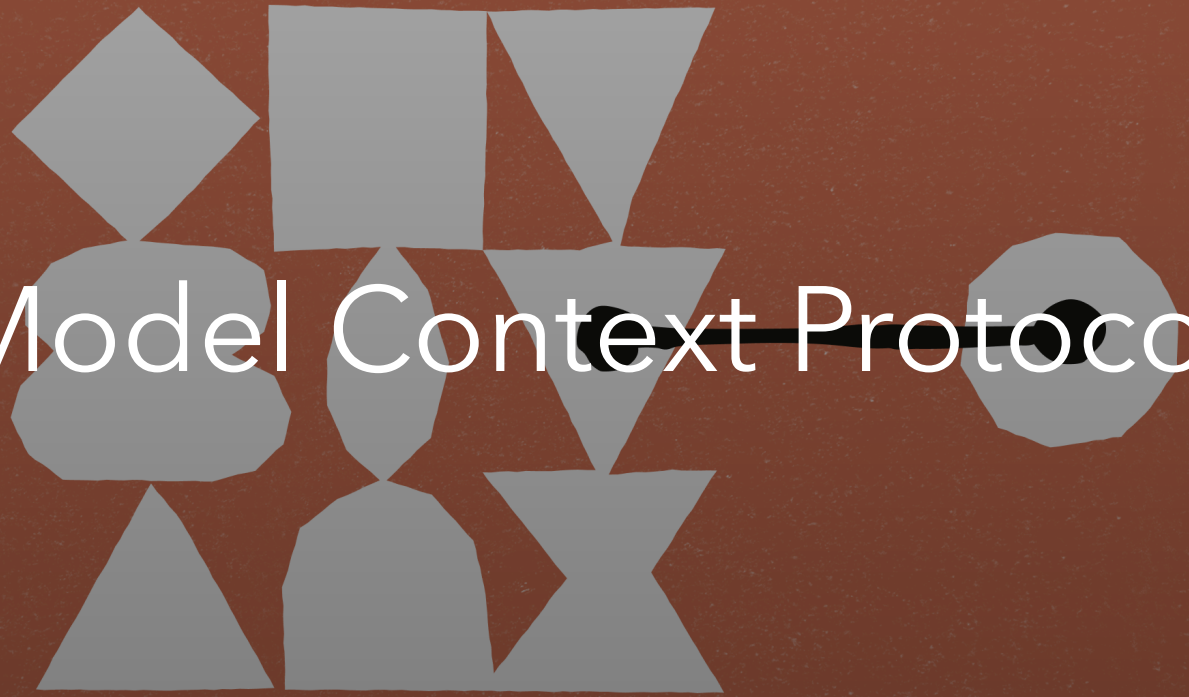

Model Context Protocol

The Integration Issue

Fragmented AI Landscape

Multiple AI providers, each with unique APIs, authentication methods, and data formats, making integration complex and time-consuming.

