary": info.get("longBusinessSummary"),
    }
```

Tool 2: get_stock_price

```python
@mcp.tool
def get_stock_price(ticker: str) -> dict:
    """Return the latest available stock price and currency
for the ticker."""
    t = yf.Ticker(ticker)
    fast = t.fast_info
    return {
        "ticker": ticker.upper(),
        "price": float(fast["last_price"]),
        "currency": fast.get("currency"),
        "previous_close": float(fast.get("previous_close")),
    }
```

Tool 3: get_stock_history

```python
@mcp.tool
def get_stock_history(ticker: str, days: int = 7) -> list[dict]:
    """Return daily OHLCV history for the last N days (default 7)."""
    period = f"{max(1, int(days))}d"
    df = yf.Ticker(ticker).history(period=period)
    df = df.reset_index()
    return [
        {
            "date": row["Date"].strftime("%Y-%m-%d"),
            "open": float(row["Open"]),
            "high": float(row["High"]),
            "low": float(row["Low"]),
            "close": float(row["Close"]),
            "volume": int(row["Volume"]),
        }
        for _, row in df.iterrows()
    ]
```

Static resource: stocks://watchlist

```
...

WATCHLIST = ["AAPL", "MSFT", "GOOGL", "NVDA", "TSLA"]

...

@mcp.resource("stocks://watchlist")
def watchlist() -> list[str]:
    """Return the default watchlist of ticker symbols."""
    return WATCHLIST
```

Resource template: stocks://{ticker}/summary

```python
@mcp.resource("stocks://{ticker}/summary")
def ticker_summary(ticker: str) -> dict:
    """Return a short summary (name, sector, latest price)
for a ticker."""
    info = get_company_info(ticker)
    price = get_stock_price(ticker)
    return {
        "ticker": ticker.upper(),
        "name": info["name"],
        "sector": info["sector"],
        "price": price["price"],
        "currency": price["currency"],
    }
```

Prompt template: analyze_stock

```python
@mcp.prompt
def analyze_stock(ticker: str) -> str:
    """Reusable prompt template that asks the agent to analyze a stock."""
    return (
        f"Analyze the stock {ticker.upper()}.\n\n"
        "Use the MCP tools to gather:\n"
        "1. Company information (get_company_info)\n"
        "2. Current price (get_stock_price)\n"
        "3. 30-day price history (get_stock_history with days=30)\n\n"
        "Then write a concise report covering: business overview, recent "
        "price trend, and one risk and one opportunity for an investor."
    )
```

Transport switch

```
if __name__ == "__main__":
    transport = os.getenv("MCP_TRANSPORT", "stdio").lower()
    if transport == "http":
        mcp.run(transport="http", host="127.0.0.1", port=8000)
    else:
        mcp.run()
```

04

Deploy and test MCP server

Option A: Local stdio with MCP Inspector

The fastest way to inspect the server is with the MCP Inspector shipped with FastMCP:

```
fastmcp dev inspector stocks_server.py
```

This command starts your server on stdio, and launches MCP Inspector in your browser.

Option A: Local stdio with MCP Inspector

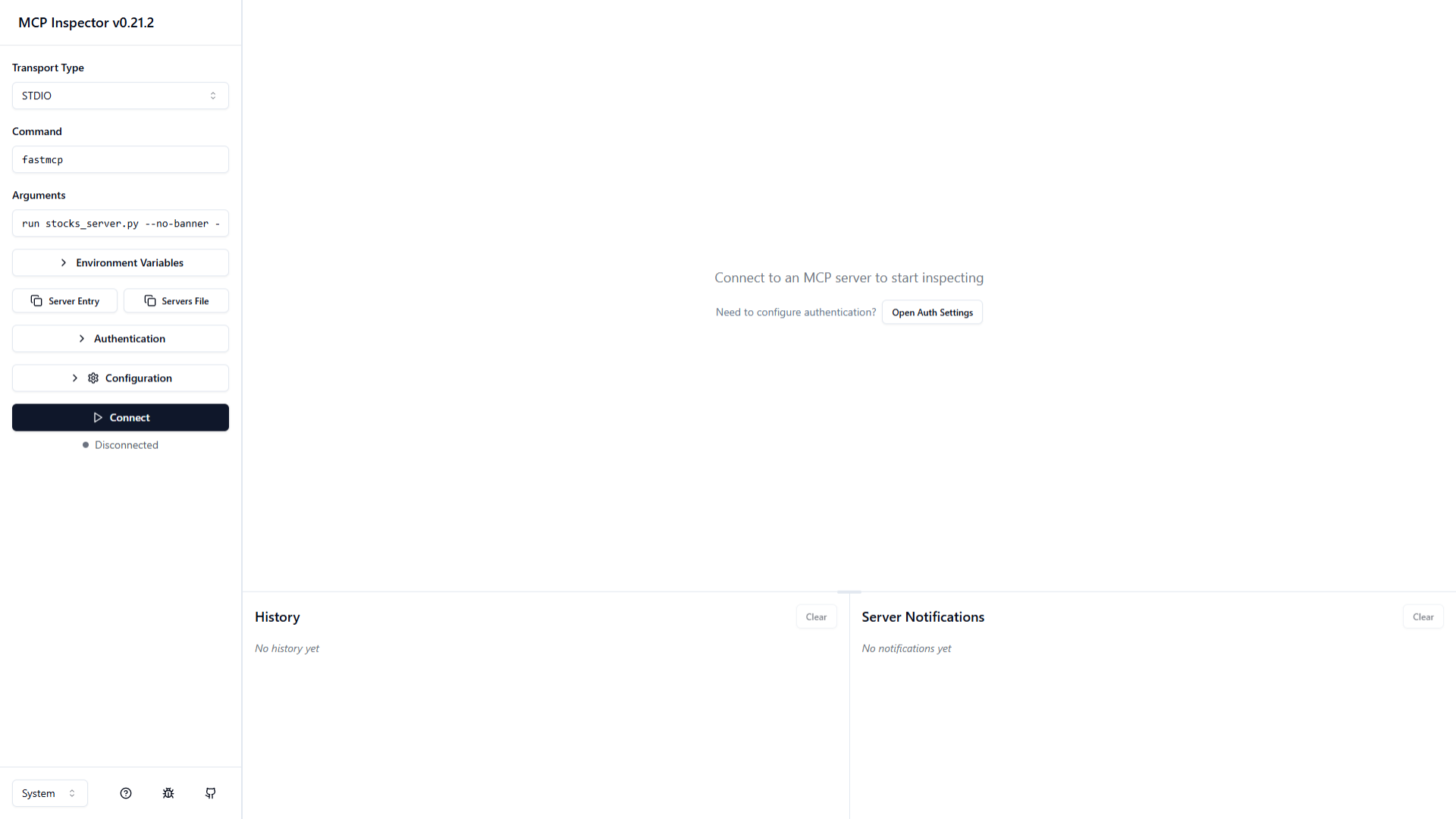
The fastest way to inspect the server is with the MCP Inspector shipped with FastMCP:

```
fastmcp dev inspector stocks_server.py
```

This command starts your server on stdio, and launches MCP Inspector in your browser.

In the Inspector you can:

- List tools, resources, and prompts that your server advertises
- Call `get_stock_price` with `ticker="AAPL"` and see the response
- Read the `stocks://watchlist` resource
- Render the `analyze_stock` prompt with `ticker="MSFT"`


MCP Inspector v0.21.2
Transport Type
STDIO
Command
fastmcp
Arguments
run stocks_server.py --no-banner -
Environment Variables
Server Entry
Servers File
Authentication
Configuration
Connect
Disconnected
Connect to an MCP server to start inspecting
Need to configure authentication?
Open Auth Settings
History
Clear
No history yet
Server Notifications
Clear
No notifications yet
System

Option B: HTTP transport

On Windows (PowerShell):

```
$env:MCP_TRANSPORT = "http"
python stocks_server.py
```

On macOS / Linux:

```
MCP_TRANSPORT=http
python stocks_server.py
```

The server now listens on `http://127.0.0.1:8000/mcp`

Option B: HTTP transport

To inspect the HTTP endpoint, start the Inspector on its own and point it at the URL:

```
npx @modelcontextprotocol/inspector
```

In the Inspector UI:

- Set `Transport Type` to `Streamable HTTP`
- Set `URL` to `http://127.0.0.1:8000/mcp`
- Click `Connect`

MCP Inspector v0.21.2

Transport Type

Streamable HTTP

URL

http://127.0.0.1:8000/mcp

Connection Type

Via Proxy

Server Entry | Servers File

Authentication

Configuration

Reconnect | Disconnect

Connected

stocks

Version: 3.4.4

Logging Level

debug

System

Resources | Prompts | Tools | Tasks | Apps | Ping | Sampling | Elicitations | Roots | Auth | Metadata

Resources

List Resources

Clear

Resource Templates

List Templates

Clear

Select a resource or template

Select a resource or template from the list to view its contents

History

Clear

2. logging/setLevel

1. initialize

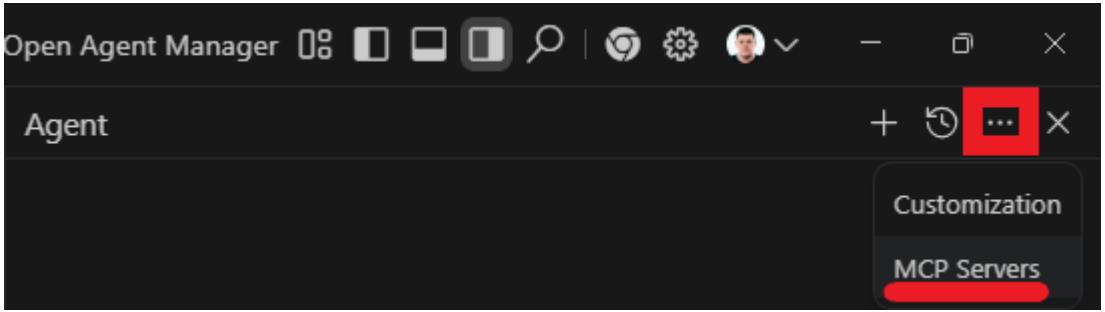
Server Notifications

Clear

No notifications yet

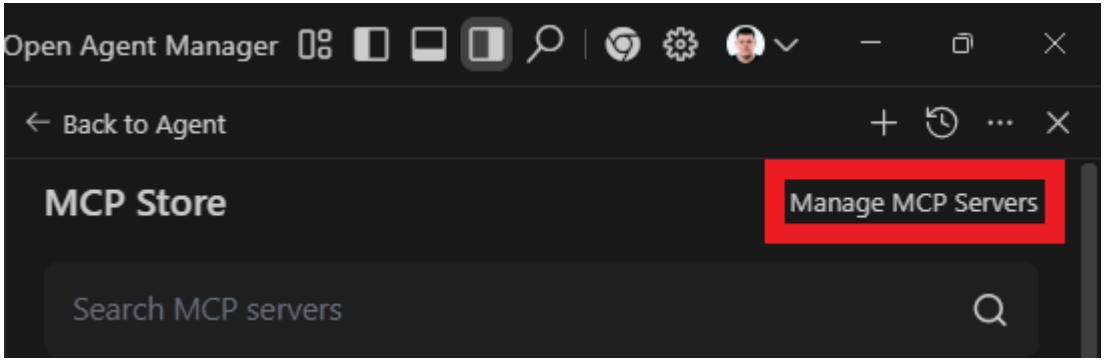
Option C: Connect the server to Antigravity

In Google Antigravity, select `Additional Options (...)` in the top right corner. Select `MCP Servers`:



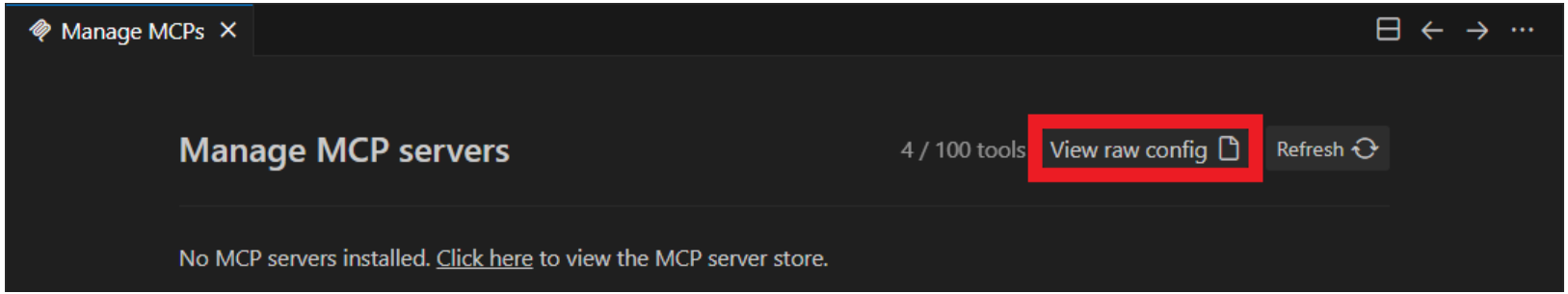
Option C: Connect the server to Antigravity

Select `Manage MCP Servers`:



Option C: Connect the server to Antigravity

Select `View raw config` in the `Manage MCPs`:



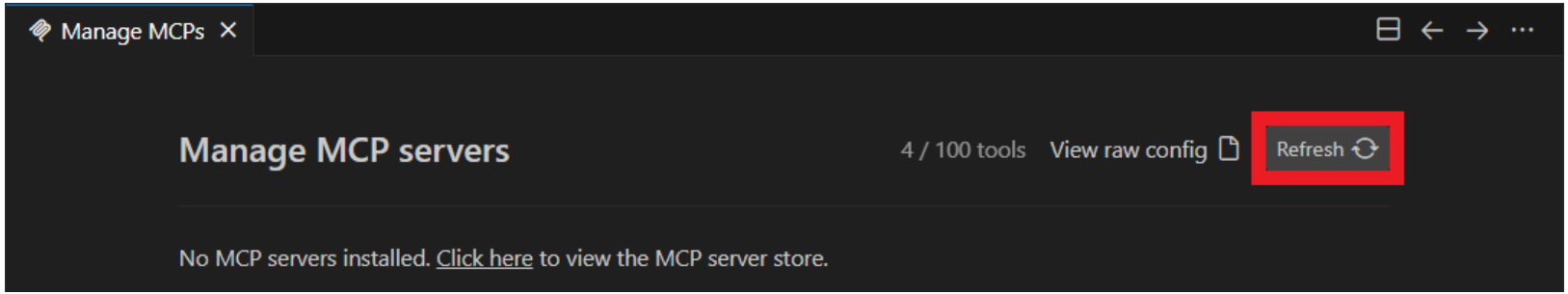
Option C: Connect the server to Antigravity

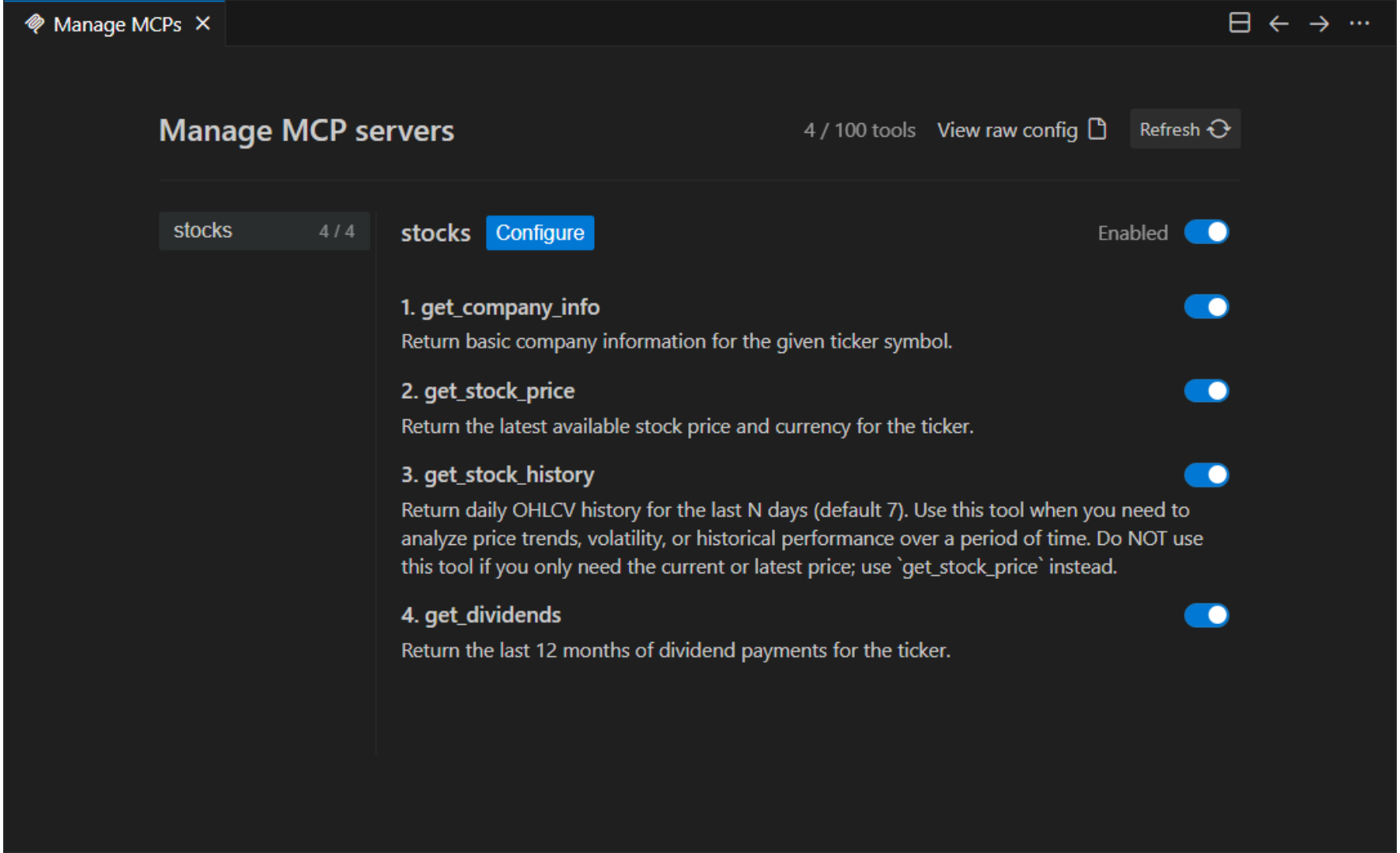
Copy and paste the following `stocks` block.

