The Model Context Protocol (MCP) — A Complete Tutorial

Dr. Nimrita Koul 25 min read · Mar 27, 2025

Anthropic released the Model Context Protocol(MCP) in Nov. 2024.

It is developed by Mahesh Murag at Anthropic. Find the complete official documentation. At present, MCP is fully implemented as Python SDK and TypeScript SDK.

Mahesh Murag delivered a great workshop on “Building Agents with Model Context Protocol” during AI Engineer Summit.

Context is the key

The basic capabilities of a Generative AI model depend on its pretraining details, the training data, and the model architecture. To make these pretrained models perform better and improve their relevance and coherence to your task, you must provide a good context to it.

Here context refers to the information the model uses to generate relevant and coherent responses. Context determines how the model understands and continues a conversation, completes a text, or generates an image.

Context can be provided in different ways, depending on the type of model and task:

1. Text-Based Models (e.g., GPT, DeepSeek, LLaMA) receive their context through:

- Prompt Context: The input text or query that guides the model's response.
- Token Window: The number of tokens the model can "remember" at a time (e.g., GPT-4-Turbo can handle ~128K tokens).
- Conversation History: In chatbots, previous exchanges help maintain context in multi-turn dialogues.
- Retrieval-Augmented Generation (RAG): Context from external documents retrieved dynamically to improve responses.

2. Image and Multimodal Models (e.g., DALL·E, Gemini) receive their context through:

- Text Descriptions: The prompt guides image generation.
- Visual Context: If an image is provided, the model analyzes its content before generating new elements.
- Cross-Modal Context: When combining text and images, models interpret both to generate meaningful outputs.

3. Code Generation Models (e.g., Codex, DeepSeek-Coder) receive their context through:

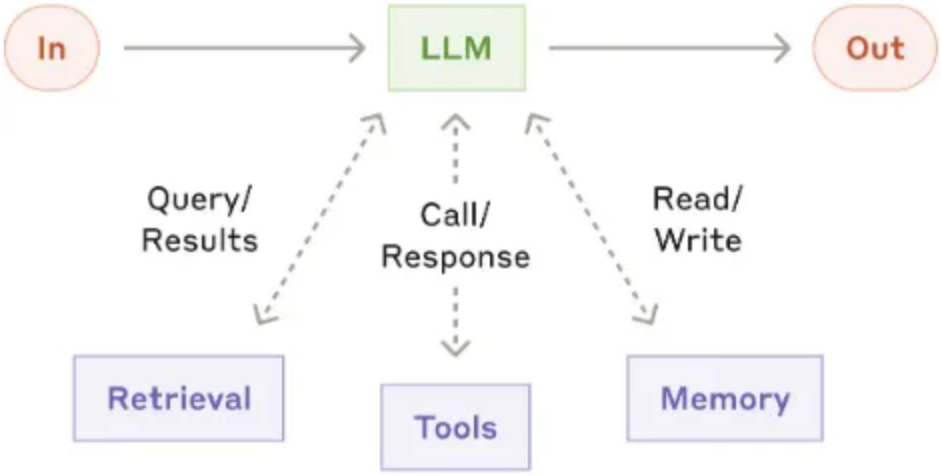
- Previous Code Blocks: Context includes existing code, function names, and comments.
- Programming Language Syntax: The model understands language-specific patterns.
- External Documentation: Some models use APIs or docs for more accurate suggestions.

4. Speech and Audio Models (e.g., Whisper, AudioPaLM) receive their context through:

- Audio Segments: Prior speech or music informs the next generated part.
- Linguistic and Acoustic Features: Tone, speed, and intonation influence transcription and generation.

In short, context is the key factor that enables generative AI to produce relevant and coherent outputs. The better the context management, the better the AI's performance.

Over time the AI models can auto fetch data to act as context. This is especially true of AI Agents which are the systems that use generative AI models at their core. This means the AI Agents have to search for data sources, request the sources for specific data and so on.



https://www.anthropic.com/engineering/building-effective-agents

Each data source (server) is implemented is its own way (for example, as open source packages in another codebase — rather than emitting messages that can be consumed by anyone. Or these can be implemented as JSON RPC for messages) so there is no standard way for an AI model (client) to search for and request for data. (Fragmentation.)

Before MCP, building AI systems often involved:

- Custom implementations for each AI application to hook into its required context, leading to a lot of duplicated effort.
- Inconsistent prompt logic and different methods for accessing and federating tools and data across different teams and companies.
- The “N times M problem” where a large number of client applications needed to interact with a large number of servers and tools, resulting in a complex web of integrations, each requiring specific development work.

Model Context Protocol (MCP) solves this problem of fragmented data access.

The MCP provides a open standard for connecting AI systems with data sources and tools (repositories, business tools, development environments), replacing fragmented integrations with a single protocol. Thus MCP provides fungibility between AI clients and servers.

