Chapter 3: What is Model Context Protocol?

To MCP Or Not To MCP, That Is The Question

- Sundar Pichai

Now that we've warmed you up with the essentials, we're diving into the core of this book—the Model Context Protocol (MCP). This is what the book was built around: a topic that's gaining traction now and is likely to remain central throughout the year, or maybe decades to come.
Before exploring MCP, it's important to understand what a protocol is.

3.1 What is a Protocol?

Think of protocols as the digital equivalent of social etiquette—like saying "hello" before a conversation or shaking hands when you meet someone. Computers also need a common set of rules to communicate properly. These rules are called **protocols**.

A. Everyday Examples of Protocols

1. **HTTP/HTTPS**: This is how your browser talks to websites. HTTP loads the page, while HTTPS adds encryption for safety—perfect for online banking or shopping.

2. **TCP/IP**: The postal service of the internet. It breaks up your data, sends it across the world, and reassembles it correctly at the other end.

3. **SMTP**: The protocol behind sending emails. When you hit "Send" in Gmail, SMTP makes sure your message gets to the right

inbox.

4. **Wi-Fi (IEEE 802.11)**: The rules your phone and router follow to connect wirelessly.

Each of these keeps our online experiences running smoothly—whether we're browsing memes or sending mission-critical emails.

B. Why Protocols Matter

Without protocols, digital communication would be like two people yelling in different languages across a noisy room. But with protocols, we get:

1. **Consistency**: Devices know what to expect.
2. **Reliability**: Messages get through in one piece.
3. **Security**: Encryption keeps hackers at bay.
4. **Efficiency**: No guessing games—data moves fast and clean.

Think of it like driving: everyone follows the same traffic rules, so there are fewer accidents and less chaos.

C. What If Protocols Didn’t Exist?

Now let’s play doomsday for a sec. What happens if we pull out protocols out of the equation?

1. **No HTTP?** Websites wouldn’t load. Typing "google.com" would do absolutely nothing.
2. **No SMTP?** Emails vanish into the void—or worse, show up as unreadable nonsense.

3. **No TCP/IP?** Say goodbye to stable video calls, smooth downloads, or messages that make sense.

4. **No Wi-Fi/Bluetooth?** Your phone and router would be total strangers.

5. **No HTTPS/SSL?** Every password or credit card number you send would be out in the open like writing your bank PIN on a billboard.

It would be like driving on roads with no signs, no lanes, no traffic lights. Total digital anarchy.

You might have understood how important protocols are, but unfortunately, the world of AI agents was operating without protocols before Model Context Protocol came in. Hence, you might have understood why there was such a ruckus all over the place.

MCP is the protocol for AI agents to connect with external tools and APIs.

3.2 What is Model Context Protocol?

Imagine trying to plug a USB device into a port that doesn't fit—frustrating, right? Before MCP, integrating AI models with various tools and databases felt just like that. Each connection required custom code, making the process cumbersome and error-prone. MCP, introduced by Anthropic in late 2024, serves as the **"USB-C" for AI integrations—**a universal, standardized protocol that simplifies and streamlines these connections**.**

3.2.1 The Need for MCP

Before MCP came onto the scene, integrating AI agents with external systems was a manual and often cumbersome process.

Developers had to:

1. **Build Custom Connectors:** Each external tool or API required its own bespoke integration. This meant writing specific code to handle the communication between the AI agent and the external service.

2. **Handle Authentication Individually**: Every service had its own authentication mechanism, necessitating separate handling for API keys, OAuth tokens, and other security protocols.

3. **Manage Data Formats and Protocols**: Different services used various data formats (like JSON, XML) and communication protocols (like REST, SOAP), requiring additional code to parse and format data appropriately.

This approach was not only time-consuming but also led to a maintenance nightmare, especially as the number of integrations grew.

And that's why model context protocol is gaining traction all over the internet.

This is also the reason why we didn't discuss frameworks like Langraph, Crew-AI, etc., because in that case, you need to customize every AI tool depending on the framework, and there's no standardisation. With MCP coming in now, the interaction between AI agents and tools has become standardised, and you don't need to learn multiple frameworks or standards to integrate third-party extensions to LLMs.

3.2.2 How MCP Works

At its heart, MCP operates on a **client-server model**, which might sound techy—but it’s actually pretty intuitive. Think of it like a waiter (client) and a kitchen (server). The waiter takes your order and brings it to the kitchen, where the magic happens. Then they bring your meal (response) back to you. MCP works the same way for AI systems.

A. MCP Clients

These are the AI applications or agents—think ChatGPT, a code assistant, or a customer support bot.

1. **Their Role**: Clients are the ones *requesting* data or actions.

2. **What They Do**: They ask servers for things like, "Hey, can I get all unresolved GitHub issues?" or "Can you run this SQL query?"

3. **Why It Matters**: The client acts as the bridge between the AI model and the real world—it knows what it needs, but it depends on the server to get it.