```
{
  "mcpServers": {
    "stocks": {
      "command": "C:/Users/<you>/Documents/agentic-ai-in-practice-dawts-2026/.venv/Scripts/python.exe",
      "args": ["C:/Users/<you>/Documents/agentic-ai-in-practice-dawts-2026/stocks_server.py"],
      "env": {
        "MCP_TRANSPORT": "stdio"
      }
    }
  }
}
```

Option C: Connect the server to Antigravity

Click the `Refresh` 🔄 button in the Manage MCPs:




Manage MCPs
Manage MCP servers
4 / 100 tools
View raw config
Refresh
stocks 4 / 4
stocks
Configure
Enabled
1. get_company_info
Return basic company information for the given ticker symbol.
2. get_stock_price
Return the latest available stock price and currency for the ticker.
3. get_stock_history
Return daily OHLCV history for the last N days (default 7). Use this tool when you need to analyze price trends, volatility, or historical performance over a period of time. Do NOT use this tool if you only need the current or latest price; use `get_stock_price` instead.
4. get_dividends
Return the last 12 months of dividend payments for the ticker.

Test MCP server in Google Antigravity

In the Editor Agent chat, ask a natural-language question:

Give me basic company information about Microsoft.

Antigravity will:

- Discover your MCP server
- Call `get_company_info("MSFT")`

Test MCP server in Google Antigravity

In the Editor Agent chat, ask a natural-language questions:

> *What is the current price of Apple stock, and how did it move over the last 5 trading days?*

Antigravity will:

- Discover your MCP server
- Call `get_stock_price("AAPL")` and `get_stock_history("AAPL", days=5)`
- Summarize the result

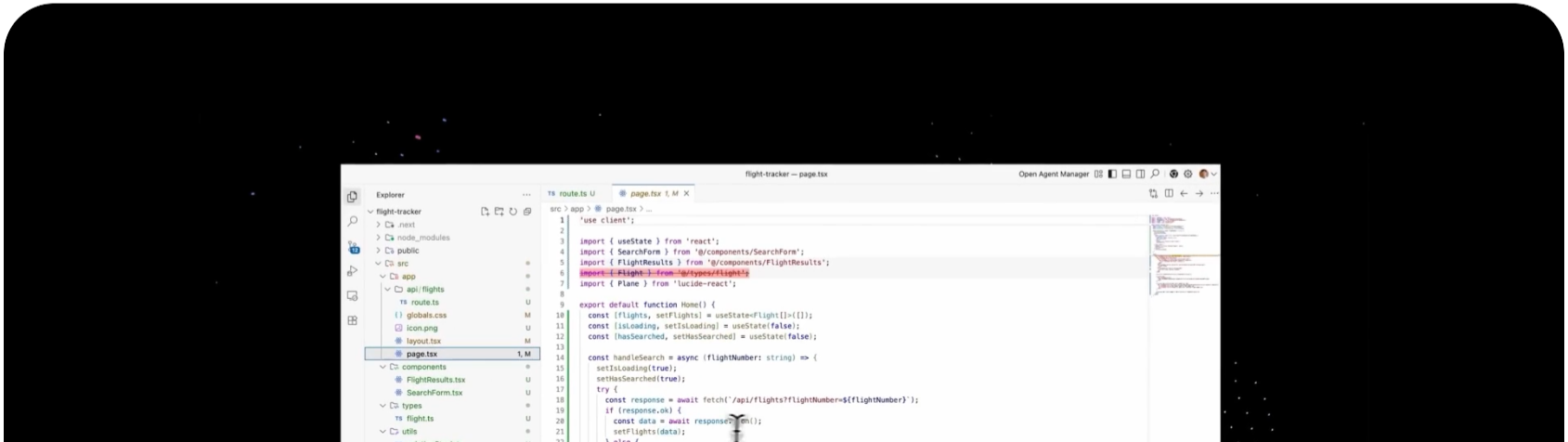
05

MCP demo with Google Antigravity

Antigravity IDE

Google Antigravity's Editor view offers tab autocompletion, natural language code commands, and a configurable, and context-aware configurable agent.

Download



06

Closing slides

Get started

What is the Model Context Protocol (MCP)?

Copy page

MCP (Model Context Protocol) is an open-source standard for connecting AI applications to external systems.

Using MCP, AI applications like Claude or ChatGPT can connect to data sources (e.g. local files, databases), tools (e.g. search engines, calculators) and workflows (e.g. specialized prompts)—enabling them to access key information and perform tasks.

Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect electronic devices, MCP provides a standardized way to connect AI applications to external systems.

Ask a question...

Ctrl+I

agentic-ai-in-practice-dawts-2026

Public

Pin | Watch 0 | Fork 1 | Star 3

main | 1 Branch | 0 Tags | Go to file T | Add file | Code

korniichuk Update readme — 483c1d1 · 3 months ago — 21 Commits

img	Update readme	3 months ago
.gitignore	Initial commit	3 months ago
LICENSE	Initial commit	3 months ago
README.md	Update readme	3 months ago
mcp.example.json	Fix mcp.example.json	3 months ago
requirements.txt	Initial commit	3 months ago
stocks_server.py	Initial commit	3 months ago
stocks_server_min.py	Initial commit	3 months ago

README | Unlicense license

Agentic AI in Practice: Google Antigravity, FastMCP, and Python

About

Agentic AI in Practice: Google Antigravity, FastMCP, and Python

- Readme
- Unlicense license
- Activity
- 3 stars
- 0 watching
- 1 fork

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 1

korniichuk Ruslan Korniichuk

RUSLAN KORNIICHUK

Data and AI Platform Architect in Poland

Hire me

Solid experience of 11+ years in IT. Former Lead Engineer and Architect at Fortune 500 companies. Core areas: modern data platforms and cloud computing.

Public speaker at European SME Congress, Warsaw IT Days, Data Science Summit, Linux Autumn, and EuroPython. Taught information system design and server-side technologies at the University of Silesia.

The honest, reliable, and hardworking person with always a keen eye for work details. **Click the button above to hire me**

DONATE TO UKRAINE ARMY FUNDRAISER

ARMY ASSISTANCE | SUPPORT THE FUND

How will my money be used?

All funds raised to help the army will be spent exclusively on purchases for the Armed Forces of Ukraine and other components of the Defense Forces.

Payment method

PAYMENT BY CARD | TRANSFERS IN UKRAINE | SWIFT TRANSFERS | CRYPTOCURRENCY

Regularity

ONE TIME | SUBSCRIPTION EVEN BETTER

Amount of donation*

€ 10 EUR, €

+100 EUR +200 EUR +500 EUR +1000 EUR

Email*

example@example.com

Do you want to receive emails from "Come Back Alive": analytics, news, and reports on how your support saves lives and strengthens the Ukrainian Defense Forces?*

○ Agree ○ Disagree

DONATE

Payment secured

tinyurl.com/EuroSciPy