Thus MCP provides a standardized way for applications to:

- Share contextual information with language models
- Expose tools and capabilities to AI systems
- Build composable integrations and workflows

The protocol uses JSON-RPC 2.0 messages to establish communication between:

- **Hosts**: LLM applications that initiate connections
- **Clients**: Connectors within the host application
- **Servers**: Services that provide context and capabilities

There are a number of popular AI tools that support MCP, including:

- Cursor
- Windsurf (Codium)
- Cline (VS Code extension)
- Claude desktop
- Claude code

MCP takes some inspiration from the Language Server Protocol, which standardizes how to add support for programming languages across a whole ecosystem of development tools. In a similar way, MCP standardizes how to integrate additional context and tools into the ecosystem of AI applications.

Architecture of MCP

MCP follows a client-host-server architecture where each host can run multiple client instances.

- This architecture enables users to integrate AI capabilities across applications while maintaining clear security boundaries and isolating concerns.
- Built on JSON-RPC, MCP provides a stateful session protocol focused on context exchange and sampling coordination between clients and servers.

https://spec.modelcontextprotocol.io/specification/2024-11-05/architecture/

Host

The host process acts as the container and coordinator:

- Creates and manages multiple client instances
- Controls client connection permissions and lifecycle
- Enforces security policies and consent requirements
- Handles user authorization decisions
- Coordinates AI/LLM integration and sampling
- Manages context aggregation across clients

Clients

Each client is created by the host and maintains an isolated server connection:

- Establishes one stateful session per server
- Handles protocol negotiation and capability exchange
- Routes protocol messages bidirectionally
- Manages subscriptions and notifications
- Maintains security boundaries between servers

A host application creates and manages multiple clients, with each client having a 1:1 relationship with a particular server.

MCP Clients are the **AI applications or agents** that want to access external systems, tools, or data sources. Examples include Anthropic's first-party applications, Curser, Windsurf, and agents like Goose. The key characteristic of an MCP client is its **MCP compatibility**, meaning it is built to communicate using the standardised interfaces defined by the protocol: **prompts, tools, and resources.**

Once an MCP client is compatible, it can connect to **any MCP server with minimal or no additional work.** The client is responsible for **invoking tools, querying for resources, and interpolating prompts.**

In the context of tools, the **language model within the client application** decides when it is best to invoke the tools exposed by the server. For resources, the client application has **control over how the data exposed by**

the server is used. Prompts are considered **user-controlled tools invoked by the user through the client application.**

Servers

Servers provide specialized context and capabilities:

- Expose resources, tools and prompts via MCP primitives
- Operate independently with focused responsibilities
- Request sampling through client interfaces
- Must respect security constraints
- Can be local processes or remote services

MCP Servers act as **wrappers or intermediaries** that provide a **standardised way to access various external systems, tools, and data sources.** An MCP server can provide access to databases, CRMs like Salesforce, local file systems, and version control systems like Git. The role of the server builder is to **expose tools, resources, and prompts** in a way that is consumable by any compatible client. Once an MCP server is built, it can be adopted by **any MCP client**, solving the "N times M problem" by reducing the need for individualised integrations. For **tools**, the server defines the available functions and their descriptions, allowing the client's model to decide when to use them. For **resources**, the server defines and potentially creates or retrieves data that it exposes to the client application. For **prompts**, the server provides predefined templates for common interactions that the client application can trigger on behalf of the user.

The MCP protocol acts as the **communication layer** between these two components, standardising how requests and responses are structured and

exchanged. This separation offers several benefits, as it allows:

- **Seamless Integration:** Clients can connect to a wide range of servers without needing to know the specifics of each underlying system.
- **Reusability:** Server developers can build integrations once and have them accessible to many different client applications.
- **Separation of Concerns:** Different teams can focus on building client applications or server integrations independently. For example, an infrastructure team can manage an mCP server for a vector database, which can then be easily used by various AI application development teams.

In essence, the relationship between MCP clients and servers is one of **standardised interaction**, where clients leverage the capabilities exposed by servers through the common language of the MCP protocol, leading to a more efficient and scalable ecosystem for building AI applications and agents.

MCP Features

MCP Server Features

MCP Servers provide the fundamental building blocks (Prompts, Resources, Tools) for adding context to language models via MCP. These primitives enable rich interactions between clients, servers, and language models:

- **Prompts**: Pre-defined templates or instructions that guide language model interactions
- **Resources**: Structured data or content that provides additional context to the model

- **Tools:** Executable functions that allow models to perform actions or retrieve information

https://spec.modelcontextprotocol.io/specification/2024-11-05/server/

The Model Context Protocol (MCP) provides a standardized way for servers to expose prompts, resources, and tools to clients.

