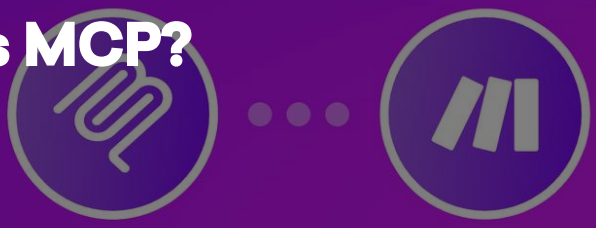
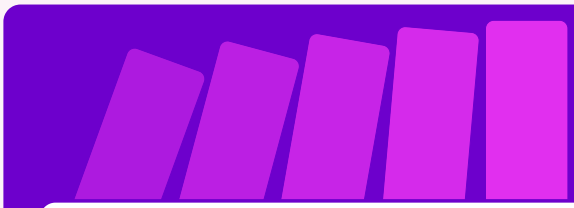
Unit 1 - What is MCP?



UNIT 1 - WHAT IS MCP?

- 1.1 Unit Introduction
- 1.2 MxN integration problem
- 1.3 MCP
- 1.4 MCP architecture
- 1.5 MCP protocol
- 1.6 MCP lifecycle
- 1.7 Wrap up



Unit 1
What is MCP?

1.1 Unit Introduction

Welcome to the first unit of the MCP course!

You've probably heard about Model Context Protocol (MCP), the protocol that lets AI systems connect to external tools and data more easily. In this unit, you'll take a closer look at what MCP is and how it works.

You will learn:

- what MCP is and the problem it solves
- the main components of MCP
- how these components work together

Let's dive in!

Continue to 1.2: MxN integration problem

1.2 MxN integration problem

For sure you're familiar with **APIs** and how to use them to **connect to third-party applications**. And if you have worked with more than one API, you have likely noticed that **each one is different**. To make a request, you need to read the API documentation, figure out how it works, and then build a request specific to that API.

Now, translate this to the world of AI agents. An AI agent needs tools to complete its tasks.

Tools let the AI agent use services from third-party applications, so it can perform specific actions automatically. The AI agent sends requests to these tools **through their API**. But since each API has its own documentation, rules, and outputs, you have to **integrate each tool one by one**.

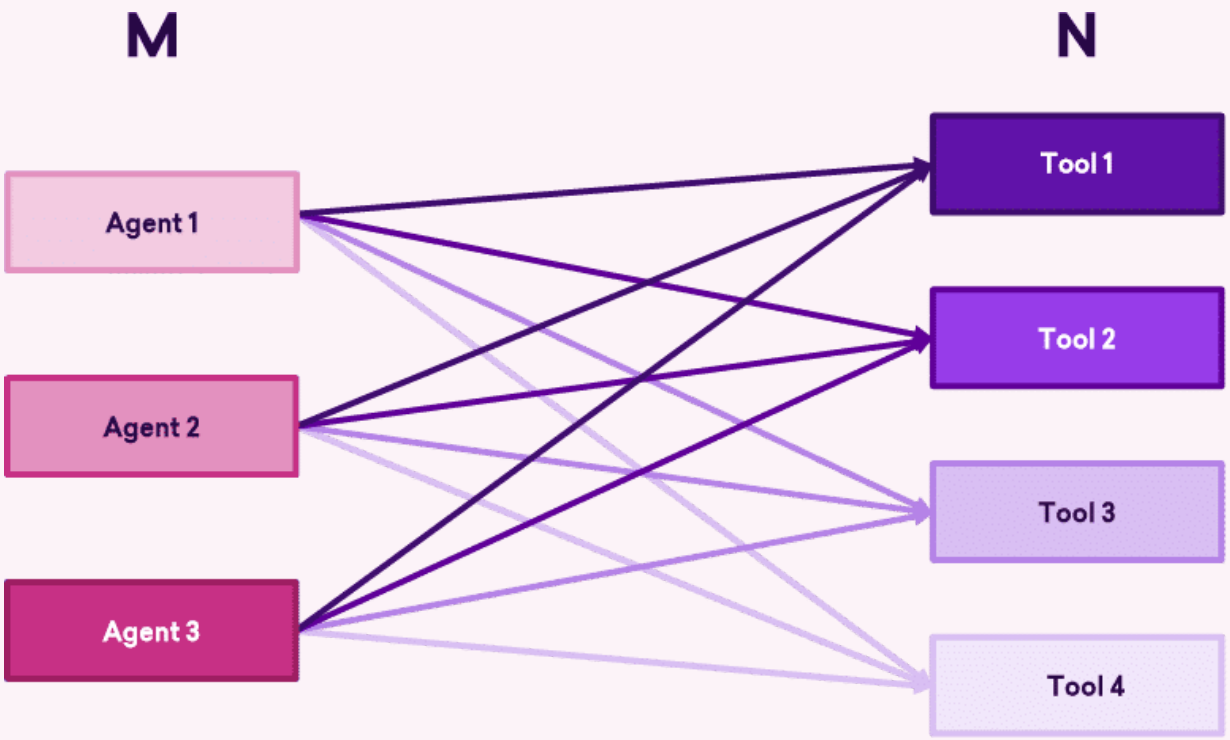
Imagine the headache if you have more than one AI agent. And this **does not even count the extra work** required when APIs change: keeping up with updates, handling new features or endpoints, and managing authentication.