B. MCP Servers

Now, here's where the tools and data live. MCP servers are like plug-and-play modules that expose:

1. **Tools**: These are functions the AI can invoke (e.g., `send_email`, `list_calendar_events`, or `fetch_customer_data`).

2. **Resources**: Static or dynamic data like CSV files, emails, vector databases, or even codebases.

3. **Prompts**: Templates that help guide how the AI should interpret or present the information.

In short, MCP servers *serve* the stuff the AI needs to get work done.

C. Standardised Communication

Here's where the real magic of MCP kicks in.

1. **Uniform Language**: The client and server don't just shout at each other in random formats. They use a standardized protocol —typically something like JSON-RPC. This ensures both sides "speak the same language."

2. **No Guesswork**: This standardization means tools and data can be discovered, used, and invoked by the AI dynamically without developers writing messy, one-off integrations.

3. **Plug-and-Play Vibes**: You can switch out one MCP-compatible server for another (e.g., swap GitHub for GitLab) and the client won't even flinch.

What is JSON-RPC?

JSON-RPC stands for **JavaScript Object Notation - Remote Procedure Call**. It's a lightweight remote procedure call (RPC) protocol encoded in JSON. Translation? It's a way for one program to call functions on another system (usually over a network) using plain ol' JSON to pass data back and forth.

D. Putting It All Together — A Real-World Example

Let's say you're using an AI assistant to help with project management.

1. **Connection**: The AI app (client) connects to an MCP server set up for GitHub.

2. **Tool Use**: The model uses the `list_issues` tool to fetch open issues in a repo.

3. **Data Return**: The server grabs the issue data and returns it as a resource.

4. **Prompt Time**: The AI uses a predefined prompt to analyze and prioritize the issues.

5. **Follow-Up Action**: It invokes another tool (like `create_task`) to push top issues into work management platforms like Asana.

You didn't write a single line of integration code. The client and server simply followed the MCP playbook.

E. Why This Architecture Rocks

1. **Modularity**: Build Once, Use Everywhere.
   The architecture allows you to develop tools once and deploy them across multiple applications and environments. Whether you're adding new services or switching clients, the modular design ensures minimal disruption and maximum reusability.

2. **Scalability**: Grow Without Growing Pains
   Multiple teams can independently manage their own MCP servers—for example, separate ones for data, DevOps, or CRM—without stepping on each other's toes. Meanwhile, any AI application can seamlessly connect to and use these services as needed.
3. **Security & Control**: Keep It Tight and Traceable
   Each server maintains full control over which tools and data it exposes to the outside world. This ensures strong security boundaries and makes all interactions auditable, giving organizations better governance over AI-driven workflows.

4. **Future-Proof**: Designed for What's Next
   As the AI landscape evolves and new tools emerge, this architecture ensures you won't need to rebuild integrations from scratch. You can plug in innovations without overhauling your existing systems, saving time and resources.

Now you see how important MCP is, and that's why this book has been written. The moment we read about Model Context Protocol, we were just blown away.

3.3 Key Components of MCP

In this subsection, we will be discussing the key components of the Model Context Protocol Server.

3.3.1 Resources: What the AI Knows

Resources are the data the AI can “read” and use for reasoning. Think of these as the AI’s memory or context—it could be a CSV file of sales numbers, a batch of emails, or the result of a database query. Whether it’s a static document like a PDF or a dynamic stream like a live API response, if it provides information, it’s a resource.

For example, imagine you’re building an AI assistant for recruiters. The resumes your team uploads? Those become resources. Now the model can comb through them to answer questions like, “Who has experience with Kubernetes and Python?” without you lifting a finger.

This shift—from manual copy-pasting into the prompt to structured, queryable context—is one of the game-changers MCP brings to the table.

3.3.2 Tools: What the AI Can Do

Resources are external knowledge bases that give the AI something to refer—but what if you want it to act? That’s where **Tools** come in. These are the model’s arms and legs, allowing it to trigger real-world functions like creating a task, sending an email, querying a database, or even executing shell commands.

So instead of just saying, “This bug looks critical,” your AI assistant could take it a step further and say, “I’ve filed the bug report in GitHub for you.” That’s the magic of tools.

Each tool is defined by developers and made discoverable through an MCP server. Once that tool is exposed, any compatible AI agent can find it, understand how to use it, and call it dynamically—no custom code required. You can think of tools as the AI’s built-in “API menu”—ready to be used whenever it makes sense.

3.3.3 Prompts: What the AI Should Say

Now, just because the AI has access to data (resources) and actions (tools), that doesn’t mean it’ll know what to do with them. That’s where **Prompts** come in.

Prompts are like the AI’s script. They tell the model how to behave, what task to perform, and how to structure the output. MCP takes prompts beyond one-off manual instructions by turning them into reusable templates that can be filled in dynamically.

For example, instead of crafting a new prompt every time, you could have one that says:
“Summarize the following customer support tickets and tag each as a bug, feature request, or question.”
Then, at runtime, the model gets the actual tickets (as a resource), plugs them into the prompt, and runs the analysis.