Prompts (Protocol Revision: 2024–11–05)

Prompts allow servers to provide structured messages and instructions for interacting with language models. Clients can discover available prompts, retrieve their contents, and provide arguments to customize them.

Prompts are designed to be **user-controlled**, meaning they are exposed from servers to clients with the intention of the user being able to explicitly select them for use.

Typically, prompts would be triggered through user-initiated commands in the user interface, which allows users to naturally discover and invoke available prompts. For example, as slash commands.

Servers that support prompts **MUST** declare the `prompts` capability during initialization:

https://spec.modelcontextprotocol.io/specification/2024-11-05/server/prompts/

Resources : Protocol Revision: 2024–11–05

Resources allow servers to share data that provides context to language models, such as files, database schemas, or application-specific information. Each resource is uniquely identified by a URI.

Resources in MCP are designed to be **application-driven**, with host applications determining how to incorporate context based on their needs.

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

Servers that support resources **MUST** declare the `resources` capability:

https://spec.modelcontextprotocol.io/specification/2024-11-05/server/resources/

The capability supports two optional features:

- `subscribe` : whether the client can subscribe to be notified of changes to individual resources.
- `listChanged` : whether the server will emit notifications when the list of available resources changes.

Tools — Protocol Revision: 2024–11–05

MCP allows servers to expose tools that can be invoked by language models. Tools enable models to interact with external systems, such as querying databases, calling APIs, or performing computations. Each tool is uniquely identified by a name and includes metadata describing its schema.

Tools in MCP 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. However, implementations are free to expose tools through any interface pattern that suits their needs.

Servers that support tools **MUST** declare the `tools` capability:

https://spec.modelcontextprotocol.io/specification/2024-11-05/server/tools/

`listChanged` indicates whether the server will emit notifications when the list of available tools changes.

MCP Client Features

Clients can implement additional features to enrich connected MCP servers: **Roots and Sampling**.

Roots

- **Roots** define the boundaries of **where servers can operate within the filesystem,** allowing them to understand which directories and files they have access to. MCP provides a standardized way for clients to expose filesystem "**roots**" to servers. Servers can request the list of roots from supporting clients and receive notifications when that list changes.

A root definition includes:

- `uri` : Unique identifier for the root. This **MUST** be a `file://` URI in the current specification.
- `name` : Optional human-readable name for display purposes.

Example roots for different use cases:

https://spec.modelcontextprotocol.io/specification/2024-11-05/client/roots/

Sampling (Protocol Revision: 2024–11–05)

MCP provides a standardized way for servers to request LLM sampling (“completions” or “generations”) from language models via clients. This flow allows clients to maintain control over model access, selection, and permissions while enabling servers to leverage AI capabilities — with no server API keys necessary. Servers can request text or image-based interactions and optionally include context from MCP servers in their prompts.

Sampling in MCP allows servers to implement agentic behaviors, by enabling LLM calls to occur *nested* inside other MCP server features.

Implementations are free to expose sampling through any interface pattern that suits their needs — the protocol itself does not mandate any specific user interaction model.

Composability

- Composability in MCP highlights that the **distinction between a client and a server is logical rather than physical**. This means that **any application, API, or agent can function as both an MCP client and an MCP server simultaneously**.
- This dual role allows for the creation of **layered and chained systems**. A user might interact with a primary agent application (a client), which then communicates with a specialised sub-agent (acting as a server). This sub-agent, in turn, can act as a client and invoke other MCP servers (such as a file system server or a web search server) to fulfil its task.
- **Relevance to Agents:** Composability is crucial for building advanced, modular agent architectures. It enables the creation of **hierarchical systems of agents**, where different agents can specialise in specific tasks and delegate sub-tasks to other agents. For instance, an orchestrator agent can receive a high-level goal and then break it down into smaller tasks, delegating these tasks to research agents, coding agents, or fact-checking agents, each operating as an MCP server but also potentially acting as a client to access necessary tools and data. This allows for **building complex workflows and intelligent behaviours by combining the capabilities of multiple specialised agents**. It also allows for **reusing and connecting to agents built by others**, even if they were not initially part of the main agent's design.

In combination, **sampling and composability** are powerful enablers for advanced AI agents. They allow for:

- **Distribution of intelligence** across a multi-agent system, with the client controlling the actual LLM interactions while servers (agents) can request these capabilities as needed through sampling.
- The construction of **complex, multi-layered agent systems** where specialised agents can work together by acting as both clients and servers.
- **Increased flexibility and modularity** in agent design, as new capabilities (exposed as mCP servers) can be integrated into existing agent workflows.
- The potential for **agents to evolve and adapt** by interacting with other agents and services in a composable manner.

These concepts move beyond monolithic agent designs and towards more distributed, collaborative, and adaptable AI systems.