This is the famous MxN integration problem.

📌 Definition

Integration means connecting different tools or systems so they can work together and share information.

If you have **M** AI agents and **N** tools, you need to **set up each connection individually because the approach is not standardized**. It is doable, but inefficient, costly, and hard for developers to maintain.



If you need a recap on AI agents, check out the Automation to AI Agents: Foundation learning path. There you will find all the information you need to know to be able to follow this course.

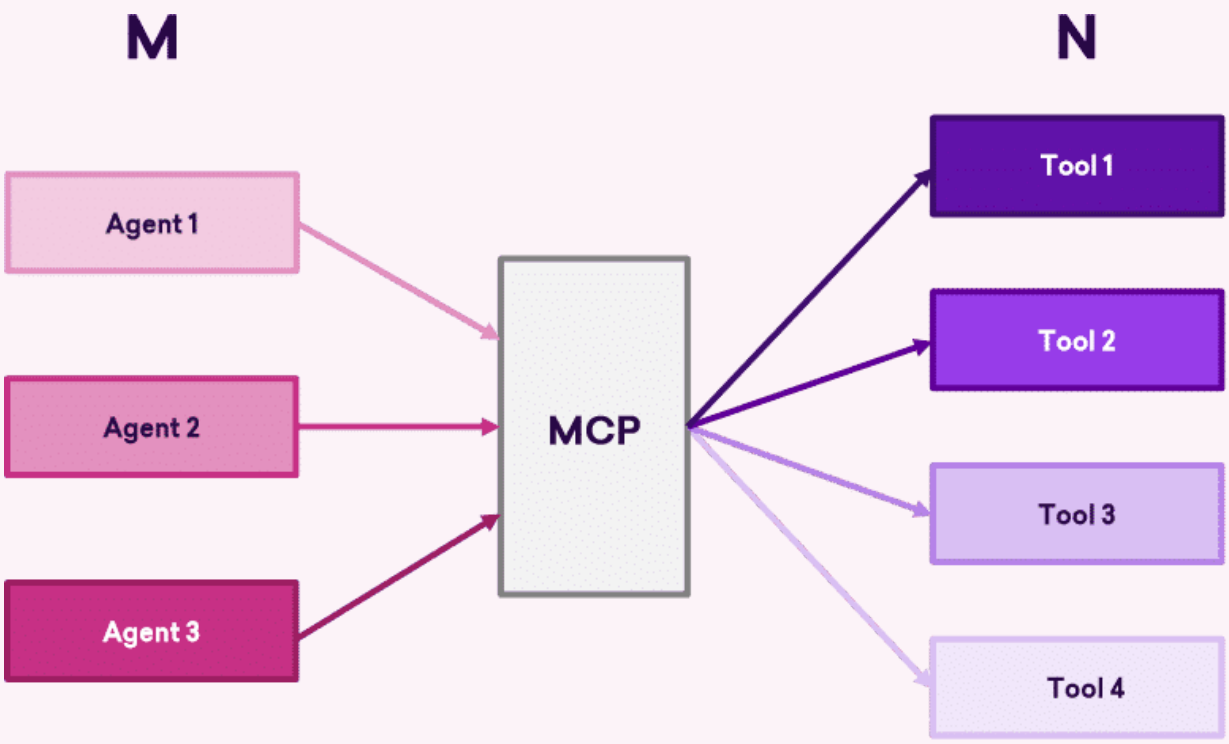
Imagine a world where you could use the same protocol, **a standard way for AI applications and tools to share information**, for all your

applications.

No more sleepless nights reading API documentation, struggling with misbehaving query parameters, or crying over a failed GET request.

This is the world of **MCP**. Exciting times ahead!

MCP (Model Context Protocol) turns integrations into a much simpler process.



Instead of every AI app needing a custom connection to every tool or data source, **MCP provides a standard way to connect.**

Each AI agent only needs to implement MCP once to know how to use any tool that provides MCP. Each tool only needs to implement the MCP server, so any AI agent can access it.

This makes integrations easier to build and much simpler to maintain.

> MCP allows **AI models, AI agents, and orchestration tools for automation to connect efficiently with external tools, data, or services**.

Here are some examples:

Click each one to learn more.

AI MODELS	AI AGENTS	ORCHESTRATION TOOLS FOR AUTOMATION

e.g., Claude and ChatGPT

MCP allows AI to **access specific data and tools**.

> Example: Ask Claude to search your company's knowledge base or query your customer database directly from the chat.

