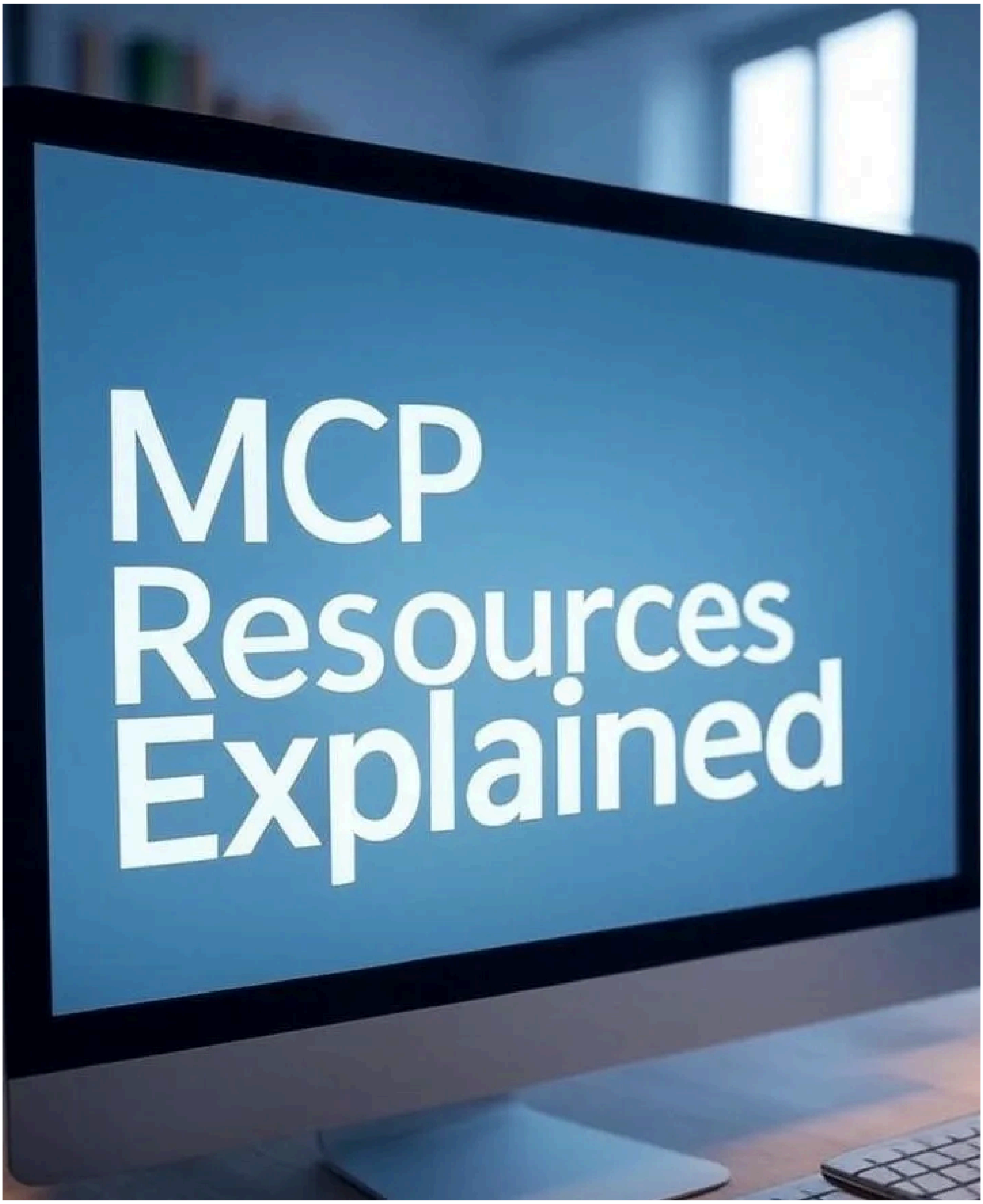
MCP Resources explained (and how they differ from MCP Tools)

6 min read · Aug 1, 2025

Laurent Kubaski Following

This article is part of my MCP Explained series and a follow up to my MCP Prompts Explained article.