Additional Utilities offered by MCP :

- Configuration, Progress tracking, Cancellation, Error reporting, Logging

Security and Trust & Safety

MCP enables powerful capabilities through arbitrary data access and code execution paths. With this power comes important security and trust considerations that all implementors must carefully address.

Key Principles of MCP Security, Trust and Safety

1. **User Consent and Control:** Users must explicitly consent to and understand all data access and operations. They must retain control over

what data is shared and what actions are taken. Implementors should provide clear UIs for reviewing and authorizing activities

2. Data Privacy: Hosts must obtain explicit user consent before exposing user data to servers. Hosts must not transmit resource data elsewhere without user consent. User data should be protected with appropriate access controls

3. Tool Safety: Tools represent arbitrary code execution and must be treated with appropriate caution. Hosts must obtain explicit user consent before invoking any tool. Users should understand what each tool does before authorizing its use

4. **LLM Sampling Controls:** Users must explicitly approve any LLM sampling requests. Users should control — Whether sampling occurs at all, The actual prompt that will be sent, What results the server can see, The protocol intentionally limits server visibility into prompts

Implementation Guidelines: Implementors of MCP should:

1. Build robust consent and authorization flows into their applications
2. Provide clear documentation of security implications
3. Implement appropriate access controls and data protections
4. Follow security best practices in their integrations
5. Consider privacy implications in their feature designs

MCP Design Principles

MCP is built on several key design principles that inform its architecture and implementation:

1. **Servers should be extremely easy to build**

- Host applications handle complex orchestration responsibilities
- Servers focus on specific, well-defined capabilities
- Simple interfaces minimize implementation overhead
- Clear separation enables maintainable code

2. Servers should be highly composable

- Each server provides focused functionality in isolation
- Multiple servers can be combined seamlessly
- Shared protocol enables interoperability
- Modular design supports extensibility

3. Servers should not be able to read the whole conversation, nor "see into" other servers

- Servers receive only necessary contextual information
- Full conversation history stays with the host
- Each server connection maintains isolation
- Cross-server interactions are controlled by the host
- Host process enforces security boundaries

4. Features can be added to servers and clients progressively

- Core protocol provides minimal required functionality

- Additional capabilities can be negotiated as needed
- Servers and clients evolve independently
- Protocol designed for future extensibility
- Backwards compatibility is maintained

MCP Message Types

MCP defines three core message types based on JSON-RPC 2.0:

- **Requests:** Bidirectional messages with method and parameters expecting a response
- **Responses:** Successful results or errors matching specific request IDs
- **Notifications:** One-way messages requiring no response

Each message type follows the JSON-RPC 2.0 specification for structure and delivery semantics.

Capability Negotiation System in MCP

MCP uses a capability-based negotiation system where clients and servers explicitly declare their supported features during initialization. Capabilities determine which protocol features and primitives are available during a session.

- Servers declare capabilities like resource subscriptions, tool support, and prompt templates
- Clients declare capabilities like sampling support and notification handling
- Both parties must respect declared capabilities throughout the session

- Additional capabilities can be negotiated through extensions to the protocol

Each capability unlocks specific protocol features for use during the session. For example:

- Implemented server features must be advertised in the server's capabilities
- Emitting resource subscription notifications requires the server to declare subscription support
- Tool invocation requires the server to declare tool capabilities
- Sampling requires the client to declare support in its capabilities

This capability negotiation ensures clients and servers have a clear understanding of supported functionality while maintaining protocol extensibility.

https://spec.modelcontextprotocol.io/specification/2024-11-05/architecture/

Details of the Base MCP Protocol

Protocol Revision: 2024–11–05

All messages between MCP clients and servers **MUST** follow the JSON-RPC 2.0 specification. The protocol defines three fundamental types of messages:

https://spec.modelcontextprotocol.io/specification/2024-11-05/basic/

Responses are further sub-categorized as either **successful results** or **errors.** Results can follow any JSON object structure, while errors must include an error code and message at minimum.

Protocol Layers

The Model Context Protocol consists of several key components that work together:

- **Base Protocol:** Core JSON-RPC message types
- **Lifecycle Management:** Connection initialization, capability negotiation, and session control
- **Server Features:** Resources, prompts, and tools exposed by servers
- **Client Features:** Sampling and root directory lists provided by clients
- **Utilities:** Cross-cutting concerns like logging and argument completion

All implementations **MUST** support the base protocol and lifecycle management components. Other components **MAY** be implemented based on the specific needs of the application.

These protocol layers establish clear separation of concerns while enabling rich interactions between clients and servers. The modular design allows implementations to support exactly the features they need.

Life Cycle for client-server connections:

The Model Context Protocol (MCP) defines a rigorous lifecycle for client-server connections that ensures proper capability negotiation and state management.