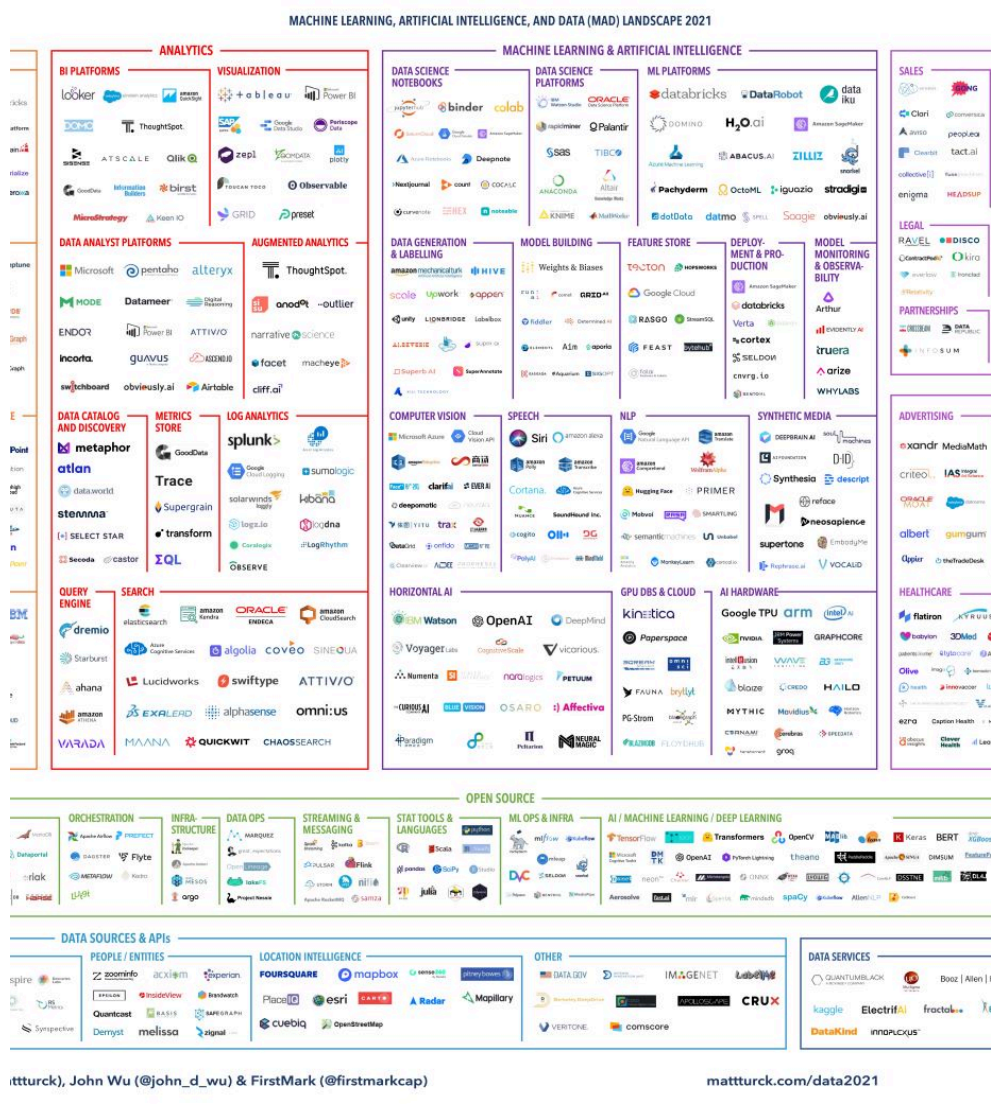
Complex Integration

Diverse tooling, context management, and unique response handling create development overhead for custom implementations.

Security Concerns

Data privacy, access control, and secure communication channels pose critical challenges when integrating AI capabilities.

Growing Number of AI Services	Increasing application complexity	Development time and costs rise



Enter MCP

Model Context Protocol

The missing standard for AI integration

by Anthropic 

What is MCP?

MCP is an open source protocol that enables the seamless integration AI apps and agents with tools and data sources

REST API in Web Development

Connects web applications to data and services

=

MCP in AI Ecosystem

Connects AI applications to tools and data

Open Standard

An open protocol designed for community adoption and extension

Tool Integration

Securely connect AI with your application's tools and capabilities

Data Access

Controlled access to application resources and contextual data

Growing MCP Support

Official MCP Servers Repository

11.4k+ stars · 1.2k+ forks · 200+ contributors

Collection of reference MCP servers.

GitHub	Jira	PostgreSQL
Weather	Google	FileSystem

Community-Created Servers

500+ servers in the awesome-mcp-servers list

JetBrains	Bluesky
Binance	Box

README | Code of conduct | MIT license | Security

🌟 Reference Servers

These servers aim to demonstrate MCP features and the TypeScript and Python SDKs.