1. MCP Resources vs MCP Tools

> *“**MCP Resources** allow MCP servers to share data that provides context to language models, such as files, database schemas, or application-specific information”*
>
> Source: MCP Specification

Now compare this with the official definition for MCP Tools:

> *"**MCP Tools** enable models to interact with external systems, such as querying databases, calling APIs, or performing computations"*
>
> Source: MCP Specification

Yes you're right: there is not much difference between sharing the content of a file (first definition) and querying a database (second definition), which is why the #1 question people ask when they discover MCP Resources is "*what the hell is the difference between a MCP Resource and a MCP Tool?*".

I've tried asking that question to Claude and I got this answer:

> *The key difference is intent and usage pattern:*
> *- Resources are for when you want to give the model access to existing data to read and reference.*
> *- Tools are for when you want the model to perform actions or computations.*
>
> Source: Claude Sonnet 4

But as I wrote above **that explanation doesn't make** sense since a Tool can also be used to "*give the model access to existing data*". To do that, you just need to create a tool that… returns something.

For example, here is a FastMCP 2.0 server with a Resource and a Tool that do exactly the same thing:

```python
from fastmcp import FastMCP

server = FastMCP("Greetings")

@server.resource("resource://greeting")
def get_greeting_resource() -> str:
    return "Hello from FastMCP Resource"

@server.tool
def get_greeting_tool() -> str:
    return "Hello from FastMCP Tool"
```

The real difference between the two is their user interaction model:

- **Tools** are designed to be **model-controlled**, meaning that the language model can discover and invoke tools automatically based on its contextual understanding and the user's prompts (source: MCP Specification)
- **Resources** are designed to be **application-driven**, with host applications determining how to incorporate context based on their needs (source: MCP Specification)

Put differently: the model is never going to automatically retrieve a Resource, it's up to the client application to decide when to retrieve a Resource and when to send it to the model.

2. Application-Driven?

But according to me, stating that "*Resources are designed to be application-driven*" is confusing. This seems to imply that a client application can decide on its own which Resources to use and when… but how can it know that?

For example, let's say that you have this Resource:

```
@server.resource("resource://greeting")
def get_greeting_resource() -> str:
    return "Hello from FastMCP Resource"
```

If this was a **Tool** instead, then the model would know when to call it: in that case, that would be when a user asks "greet me".

Here however it's a **Resource** so it's supposed to be called by the client application: but when?

And that's where the MCP specification gets confusing with the examples it provides:

> *For example, applications could:*
> *- Expose Resources through UI elements for explicit selection, in a tree or list view*
> *- Allow the user to search through and filter available Resources*
> *- Implement automatic context inclusion, based on heuristics or the AI model's selection*

In the first two examples, The MCP Resource is **selected by the user** and in the last example, it is **selected by the application with the help from the model.** So said different, even though in the end it's indeed the *application* that decides what to do with the Resource, most of the time it's really *the user* who decides which Resource to use.

So with that said, my definition of an MCP Resource becomes the following:

> *"MCP Resources represent data or files that an MCP client can read. They are typically selected by the user, and applications then determine how to incorporate them in the context used by the model"*
>
> *Source: Laurent Kubaski*

3. When to use a Tool and when to use a Resource?

Now you know:

- If the model needs to discover and call it, it's a **Tool**.
- If the user and/or application decides when it's relevant to call it so that it can be used as context to the model, it's a **Resource**.

4. How MCP Resources are used by various MCP Clients

In this chapter I'll use this simple MCP Server created using FastMCP 2.0 that defines a single Resource:

```python
from fastmcp import FastMCP

server = FastMCP("Greetings")

@server.resource("resource://greeting")
def get_greeting() -> str:
    return "Hello from FastMCP Resources"

if __name__ == "__main__":
    server.run(transport="stdio")
```

The official MCP site maintains a list of example MCP Clients and mentions which part of the MCP Specification those clients support: as I'm writing this, there are 58 MCP Clients listed and only 26 of them support Resources.

Let’s now see how to use MCP Resources in some of those clients.

4.1. Using MCP Resources in Claude Desktop

- I’ll assume that Claude Desktop is already connected to the MCP Server above (see Connecting your Python MCP Server to Claude Desktop)
- In Claude Desktop, Click the “+” button just below the place where you type your prompts.
- Select “Add from [name of your MCP server]”:

Claude Desktop: using the Resource (step 1)

- Select your Resource:

Claude Desktop: using the Resource (step 2)

- And voilà, the Resource is now added as an attachment and you can ask Claude Desktop to interact with it. For example you can type “summarize this” and hit enter.