1. **Initialization**: Capability negotiation and protocol version agreement
2. **Operation**: Normal protocol communication
3. **Shutdown**: Graceful termination of the connection

Lifecycle Phases:

1. **Initialization:** The initialization phase **MUST** be the first interaction between client and server. During this phase, the client and server:

- Establish protocol version compatibility
- Exchange and negotiate capabilities
- Share implementation details

The client **MUST** initiate this phase by sending an `initialize` request containing:

- Protocol version supported
- Client capabilities

- Client implementation information

The server **MUST** respond with its own capabilities and information.

After successful initialization, the client **MUST** send an `initialized` notification to indicate it is ready to begin normal operations.The client **SHOULD NOT** send requests other than pings before the server has responded to the `initialize` request. The server **SHOULD NOT** send requests other than pings and logging before receiving the `initialized` notification.

2. Version Negotiation: In the `initialize` request, the client **MUST** send a protocol version it supports. This **SHOULD** be the *latest* version supported by the client. If the server supports the requested protocol version, it **MUST** respond with the same version. Otherwise, the server **MUST** respond with another protocol version it supports. This **SHOULD** be the *latest* version supported by the server. If the client does not support the version in the server's response, it **SHOULD** disconnect.

3. Capability Negotiation: Client and server capabilities establish which optional protocol features will be available during the session.

Key capabilities include:

https://spec.modelcontextprotocol.io/specification/2024-11-05/basic/lifecycle/

Capability objects can describe sub-capabilities.

4. Operation: During the operation phase, the client and server exchange messages according to the negotiated capabilities.

Both parties **SHOULD:**

- Respect the negotiated protocol version
- Only use capabilities that were successfully negotiated

5. Shutdown: During the shutdown phase, one side (usually the client) cleanly terminates the protocol connection. No specific shutdown messages are defined — instead, the underlying transport mechanism should be used to signal connection termination:

stdio

For the stdio transport, the client **SHOULD** initiate shutdown by:

1. First, closing the input stream to the child process (the server)
2. Waiting for the server to exit, or sending `SIGTERM` if the server does not exit within a reasonable time
3. Sending `SIGKILL` if the server does not exit within a reasonable time after `SIGTERM`

The server **MAY** initiate shutdown by closing its output stream to the client and exiting.

6. HTTP: For HTTP transports, shutdown is indicated by closing the associated HTTP connection(s).

7. Error Handling: Implementations **SHOULD** be prepared to handle these error cases:

- Protocol version mismatch
- Failure to negotiate required capabilities
- Initialize request timeout
- Shutdown timeout

Implementations **SHOULD** implement appropriate timeouts for all requests, to prevent hung connections and resource exhaustion.

Authentication and authorization are not currently part of the core MCP specification, but may be introduced soon.

The full specification of the protocol is defined as a TypeScript schema. This is the source of truth for all protocol messages and structures.

There is also a JSON Schema, which is automatically generated from the TypeScript source of truth, for use with various automated tooling.

Mindmap of MCP Protocol

Benefits of MCP for Stakeholders

For Application Developers

For application developers, the MCP offers several key benefits.

- **Zero Additional Work for Server Connection:** Once an application is **MCP compatible**, it can connect to **any mCP server with zero additional work**. This means developers don't need to write specific integration logic for each new tool or data source they want their application to access, significantly reducing development time and effort.
- **Standardised Interface:** MCP standardises how AI applications interact with external systems through its three primary interfaces: **prompts, tools, and resources**. This provides a **consistent way to access and utilise the capabilities** offered by different servers, simplifying the development

process and making it easier for developers to understand and integrate new functionalities.

- **Access to a Broad Ecosystem:** By building an MCP client, developers gain access to a growing ecosystem of **community-built and officially supported** MCP **servers.** This allows them to easily integrate a wide range of functionalities, such as accessing databases, CRMs, local file systems, and more, without having to build these integrations themselves. The upcoming **mCP registry API** will further enhance this by providing a **centralised way to discover and pull in** MCP **servers.**
- **Focus on Core Application Logic:** MCP allows application developers to **focus on the core logic and user experience of their AI application** rather than spending time on the complexities of integrating with various external systems. The protocol handles the underlying communication and standardisation, freeing up developers to concentrate on the unique value proposition of their application. As Mahesh explained, developers can focus on the "agent Loop" and context management, while MCP handles the standardised way of bringing context in.
- **Leveraging Model Intelligence for Tool Use:** The "tools" interface allows developers to expose functionalities to the language model within their application, enabling the **model itself to intelligently decide when and how to invoke these tools**. This reduces the need for developers to explicitly program every interaction with external systems, making the application more dynamic and responsive.
- **Richer User Interactions:** The "resources" interface provides a way for servers to expose data beyond simple text, such as images and structured data. This enables application developers to create **richer and more interactive experiences** for their users.