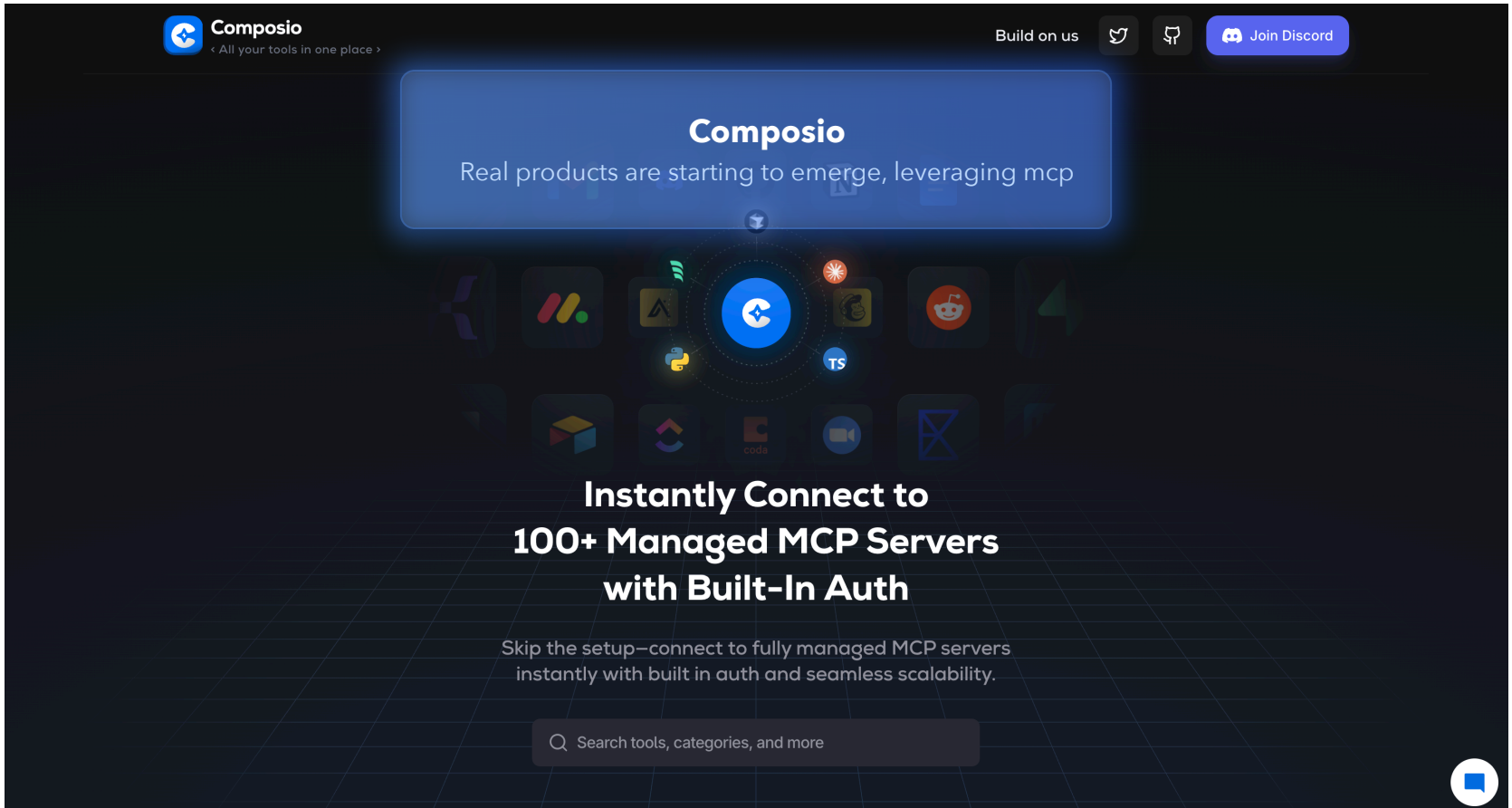
- **AWS KB Retrieval** - Retrieval from AWS Knowledge Base using Bedrock Agent Runtime
- **Brave Search** - Web and local search using Brave's Search API
- **EverArt** - AI image generation using various models
- **Everything** - Reference / test server with prompts, resources, and tools
- **Fetch** - Web content fetching and conversion for efficient LLM usage
- **Filesystem** - Secure file operations with configurable access controls
- **Git** - Tools to read, search, and manipulate Git repositories
- **GitHub** - Repository management, file operations, and GitHub API integration
- **GitLab** - GitLab API, enabling project management
- **Google Drive** - File access and search capabilities for Google Drive
- **Google Maps** - Location services, directions, and place details
- **Memory** - Knowledge graph-based persistent memory system
- **PostgreSQL** - Read-only database access with schema inspection
- **Puppeteer** - Browser automation and web scraping
- **Sentry** - Retrieving and analyzing issues from Sentry.io
- **Sequential Thinking** - Dynamic and reflective problem-solving through thought sequences
- **Slack** - Channel management and messaging capabilities
- **Sqlite** - Database interaction and business intelligence capabilities
- **Time** - Time and timezone conversion capabilities