Prompts bring consistency, modularity, and intelligence to AI workflows—especially when paired with tools. You could summarize GitHub issues, then use another tool to automatically file tasks based on the model’s suggestions—all in one smooth flow.

3.3.4 Real-World Use Cases

Let’s say you’re using an AI-powered productivity agent at work. You ask it, *“Can you clean up my project backlog and assign tasks to the* *right people?"*

Behind the scenes, here's what happens:

1. It pulls all your unassigned GitHub issues (resources).
2. It uses a prompt to prioritize and categorize them based on urgency and owner.
3. Then it runs a few tools to create tickets in Asana, notify your team on Slack, and close out stale issues.

All of that happens because the AI has the context (resources), the intelligence (prompts), and the capability (tools) to get it done.

3.4 MCP = AI Agents 2.0?

Before MCP came along, building AI agents felt like assembling IKEA furniture blindfolded—***technically possible, but frustrating*** and definitely not for the faint of heart. You had to hardwire tools into your model pipeline, design complex JSON schemas, maintain flaky integrations, and handle a mess of edge cases. Every new API? A new round of glue-code nightmares.

Even worse, it was mostly ***developer-only territory***. If you weren't comfortable with Python, Docker, and Postman, good luck trying to wire your AI agent to pull data from your CRM or run database queries.

So yeah—tool calling was cool when it first dropped. But let's be real: it wasn't scalable, it wasn't user-friendly, and it definitely wasn't democratized.

…here comes MCP

Now imagine someone walked in and said,

"Hey, what if we just built a universal plug-and-play system for AI tools and data? One where models could discover resources, invoke APIs, and generate dynamic prompts without you lifting a finger every time?"

That’s **MCP** in a nutshell.

MCP makes AI agents more modular, scalable, and composable. It gives us **AI Agents 2.0**—where:

1. Tools aren’t hardcoded, they’re *discoverable*.

2. Prompts aren’t duplicated, they’re *centralized and templatized*.

3. Context isn’t copy-pasted, it’s *streamed in real-time*.

4. And agents aren’t dev-locked—they’re *user-accessible*.

In other words, the barrier to entry drops from DevOps engineer to curious knowledge worker with a laptop.

Here’s the magic: MCP servers can run **locally**. That means no cloud costs, no creepy data surveillance, and no monthly subscriptions. Want your AI agent to pull up your meeting notes, summarize last week’s emails, and generate a to-do list from a PDF? You can do all of that with MCP—on your laptop, for free.

This flips the entire AI dev ecosystem on its head.

Before:“Let’s spin up an OpenAI function-calling endpoint, wrap it in LangChain, and deploy it to a web dev framework like Vercel.”

After: “Let’s run a local MCP server and just let the agent figure it out.”

See the difference?

1. **From Code to Clicks**: No more coding every tool. Just expose it via MCP, and your agent can use it.

2. **From Static to Dynamic**: Agents can discover tools and prompts at runtime—no rebuild needed.

3. **From Silos to Ecosystems**: Anyone can build, share, or remix MCP-compatible components.

4. **From Experts to Everyone**: You no longer need to be a dev to run advanced AI workflows.

So yeah—MCP is more than just a spec. It's the **platform shift** that makes AI agents mainstream, modular, and accessible. Just like how HTML + HTTP made the internet usable, MCP is the protocol layer that turns AI into a real platform—not just a playground.

Wanna build your first MCP agent or see what a local setup looks like? Let's do it.

3.5 Basic Setup

Before we jump onto installing different MCP servers, a basic setup is mandatory

1. Install Claude Desktop app and sign in : https://claude.ai/download
2. Install Python 3.10 or above : https://www.python.org/downloads/
3. Install Docker : https://www.docker.com/products/docker-desktop/
4. pip install uv (a python package)
5. Node.js : https://nodejs.org/en/download
6. Path to Claude's config.json: ***Claude Desktop App→File (top left)-->Settings→Developer→Edit Config***

Don't feel overwhelmed by names like Docker or UV. You won't require in depth knowledge of these tools

Also, a few things to note as well:

1. This entire book assumes that you are ***using MCP with Claude AI Desktop app*** (free to use for some hits) **in Windows**. MCP Servers can also be used with other AI clients like Cursor AI, Windsurf etc. and even with local LLMs. In that case, just the location for the config.json file (that we would be configuring for MCP servers) will change. Rest of the process remains the same.
2. As the world of Generative AI is evolving fast, it may be the case that ***some MCP servers may go obsolete or the installation guide may change*** while we publish this book (Around June''2025). In case of any errors, kindly drop a mail to the authors of the book, or refer to the git repo link mentioned for every MCP server.
3. If any of the MCP servers doesn’t work, kindly retry with providing full path for commands in any system, be it windows, linux or Mac.

The rest of the book will deal with how to get started with Model Context Protocol, running them locally and using APIs and different useful MCP servers that everyone should know. Get your backpack ready for this crazy adventure. We’ll begin by exploring how MCP servers are accessible to anyone and how they enhance productivity by integrating seamlessly with familiar tools like Gmail, PowerPoint, and more in the next section