The Model Context Protocol (mCP) offers distinct benefits to Tool/API providers, end users, and enterprises as well:

Tool/API Providers:

- **Increased Adoption:** By building an MCP **server once**, Tool or API providers can see **adoption of their services across a wide range of** MCP-**compatible AI applications**. This eliminates the need to build individual integrations for each client application, significantly expanding their potential user base. As Mahesh put it, they can "build your mCP server once and see adoption of it everywhere across all of these different AI applications".
- **Simplified Integration:** MCP provides a **standardised way to expose their tools and data** to AI applications through the prompts, tools, and resources interfaces. This simplifies the process for application developers to integrate with their services, making it more likely they will be adopted.
- **Reduced Integration Overhead:** Providers no longer need to deal with the **"N times M problem"** of building and maintaining numerous custom integrations for different AI clients. MCP acts as a unifying layer, streamlining the integration process.
- **Access to Intelligent Agents:** MCP allows Tool/API providers to make their services accessible to **increasingly intelligent agents** that can autonomously decide when and how to utilise their capabilities. This can lead to new and innovative ways their tools and data are used.
- **Potential for New Use Cases:** By exposing their services through MCP, providers can unlock **new use cases and applications** they may not have

previously considered, as AI agents can leverage their tools in unforeseen ways.

End Users:

- **More Powerful and Context-Rich AI Applications:** MCP leads to the development of **more powerful and personalised AI applications** that have seamless access to relevant data and tools. This results in a more effective and useful user experience. As Mahesh mentioned, end users benefit from "more powerful and context-rich AI applications".
- **Context-Aware Interactions:** Applications built with MCP can be **more context-aware**, understanding the user's current situation and accessing the necessary information to provide relevant assistance. Examples like Curser and Windsurf demonstrate how MCP enables systems to "actually know things about you and can go and take action in the real world".
- **Seamless Integration with Familiar Tools:** MCP allows AI applications to **integrate seamlessly with the tools and services users already rely on**, such as GitHub, Asana, and more. This provides a more unified and efficient workflow.
- **More Intelligent Assistance:** Agents powered by MCP can **intelligently utilise a wider range of tools and data**, leading to more capable and helpful assistance in various tasks. The ability for agents to discover and use new tools dynamically through an MCP registry (once available) will further enhance this.
- **Customisable and Personalised Experiences:** Users may benefit from AI applications that can be **customised with their own data sources and preferred tools** through the MCP framework.

Enterprises:

- **Standardised AI Development:** MCPprovides a **clear and standardised way to build AI systems** within an organisation, reducing the fragmentation and duplicated effort often seen across different teams. This allows for a more cohesive and efficient approach to AI development.
- **Separation of Concerns:** MCP enables a **clear separation of concerns** between teams responsible for infrastructure (like vector databases) and teams building AI applications. This allows each team to focus on their core expertise and move faster. For example, one team can manage a vector database mCP server, while other teams can easily access it without needing to understand its underlying implementation.
- **Faster Development Cycles:** With standardised interfaces and readily available MCP servers, enterprise teams can **build and deploy AI applications more quickly**. They don't need to spend time on custom integrations for every data source or tool.
- **Centralised Access Control and Governance (in the future):** While still evolving, the concepts around remote servers and authentication (OAuth 2.0) within MCP lay the groundwork for **more centralised control over data access** for AI applications. The future mCP registry can also contribute to governance by providing a way to verify and manage approved servers within the enterprise.
- **Improved Scalability and Maintainability:** The standardised nature of MCP can lead to **more scalable and maintainable AI systems** as integrations are consistent and less prone to breaking due to changes in individual APIs.

- **Leveraging Existing Infrastructure:** Enterprises can **wrap their existing tools and data sources with** MCP **servers**, making them easily accessible to AI applications without requiring significant re-architecting.

MCP Registry API (Currently under development)

The purpose of the **MCP registry API currently under development is to provide a centralised way to discover, verify, and manage MCP servers**.

The key challenges that the MCP registry API aims to address include:

- **Discoverability:** Currently, finding relevant mCP servers is fragmented. The registry will serve as a **unified metadata service** making it easier for users and applications to locate available servers and their capabilities.
- **Protocol Information:** The registry will provide information about the **protocol used by a server** (e.g., standard IO or SSSE) and its location (local file or a remote URL).
- **Trust and Verification:** A significant challenge is determining the **trustworthiness and authenticity of mCP servers**. The registry aims to address this by potentially offering mechanisms for **verification**, such as indicating official servers from reputable companies like Shopify or Grafana.
- **Publication:** The registry will simplify the process for developers to **publish their mCP servers** and make them accessible to a wider audience.
- **Versioning:** As mCP servers evolve, the registry will provide a way to manage and track **different versions** of servers and their capabilities. This will help users understand what has changed and potentially pin to specific versions if needed.