🤝 Third-Party Servers

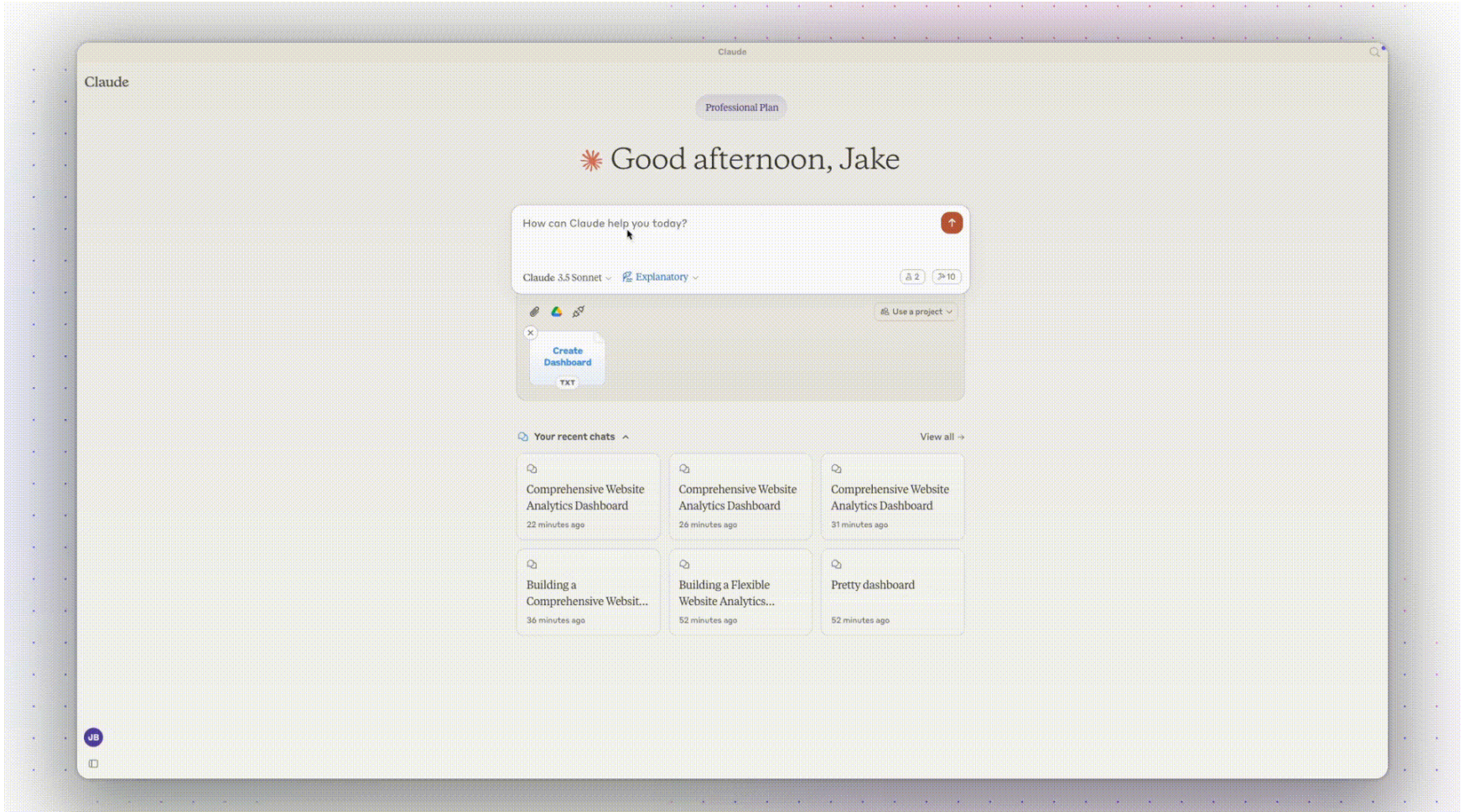
🎖️ Official Integrations

Official integrations are maintained by companies building production ready MCP servers for their