AI MODELS	AI AGENTS	ORCHESTRATION TOOLS FOR AUTOMATION

e.g., Make AI agent

MCP allows AI agents to **use the tools they need to complete their tasks autonomously**. With MCP, the connection is easy, but you're stuck with the tool descriptions that the third-party app provides, you can't customize them like you can with module tools in Make.

> Example: An AI agent that monitors your inbox, reads attachments via MCP, and automatically files them in the right folders.

AI MODELS	AI AGENTS	ORCHESTRATION TOOLS FOR AUTOMATION

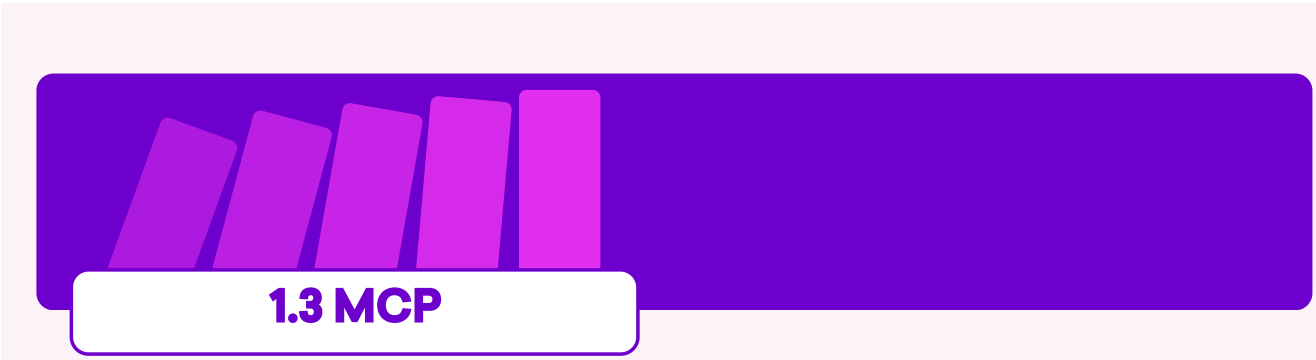
e.g., Make

MCP in Make allows you to **define which tools to call in a goal-oriented way**: you describe what you want to achieve, and the MCP client automatically selects and executes the appropriate tool, rather than manually configuring specific API endpoints for each task. You can **specify which tools** you want your scenario to be able to call and which ones not, giving you control over what actions the AI can perform.

> Example: Instead of configuring separate Asana API calls for creating tasks, updating statuses, or adding comments, you simply tell the MCP client update the project based on this data and it automatically chooses the right action based on the context.

You will practice using MCP in Make in the next units. Now let's look in more detail at how MCP is built and how it works.