- **Metadata Management:** The registry will host **metadata about mCP servers**, including their capabilities (resources and tools), potentially making it easier to understand how to interact with them.

Remote servers and the integration of OAuth 2.0 within the mCP protocol.

Significance of Remote Servers:

- Currently, MCP often relies on local or in-memory communication using standard IO, which can introduce friction in terms of setup and deployment. **Remote servers, facilitated by protocols like SSE (Server-Sent Events), allow MCP servers to be hosted on public URLs**, making them accessible over the internet.
- This development **removes the need for users to understand the intricacies of MCP server hosting or building.** Much like accessing a website, users or applications can potentially connect to a remote mCP server simply via a URL.
- Remote servers **decouple the location of the MCP client (e.g., an AI agent or application) from the MCP server**, allowing them to run on completely different systems. This enhances flexibility in architectural design and deployment.
- This increased availability of servers will broaden the range of capabilities accessible through mCP.

Integration of OAuth 2.0:

- The integration of OAuth 2.0 provides a **standardised and secure mechanism for authentication and authorisation** between mCP clients

and remote servers.

- MCPl now supports an OAuth 2.0 handshake where the **server orchestrates the authentication flow**, interacting with the OAuth provider (e.g., Slack). The client (user) typically authorises the connection through a familiar web-based flow.
- Once authenticated, the **server holds the OAuth token** and can then provide the client with a session token for subsequent interactions. This approach **gives the server more control over the interaction with the underlying service** (like Slack).
- This secure authentication mechanism is crucial for enabling access to sensitive data and functionalities offered by remote servers. It **builds trust** and provides a framework for managing permissions.

Impact on Accessibility and Usability:

These advancements are likely to have a profound positive impact on the accessibility and usability of mCP servers:

- **Reduced Barrier to Entry:** The ease of connecting to remote servers via URLs significantly lowers the technical barrier for both developers wanting to utilise MCP capabilities in their applications and end-users interacting with those applications. Users may not even need to be aware that they are interacting with an mCP server.
- **Wider Range of Applications:** The ability to host servers remotely opens up possibilities for a broader spectrum of applications to leverage MCP. Web-based applications, mobile apps, and cloud services can now more easily integrate with MCP servers without requiring complex local setups.

- **Increased Server Availability:** The reduced friction in development and deployment will likely lead to a proliferation of MCP servers offering diverse functionalities. This expanded ecosystem will provide a richer set of tools and resources for AI applications.
- **Improved Security and Trust:** The adoption of OAuth 2.0 provides a robust and widely recognised standard for secure access to remote resources. This is essential for building user confidence and encouraging the use of MCP servers that interact with personal or sensitive data. As you noted in your previous turn regarding trust and verification in the context of the registry, a secure authentication mechanism is a fundamental building block for a trustworthy ecosystem.
- **Simplified Development:** Developers can focus more on building the core logic of their applications and servers, rather than dealing with the complexities of local communication and custom authentication methods. The standardised OAuth 2.0 flow simplifies the integration process.

Self-evolving agents enabled by an mCP server registry.

An MCP **server registry** allows agents to **dynamically discover new capabilities and data on the fly without having been explicitly programmed with that knowledge from the outset**. This means an agent can encounter a new task or require access to a previously unknown data source and proactively seek out the necessary tools to fulfil that need.

Consider the example of a general coding agent tasked with checking Grafana logs. If the agent wasn't initially configured with the specifics of a Grafana server, it could interact with the **MCP registry**. By searching for an **official and verified Grafana server** offering the necessary APIs, the agent can then **install or invoke** that server (potentially remotely via SSE, as

discussed previously) and proceed with querying the logs and addressing the bug.

Potential Advantages of Self-Evolving Agents:

- **Enhanced Adaptability:** The most significant advantage is the agent's **increased ability to adapt to novel situations and tasks**. Instead of being limited to its pre-programmed capabilities, it can learn and expand its functionality as needed.
- **Broader Range of Applications:** Self-evolving agents can tackle a **wider variety of tasks** without requiring developers to anticipate every possible scenario during the initial design and programming phase.
- **Improved User Experience:** Users can interact with more general-purpose agents that can handle a diverse set of requests, even those involving systems or data sources the agent was not explicitly built to use. The agent takes on the responsibility of finding the right tools.
- **Continuous Improvement:** Agents can effectively **"learn" and improve over time** by discovering and integrating new and potentially more efficient tools and data sources as they become available in the registry.
- **Reduced Development Overhead:** Developers can focus on building the core reasoning and task execution logic of the agent, relying on the registry to provide access to a constantly growing ecosystem of capabilities. This reduces the need to build bespoke integrations for every potential data source or service.