MCP Flow in Composio



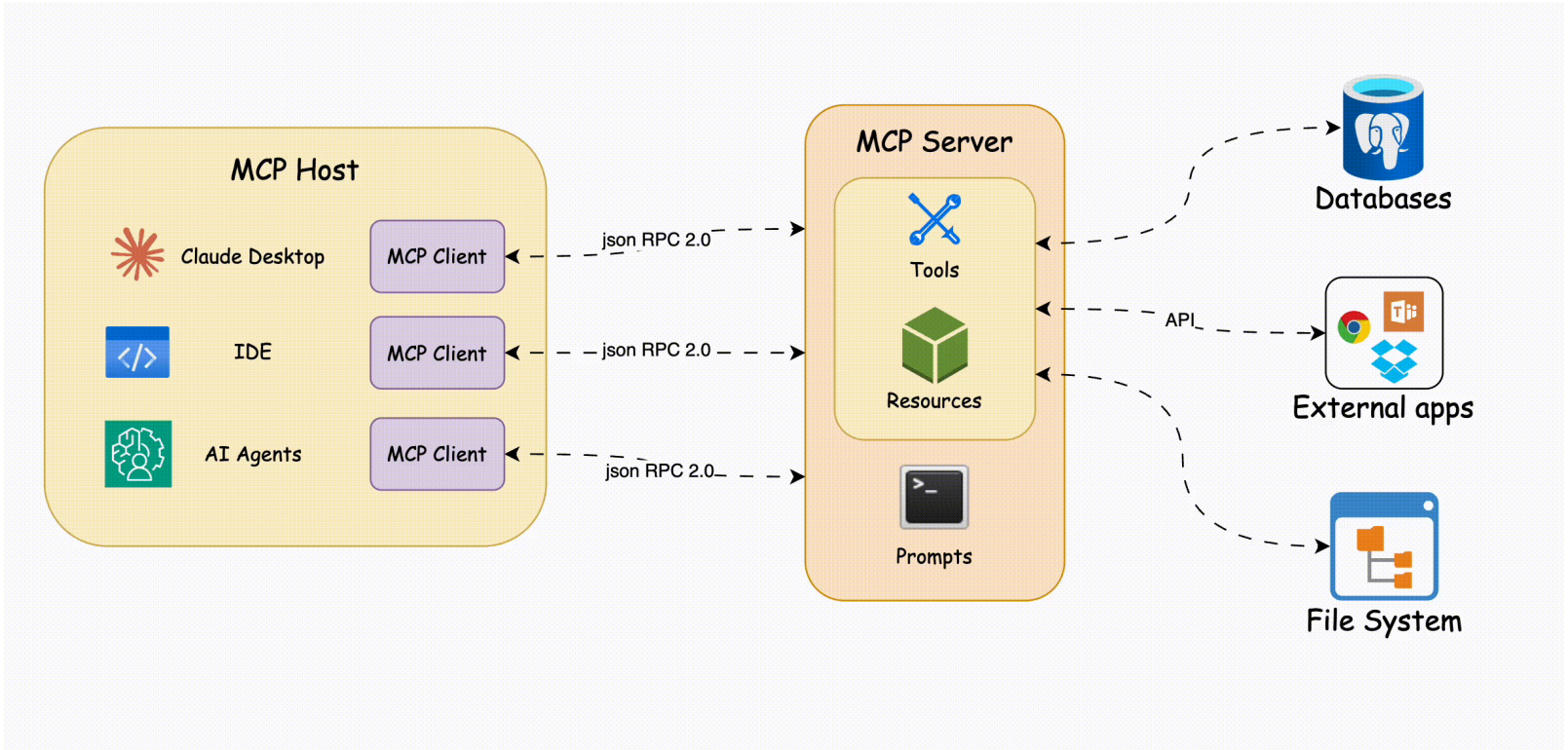
What Can an MCP do?