Continue to 1.3: MCP

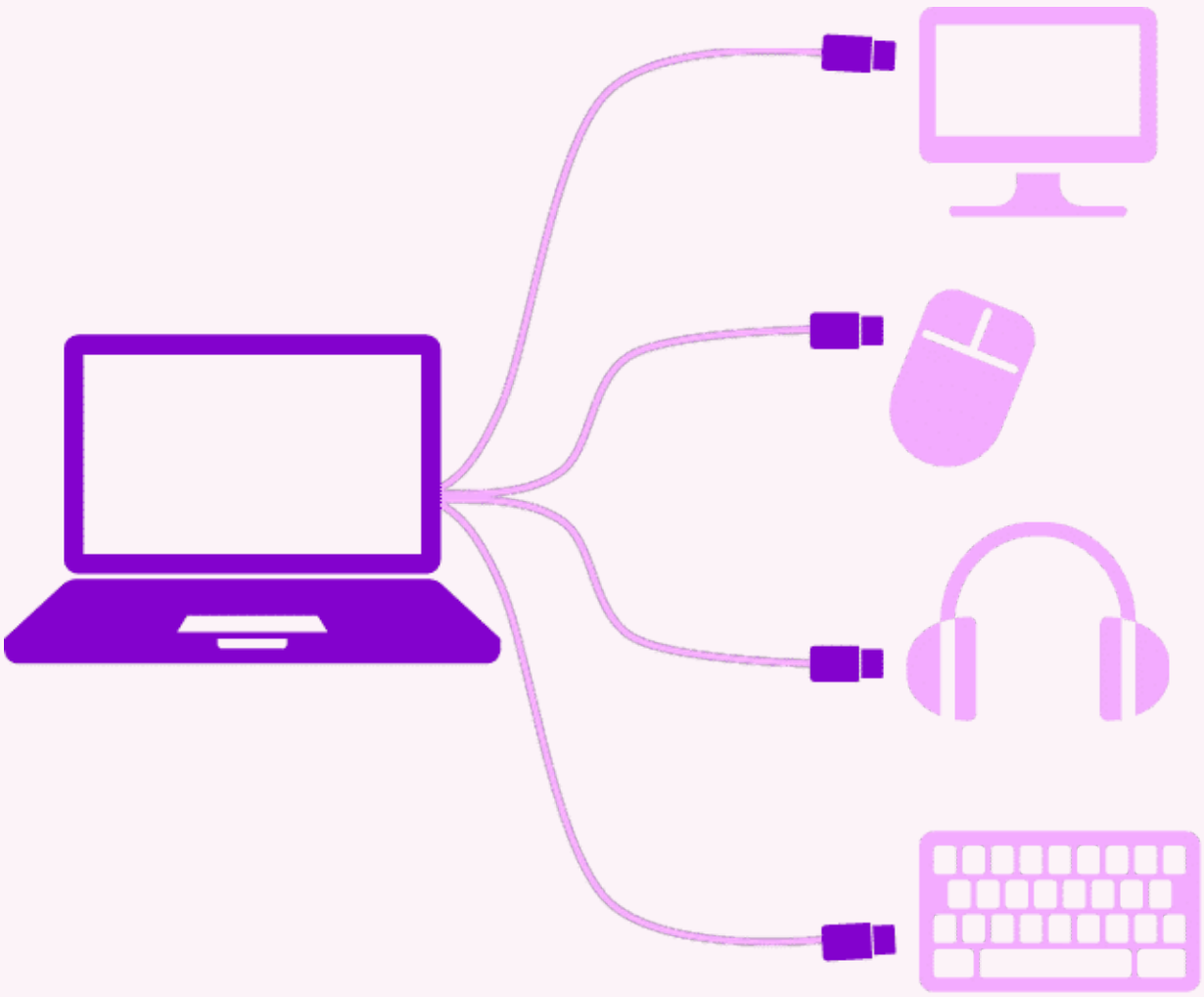


MCP is an open-source standard for AI to connect to and use external tools and data sources.

📌 Definition

An open source standard is a set of publicly available rules that anyone can use and implement.

MCP makes these interaction easy and consistent by using the same approach for every tool, so you don't need to manually set up each individual interaction.

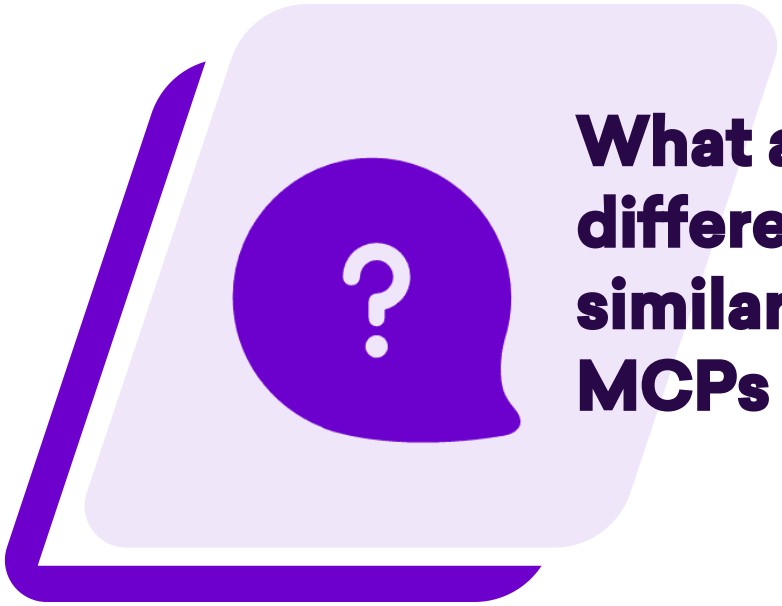


If you have read about MCP, you have probably found this analogy: MCP is often called the **USB-C for AI applications**.

Just as USB-C gives computers a standard way to connect different devices, MCP provides a standard way to connect AI models with external tools and

systems.

Basically, it does the same job as APIs, but in a more standardized way.



What are the differences and similarities between MCPs and APIs?

Similar

- **Client/server architecture.** They both use a client/server setup, in which the client sends a request, and the server replies with the answer.
- **Layer of abstraction.** They both hide the technical details of how each system works. The client doesn't need to know how the server does its

Different

- **Purpose.** MCPs are made to connect AI applications with external tools, while APIs are more general and not specifically designed for AI.
- **Dynamic self-discovery.** MCPs let LLMs and AI agents find available tools and what they can do automatically, which is something

work, it only needs to know how to send requests and what kind of response to expect.

- **Simplify integration.** They both **make it easier to connect** different systems, so developers can integrate tools together without having to build

APIs don't provide.

- **Standardized interface.** Every MCP server follows the same rules, so you can set it up once and use it with many tools. APIs are all different,

g AP
API n
nts c

Let's look at the architecture of MCP and how it works in more detail.

Continue to 1.4: MCP architecture

1.4 MCP architecture

MCP uses a client/server architecture.

📌 Definition

Architecture is the overall structure and design of how a system's different parts are organized and work together.

Continue to 1.4.1: MCP components

1.4.1 MCP components

MCP has three main components:

1. Host
2. Client
3. Server

1: MCP host

1

MCP host

The host is the AI application that users interact with directly. It also coordinates and manages MCP clients.