Claude Desktop: using the Resource (step 3)

So if we come back to my earlier definition of an MCP Resource where I wrote "*They are typically selected by the user, and applications then determine how to incorporate them in the context used by the model*", you can see that in this case, the Claude Desktop application adds the resource to the context as an attachment.

Just for fun I looked under the cover to see what actual message the Claude Desktop application sends behind the hood and it looks like this:

4.2. Using MCP Resources in VS Code GitHub Copilot

- I’ll assume that Visual Studio Code is already connected to the MCP Server above (see Connecting your Python MCP Server to Copilot Chat in Visual Studio Code)
- In the Copilot Chat field, click the “Add Context” button:

VS Code: using the Resource (part 1)

- Click “MCP Resources”:

VS Code: using the Resource (part 2)

- Choose the MCP Resource:

VS Code: using the Resource (part 3)

- That’s it, the MCP Resource is now ready to be used:

VS Code: using the Resource (part 3)

4.3. Using MCP Resources in Claude Code

Since I don’t have a Claude Code subscription (which starts at $17/month), I’ll shamelessly copy-paste the relevant part from their documentation on using MCP Resources :

Claude Desktop: using a Resource

5. Further Readings

- Introduction to Model Context Protocol — Resources
- The Complete MCP Course — Resources
- MCP Resources vs Tools

Mcp Server

Artificial Intelligence

Generative Ai Tools

Following

Written by Laurent Kubaski

114 followers · 3 following

I'm a product manager at Salesforce, more info here: https://kubaski.com/

Responses (3)

Bgerby

What are your thoughts?

Lucas Gomes
Aug 10

Great article, it cleared up my doubts about when and how to use resources

1 Reply

Brayan Neciosup
Oct 22

Gracias por la información !!. Estaba sufriendo en el porque mi cliente MCP no podia encontrar los recursos, trate de exponerlos como herramientas y todo, pero, no estaba contextualizandolos correctamente en el prompt al cliente MCP. Miles de gracias!!!

Reply

Viraj Vaitha
Oct 2

This is the first read I've found that explains it clearly with a visual example lol!

It was nice just to see the dropdown of the resources you can select.

Thanks!

Reply

More from Laurent Kubaski

Laurent Kubaski

Ollama prompt templates

This article is part my “Ollama Explained” series.

Feb 9 17 1

Laurent Kubaski

Connecting your Python MCP Server to Copilot Chat in Visual Studio Code

This article is part of my MCP Explained series.

Jun 20 5

Laurent Kubaski

Ollama endpoints options parameter

This article is part my “Ollama Explained” series.

Mar 16 37 1

Laurent Kubaski

Finding Salesforce Duplicate Permission Sets using the Jaccard index

Or “why Generative AI should not be seen as the default solution to every problem you find”

5d ago 1

See all from Laurent Kubaski

Recommended from Medium

Laurent Kubaski

MCP Explained (article series)

Don’t worry, the Internet is already swamped with generic articles on that topic so I’m not planning to write another one: this article is…

Jun 20 6

Anoop Ninan George

Building a Remote MCP Server and Client with FastMCP, LangChain, and LangGraph

In this tutorial, we'll explore how to set up a remote Model Context Protocol (MCP) server using FastMCP and build a client using LangChain…

May 29 34

Anil Goyal

How to Test your MCP Server using MCP Inspector

The above is my blog on creating a MCP server using python using FastMCP.

Jul 16 5 1

Heeki Park

Building an MCP server as an API developer

Anthropic released MCP at the end of November 2024. It took a few months for it to catch on, but my, oh my, it feels like the community is…

May 14 576 5

Guangya Liu

MCP Sampling: Architecture, Workflow, and Practical Guide

Not a Medium member? Visit https://gyliu513.github.io/

Aug 21 2

Johnny O

How to Set Up Atlassian's MCP Server Locally: A Step-by-Step Guide

I recently tested out Atlassian's MCP server and really enjoyed the experience.

Jul 16 1

See more recommendations