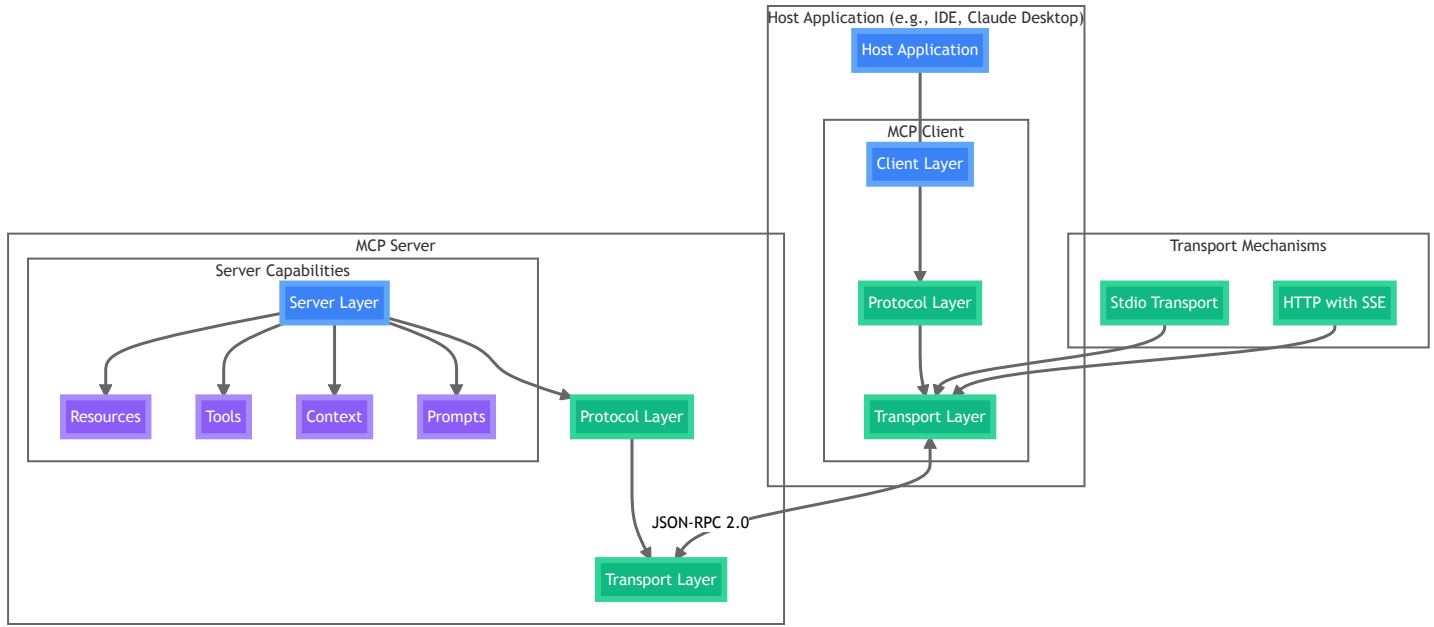
Easy

Setting Up an MCP Server in Claude Desktop

MCPs are not just for Claude



Core Architecture



Host Layer
LLM Applications that want to access data through MCP

Client Layer
Maintains and manages 1:1 connections to MCP Servers

Server Layer
Provides data, tools and prompts to Clients

MCP Client is not just an Interface

It is a Fully Featured Middleware

MCP Client

Host App → LLM

Full-featured middleware that orchestrates communication, manages context, executes tools, and maintains session state