Some examples of hosts are:

- **AI chat applications**, like ChatGPT, Gemini, Claude, etc.
- **Orchestration tools for automation**, like Make.

The main responsibilities of an MCP host are:

User Interaction

The MCP host is the main AI application or interface that the user communicates with. It serves as the entry point for receiving user requests and delivering responses.

Permission management

The MCP host controls what actions are allowed, ensuring that only authorized users or tools can perform certain operations.

MCP client management

The MCP host makes sure each MCP client is properly connected to the MCP servers it needs. It keeps track of which clients are active and ensures they can communicate properly with MCP servers.

MCP server connection

The MCP host starts and keeps the connections with servers, so the AI can use the tools and the services it needs.

Request flow coordination

The MCP host directs each user request to the right MCP client or tool. It ensures that the process moves in the correct order from input to AI processing to tool execution and finally to the response.

🧠 **Context management**

The MCP host keeps track of what the user asks and which data the MCP shares with tools. This helps the AI call the right tools with the correct parameters and respond appropriately without losing track of the conversation.

📚 **Results presentation**

The MCP host collects the outputs from MCP clients or tools and displays them to the user in a clear and understandable way.

2

MCP client

The client is a part of the MCP host that connects to MCP servers, sends requests for tools or data, and receives responses to pass back to the host.

In Make, there is a specific **MCP Client app** that allows you to connect to a MCP server, list the tools provided, select one, and call it when the scenario runs. You will learn more about it in the next unit.

The main responsibilities of an MCP client are:

⚡ **MCP server connection**
The MCP client establishes and maintains communication with an MCP server. Each MCP client connects to only one MCP server at a time.

💬 **Communication management**
The MCP client sends requests and receives responses, follows MCP rules, and handles interruptions or timeouts.

📮 **Requests and responses processing**
The MCP client takes the host's requests, converts them into a format the MCP server can understand, and sends them. When the server replies, the client checks the response, handles errors, and ensures the information is correct and meaningful before passing it back to the host.

🔍 **Dynamic discovery**
The MCP client queries the server to find out what tools, resources, and prompts are available. You will learn more about this later in the unit.

3: MCP server

3

MCP server

The server is a program that gives AI apps access to specific tools or services of third-party applications through a standardized way of communicating.

Some examples of servers are:

- **PostgreSQL** to query and manage database records.
- **Slack** to read messages, send notifications, and manage channels.
- **Google Drive** to access, read, and manage files in cloud storage.
- **Make** to trigger automation workflows and manage scenarios.

Make can work like an MCP server by exposing scenarios and the tools needed to manage them, allowing external AI applications to use these resources. You'll learn more about this in the final unit of the course.

The main responsibilities of an MCP server are:

🧰 **Tools and services providing**

The MCP server makes specific functions or actions available to AI applications in a standardized way.

📥 **Requests processing**

The MCP server receives requests from MCP clients and converts them from the MCP protocol into the format the third-party application understands. It then executes the requested action and converts the results back into MCP format before sending them to the client.

🔑 **Authentication and permissions handling**

The MCP server ensures that only authorized clients can access tools and data.

🔍 **Dynamic discovery**

The MCP server provides clients with an up-to-date list of available tools, resources, and prompts.

🛠️ **Tool definitions maintenance**

The MCP server keeps the descriptions, inputs, and outputs of available tools accurate and up-

to-date.

MCP servers provide access to **three main types of capabilities**, often called **primitives**. These define what the server can do and what the AI can request.

📌 Definition

A **capability** is a specific action or resource that an MCP server makes available for AI to use.

Capabilities

⚒️ Tools

Actions or functions that the AI can trigger to perform specific tasks.

Each tool includes details like its name, description, input, and output.

Example: a tool that creates a new task in a project management app.

🗂️ Resources

Read-only data or content that the server provides, like documents, files, or database entries.

They give the AI access to reliable information. The AI can use this information to make better decisions,

✍️ Prompt templates

Pre-made text examples that guide an AI on how to respond or perform a task.

They help the AI to correctly format requests and commands.

Example: a prompt template for creating a new task in Asana: guides the AI to fill

without needing to run a tool or perform an action.