Considerations Surrounding Self-Evolving Agents:

- **Governance and Security:** A crucial consideration is **controlling which servers and capabilities an agent can access**. Without proper safeguards, a self-evolving agent could potentially connect to untrusted or even malicious servers, posing security risks or data privacy concerns. As Mahesh mentioned, approaches such as **self-hosted registries with approved servers, whitelisting, and verification mechanisms** will be essential to mitigate these risks. The concept of trust in servers becomes increasingly important, as highlighted in our previous discussion.
- **Trust and Verification:** Ensuring the **reliability and trustworthiness of the discovered servers** is paramount. The **mCP registry aims to address this by providing a mechanism for verification**, potentially allowing official entities (like Shopify or Grafana) to "bless" their servers.
- **Performance and Efficiency:** The process of discovering and integrating new servers dynamically could introduce **latency** into the agent's workflow. Optimising the registry search and server integration processes will be important.
- **Tool Selection and Reasoning:** While models are becoming increasingly adept at tool use, there's still a need to ensure the agent **makes intelligent decisions about which tools to select and how to utilise them effectively** from the potentially vast number of options available in the registry. Overwhelming the agent with too many options could also be a challenge.
- **Debugging and Observability:** As agents become more complex and rely on dynamically discovered components, **debugging and monitoring their behaviour** could become more challenging. Clear mechanisms for observing the agent's interactions with the registry and the invoked servers will be necessary. As discussed previously, best practices for

server debuggability and client-server communication of metadata will be important.

- **Versioning and Compatibility:** As servers evolve and their APIs change, there's a need for **versioning mechanisms** within the registry to ensure compatibility with existing agents. Agents might need to be aware of different server versions and potentially handle compatibility issues.

References:

[1]. Mahesh Murag's MCP Workshop during AI Engineer Summit- https://www.youtube.com/watch?v=kQmXtrmQ5Zg

[2]. https://supabase.com/docs/guides/getting-started/mcp

[3]. https://www.anthropic.com/engineering/building-effective-agents

[4]. https://www.anthropic.com/news/model-context-protocol

[5]. https://spec.modelcontextprotocol.io/specification/2024-11-05/

[6]. Getting Started: https://modelcontextprotocol.io/introduction

[7]. Python SDK: https://github.com/modelcontextprotocol/python-sdk

[8]. TypeScript SDK: https://github.com/modelcontextprotocol/typescript-sdk

Model Context Protocol | AI | Generative Ai Tools

Written by Dr. Nimrita Koul

668 followers · 305 following

https://www.linkedin.com/in/nimritakoul/

Follow

Responses (5)

Dimitar Gueorguiev

What are your thoughts?

Mustafa Yagmur
Apr 4

I really wonder how much of this article AI generated? It is hard to read long context paragraphs which usually a sign of AI generation to me

35 3 replies Reply

R. Thompson (PhD) he/him
Mar 31

MCP registry = PyPI meets ChatGPT plugins.

12 Reply

Bryan O'neal Womble
Apr 8

Very well organized and structured and informative.

8 Reply

See all responses

More from Dr. Nimrita Koul

Dr. Nimrita Koul

Offline Speech to Text in Python

Writing, coding, blogging, office work, documentation, reporting all needs one to...

Feb 2, 2024 399 1

Dr. Nimrita Koul

Image Processing using OpenCV — Python

OpenCV

Dec 20, 2023 434 1

Dr. Nimrita Koul

uv Package Manager for Python

uv is an extremely fast Python package and project manager written in Rust. uv is backe...

Mar 25 164

Dr. Nimrita Koul

Getting started with Gradio Python Library

Gradio is an open-source Python package for creating user interfaces / interactive demos...

Apr 11, 2024 67 1

See all from Dr. Nimrita Koul

Recommended from Medium

Edwin Lisowski

MCP Explained: The New Standard Connecting AI to Everything

How Model Context Protocol is making AI agents actually do things

Apr 15 938 13

Cobus Greyling

Using LangChain With Model Context Protocol (MCP)

The Model Context Protocol (MCP) is an open-source protocol developed by...

Mar 10 1.6K 7

alejandro

Ollama + MCP Servers from Scratch

Integrate Ollama with MCP servers from scratch

Apr 15 388 7

In Predict by Ashraff Hathibelagal

Using the Model Context Protocol (MCP) With a Local LLM

Using MCP to augment a locally-running Llama 3.2 instance.

Mar 26 107 2

Ruchi

Integrating MCP Servers with FastAPI

What is MCP? The AI-API Communication

May 12 581 12

In AI Cloud Lab by Arjun Prabhulal

Model Context Protocol(MCP) with Ollama and Llama 3 : A Step-by-...

In my previous article , we explored Model Context Protocol (MCP)—a standardized wa...

Apr 4 189 3

See more recommendations