LLM → MCP Servers

Initialization

- Session lifecycle
- API configuration
- Resource management

Server Connections

- Multiple server types
- Compatibility checks
- Communication channels

Query Processing

- Conversation context
- Response handling
- Tool call execution

Interface Management

- Command-line interface
- Input/output handling
- Program flow control

Error Management

- Resource cleanup
- Connection issues
- Graceful shutdown

Server Key Components

MODEL CONTROLLED

Tools

Functions invoked by the model

- Retrieve / search
- Send a message
- Update DB records

APPLICATION CONTROLLED

Resources

Data exposed to the application

- Files
- Database Records
- API Responses

USER CONTROLLED

Prompts

Pre-defined templates for AI interactions

- Document Q&A
- Transcript Summary
- Workflow Automation

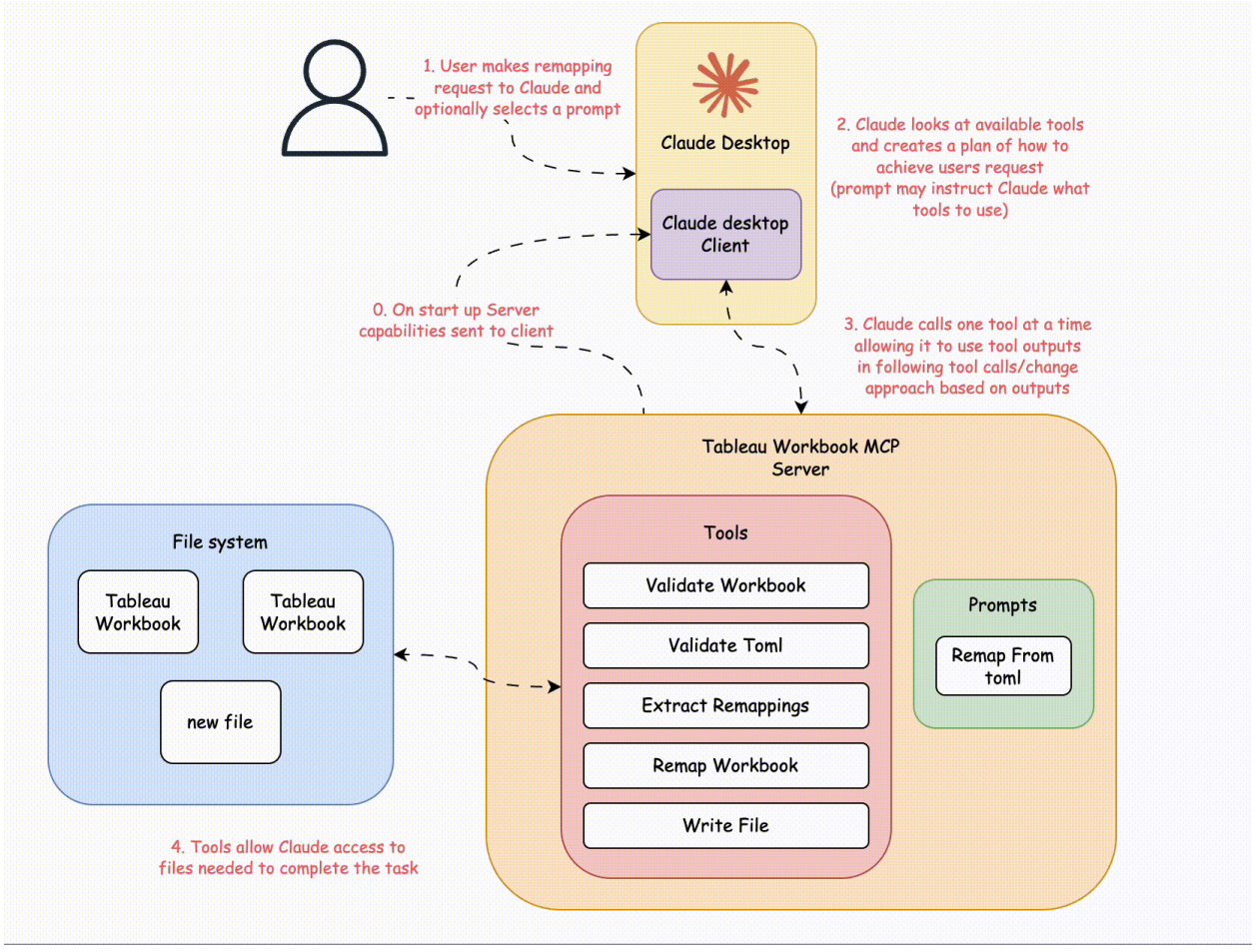
Protocol Layer

- Message Framing & Routing
- Request/Response Management
- Communication Patterns
- Versioned Protocol Schemas

Transport Layer

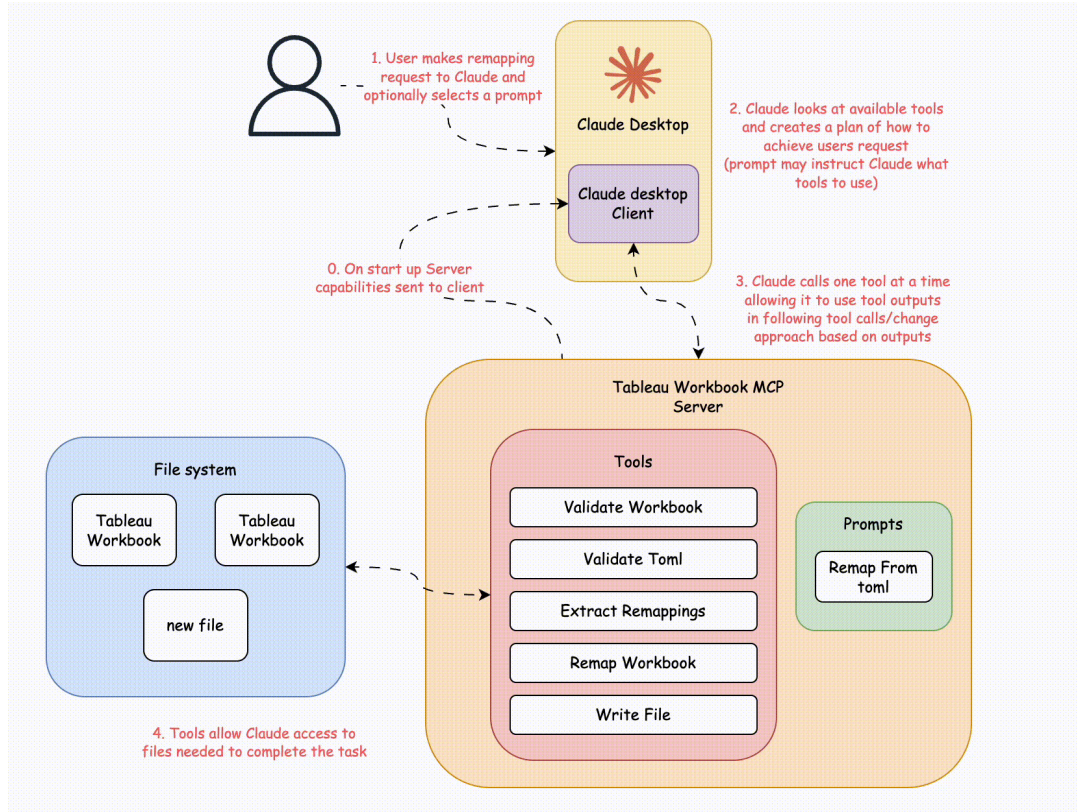
- JSON-RPC 2.0 Wire Format
- Protocol Message Transmission
- Multiple Transport Methods
- Extensible Channel System