in task name, description, assignee, and

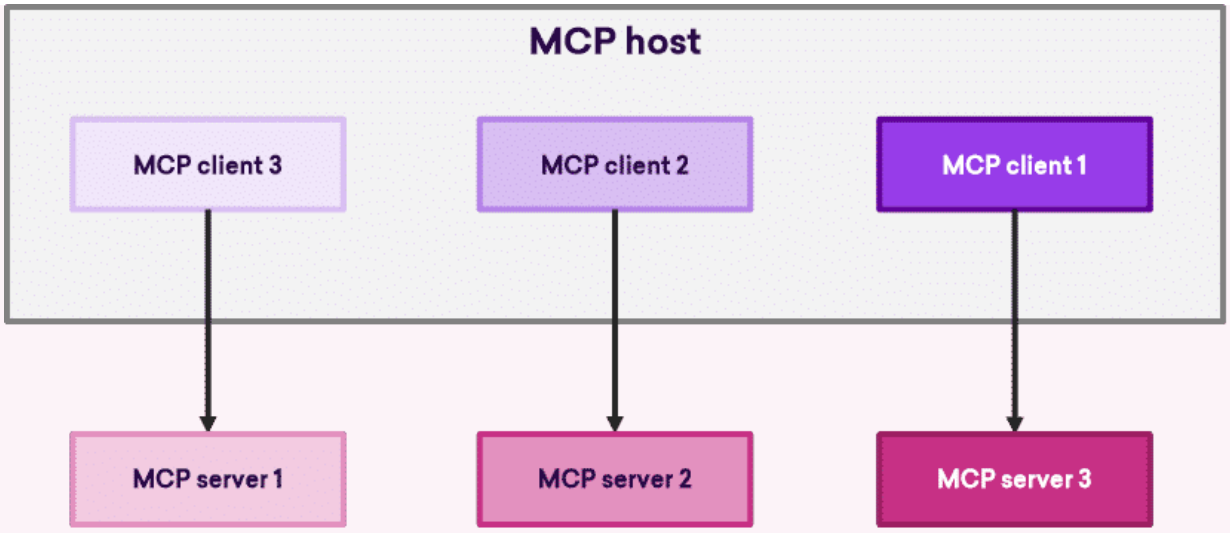
Not every MCP server offers all three types of primitives, many focus mainly on **tools**.

Continue to 1.4.2: MCP architecture

1.4.2 MCP architecture

This is what the architecture of an MCP system looks like.

The **host can connect to multiple MCP servers through clients**, with each client maintaining a 1:1 connection with its respective server.



This architecture is modular and flexible.

MCP clients let the host connect to multiple servers at the same time, and adding another client makes it easy to connect to even more servers if you need access to additional tools or data sources. This setup simplifies combining different tools and allows the system to **adapt as needs change**, so AI applications can grow and use new capabilities smoothly.

Continue to 1.5: MCP protocol

1.5 MCP protocol

The MCP protocol is the set of rules that defines how all the components (hosts, clients, and servers) communicate with each other.

It **standardizes requests, responses, and messages** so AI applications can interact with MCP servers in a consistent way. Essentially, it makes the whole system speak the same language so AI can access tools, data, and services smoothly.

Let's have a look at how it works.

Continue to 1.5.1: Message protocol

1.5.1 Message protocol

The MCP protocol uses JSON-RPC 2.0 to send and receive messages.

📌 Definition

A **message protocol** is a set of rules that defines how systems format, send, and understand messages.

RPC stands for **Remote Procedure Call**. The MCP client uses it to ask the server to perform a task, then send the result back. All messages between the client and server are formatted in **JSON**.

RPC lets programs talk to each other and call each other's functions. In the case of an MCP server, these functions are the tools, resources, and prompts that the server provides for the AI to use.

There are three types of messages.

Click each card to learn more.

Requests

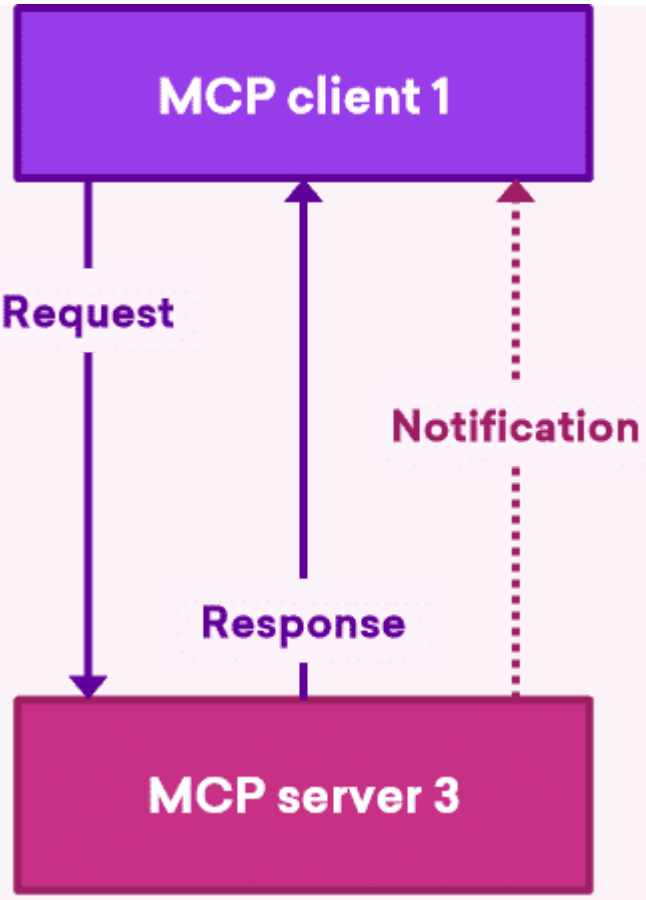
Ask to do something.

Responses

Reply with a result or error.

Notifications

Receives information from the MCP server.



The **MCP client sends a request to the MCP server** to perform a specific action using a tool. Each request includes a unique identifier (id), the method name (for example, tools/call), and any parameters needed for the action.

The server executes the action and **sends back a response with the result**, or an error if something went wrong. Responses match the request id and contain the result or an error.

Notifications are one-way messages that do not expect a reply. The server often uses them to **provide updates or inform the client about events**.

Real-life example:

The MCP client sends a **request to the Asana MCP server to create a new task** in a project. The request includes the task name, assignee, and due date.

The MCP server **performs the action using its tool and sends back a response** containing the task details, confirming it was created successfully. If something goes wrong, the server returns an error instead.

Meanwhile, the server can also send notifications to the client, like when another system or user updates or completes a task.

Continue to 1.5.2: Transport methods

1.5.2 Transport methods

JSON-RPC 2.0 doesn't depend on a specific transport method, so it can work over different ways of sending messages.

📌 Definition

A **transport method** is the way systems send and receive messages.

MCP supports multiple transport methods, including **HTTP streaming** and **Server-Sent Events (SSE)**.

Click each one to learn more.

HTTP streaming

HTTP streaming is a way for a server to send data to a client bit by bit over a single connection. The connection stays open, and the server sends chunks of information as they become available.

This is useful when you need a continuous flow of live updates, such as logs, real-time data feeds, or video.

Server-Sent Events (SSE)

SSE is a simple way for a server to push real-time updates to a client over HTTP, in one direction only. The client opens a connection, and the server sends new messages whenever updates are available.

SSE works well for things like notifications, live scores, or status changes.

Note that SSE is now considered a legacy method, but Make currently supports both transport options.

Continue to 1.5.3: Authorization

1.5.3 Authorization

When an MCP server sends or receives information, it doesn't automatically know who is making the request.

That's fine for simple situations, for example, when the server is using **public data** that anyone can access.

When the MCP needs to reach remote or restricted services, it must prove its identity through authorization, ensuring that only approved clients can access sensitive data or tools.

There are two common authorization methods.

Click each one to learn more.

API keys

An API key is tied to a specific account or project and the MCP server includes it in each request to show that the request is allowed.

API keys are quick and easy to use, but they can be risky because anyone who gets the key can use it without restrictions.

OAuth 2.0

OAuth 2.0 is a more secure, user-based way to give access. The user has to allow the MCP to access certain data or perform actions. If they agree, the service gives the MCP a temporary access token.

The MCP server then uses this token to send authorized requests. OAuth 2.0 works best when user consent and different access levels are important.

With these methods, the MCP server can safely connect to both open and protected resources while keeping data secure and access properly managed.

Note that Make currently supports both API key and OAuth 2.0 authentication methods.

Continue to 1.5.4: Dynamic discovery

1.5.4 Dynamic discovery

Dynamic discovery allows MCP clients to automatically and in real time understand what an MCP server can do, including the tools, resources, and capabilities it offers, without needing manual setup.

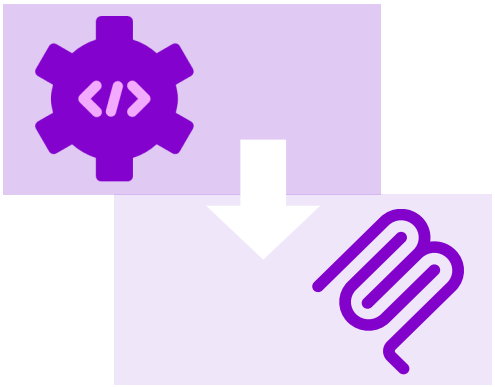
Hey MCP server, what can you do?

Hello MCP client, here is my list of tools, resources and prompts. Have fun!

Instead of relying on hardcoded lists or fixed endpoints, like for APIs, the **MCP client can request a list of available capabilities from the server.** The server then provides details of all current tools, resources, and

prompts, making the client flexible and adaptive to whatever the server offers.

This is one of the differences between APIs and MCPs.



APIs

APIs have a **fixed list of endpoints that define what actions an application can perform**. If the API changes, such as adding new endpoints or updating existing ones, developers must **manually update** the client to keep using it correctly. This can require developer intervention and maintenance every time the API evolves.

MCPs

MCPs support dynamic discovery, allowing the client to **ask the server in real time what tools, resources, and prompts are available** without any pre-programmed knowledge from the server. When the server adds, removes, or updates capabilities, the client can **automatically detect these changes** without needing manual updates, making the system more flexible and adaptive.

AI agents can automatically discover the tools and functions an MCP server offers and understand how to use them. Then they can choose the right actions to solve tasks or respond to requests in real time.