MCP Flow


1. User makes remapping request to Claude and optionally selects a prompt
Claude Desktop
Claude desktop Client
2. Claude looks at available tools and creates a plan of how to achieve users request (prompt may instruct Claude what tools to use)
0. On start up Server capabilities sent to client
3. Claude calls one tool at a time allowing it to use tool outputs in following tool calls/change approach based on outputs
Tableau Workbook MCP Server
Tools
Validate Workbook
Validate Toml
Extract Remappings
Remap Workbook
Write File
Prompts
Remap From toml
File system
Tableau Workbook
Tableau Workbook
new file
4. Tools allow Claude access to files needed to complete the task

Example MCP Flow

- Claude determines which tools to call and when based on user questions and tool descriptions
- Tools don't call each other; they're independent capabilities invoked by the model
- Results from tools feed back to the model which are used to inform next steps in the workflow
- Claude decides how to provide results from tools to the user depending on context and result type



MCP in Action: Demo

How to Build Your Own Custom MCP Servers

Implementing MCP Tools

1. Define Tool Schema

```python
@app.list_tools()
async def list_tools() -> list[types.Tool]:
    return [
        types.Tool(
            name="calculate_sum",
            description="Add two numbers together",
            inputSchema={
                "type": "object",
                "properties": {
                    "a": {"type": "number"},
                    "b": {"type": "number"}
                },
                "required": ["a", "b"]
            }
        )
    ]
```

2. Implement Tool Handler

```python
@app.call_tool()
async def call_tool(
    name: str,
    arguments: dict
) -> list[types.TextContent | types.ImageContent | type
    if name == "calculate_sum":
        a = arguments["a"]
        b = arguments["b"]
        result = a + b
        return [types.TextContent(type="text", text=sti
    raise ValueError(f"Tool not found: {name}")
```

Implementing Prompts

1. Define Prompt Template

```
@app.list_prompts()
async def list_prompts():
    return [
        # ... existing prompts ...
        {
            "name": "Your Prompt Name",
            "description": "Your prompt description
            "arguments": [
                {
                    "name": "Parameter Name 1",
                    "description": "Parameter descr
                    "required": True/False
                },
                {
                    "name": "Parameter Name 2",
                    "description": "Parameter descr
                    "required": True/False
```

2. Implement Prompt Handler

```
@app.get_prompt()
async def get_prompt(name: str, arguments: Any):
# ... existing prompts ...
    if name == "Your Prompt Name":
        return {
            "messages": [
                {
                    "role": "user",
                    "content": {
                        "type": "text",
                        "text": f"Your prompt template
                    }
                }
            ]
        }
```

Sampling in MCP

Enabling sophisticated agentic behaviors while maintaining security and privacy

What is MCP Sampling?

- Allows Servers to request LLM completions through Client
- Human-in-the-loop design
- Contextual awareness with controlled data sharing
- Conversation history, Model preferences and prompts

Agentic Capabilities

- Reading and analyzing resources with context awareness
- Making intelligent decisions based on contextual data
- Generating structured data and formatted responses
- Handling multi-step tasks with interactive assistance

Sampling Flow

Server Requests Completion → Client Reviews Request → LLM Generates Response → Client Reviews Output → Result Returns to Server

The Future of MCP

THE POWER OF MCP IS ONLY GOING TO GROW

The current development road map looks like:

Remote MCP Support

- Authentication & Authorization with OAuth 2.0
- Service Discovery for remote connections
- Stateless operations for serverless environments

Advanced Agent Support

- Hierarchical agent systems with namespacing
- Interactive workflows with permission management
- Streaming results for real-time agent operations

Distribution & Discovery
Standardized packaging and server registry

Additional Modalities
Expanding beyond text to audio, video, and more

Community-Led Standards
Open governance and standardization initiatives