The main benefits of dynamic discovery are:

- **No static lists:** clients don't need to keep fixed catalogs of tools.
- **Easier updates:** you can add, remove, or change tools without having to redo the integration.

- **Smarter agents:** AI clients can decide which tools to use and when, based on the list of tools that are available.

Dynamic discovery lets AI systems **automatically see what tools a server offers and choose the right ones for each task**. This makes AI agents more flexible and able to act intelligently in real time, without relying on fixed lists or manual setup.

Continue to 1.6: MCP lifecycle

1.6 MCP lifecycle

The MCP lifecycle consists of three phases: initialization, operation, and shutdown.

📌 Definition

The **MCP lifecycle** is the complete sequence of stages from when a client connects to a server, through using its tools, until the connection ends.

Continue to 1.6.1: Initialization

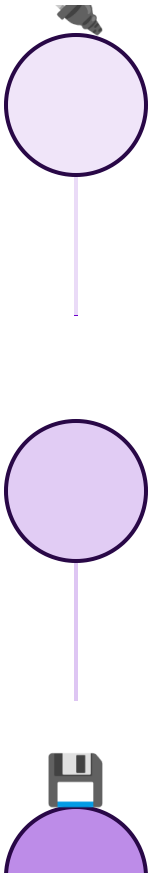
1.6.1 Initialization

The initialization phase is the first stage of the MCP lifecycle.

During this phase the server tells the client to perform what is known as a **handshake**: the server tells the client **what tools and resources it offers**, and the client confirms it's ready to work together. This all happens automatically in the background when you start your application, before any actual work begins.

This is how it works:

Connection



The MCP host starts up and its MCP client connects to the configured MCP servers

Discovery

The MCP client asks the server which capabilities they offer and the server shares the tools, resources, and prompts available

Registration

The client registers these capabilities so the AI can access and use them when needed

Continue to 1.6.2: Operation

1.6.2 Operation

The operation phase is when the real work happens.

After the host sets up the connection, the client and server start exchanging requests and responses.

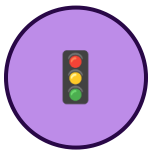
This is how it works:

User prompt

The user sends a message to the AI or defines the AI agent's user prompt.

Tool selection

The AI analyzes the user's request and determines which MCP tools (if any) it needs to answer properly.



Permission request

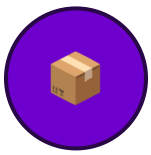
The MCP host asks the user for permission to use a specific tool.

Request sending

The MCP client formats the request in JSON-RPC 2.0 and sends it to the server.

Processing

The server handles the request and performs the requested action.



Result return

The server sends the response back to the client in JSON-RPC 2.0.

Context integration

The AI receives the results and combines them with its knowledge to create a complete answer for the user.

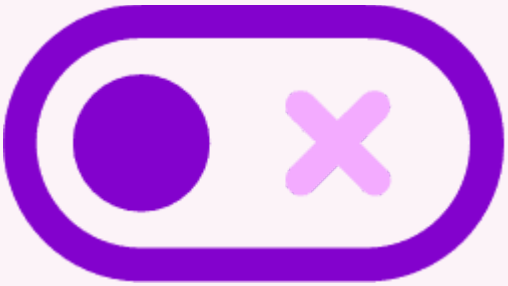
Response generation

The AI generates the answer for the user.

Continue to 1.6.3: Shutdown

1.6.3 Shutdown

The shutdown phase is the final step of the MCP lifecycle.



In this stage, the client and server **close their connection, stop exchanging information, and remove any resources they were using**. This ensures everything

ends safely and the system is ready for the next session.

Continue to complete this unit

1.7 Wrap up

1

MCP is an open-source standard that lets AI apps, agents, and automation tools **connect to external servers and call their tools in a unified, consistent way**. This simplifies the M×N integration problem: each AI system can automatically understand what each tool does and how to use it. This makes integrations faster, simpler, and easier to manage. MCP also lets AI automatically find and use the available tools.

2

MCP uses a client/server architecture with three main parts: the **host**, the **client**, and the **server**. The host is the **AI application that users**

interact with and manages requests, permissions, and context. The client **connects the host to servers**, writes the user prompt or sends requests, and receives responses. Servers **provide access to tools**, data, and prompt templates, handling requests and permissions in a standardized way. This setup is flexible, letting a host connect to multiple servers at once and making it easy to add new capabilities as needed.

3

The MCP lifecycle has three phases: **initialization**, **operation**, and **shutdown**. During initialization, the client and server perform a handshake, **exchanging information about available tools so they're ready to work together**. In the operation phase, the **AI receives user requests or a defined goal, chooses the right tools, asks for permission if needed, sends requests to servers, and integrates the results from the server to generate responses**. Finally, in the shutdown phase, the client and server **close the connection** and clean up, ensuring the system is ready for the next session.

Well done! You now know the basics of how MCP works.

In the next unit, you will learn how Make works as an MCP client.

Mark this task complete to continue to the next unit.