Chapter 3. MCP 이론

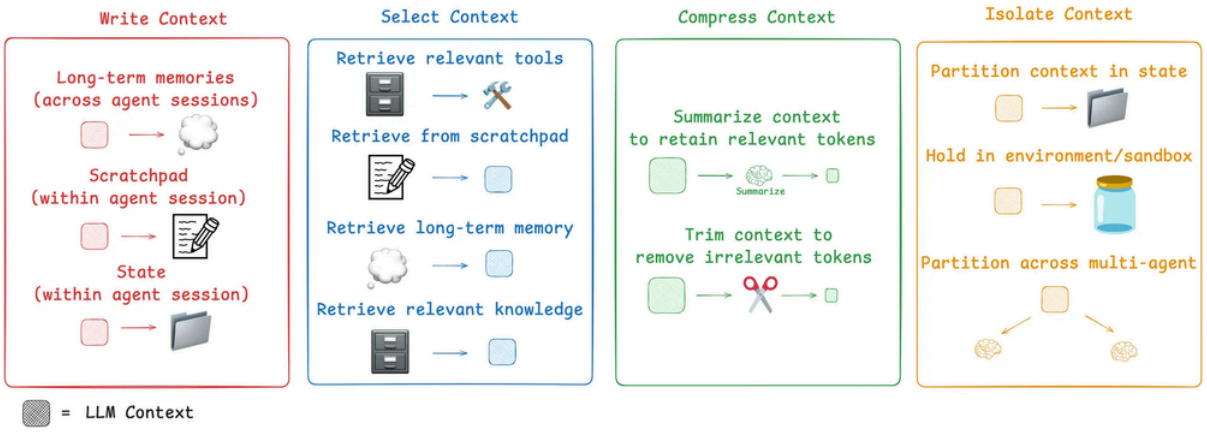
Chapter 3. MCP 완벽 이해

> ⓘ 이 챕터에서는...
>
> Chapter 2에서 구현한 Custom Component를 MCP Server로 전환하기 전에, **Model Context Protocol (MCP)**의 핵심 개념과 작동 원리를 파악합니다.
> 기존 Function Calling 방식의 제약사항을 살펴보고, MCP가 제시하는 해결 방안을 이해합니다.

Model Context Protocol: AI 도구 연동을 위한 개방형 표준

1. Context 중심 AI의 진화

1.1 AI의 실질적 가치는 어디서 나오는가



방대한 지식을 보유하는 것보다, **현재 상황에 필요한 정보를 정확한 시점에 제공**하는 능력이 핵심입니다.

비교:

- 📚 백과사전형 지식 보유 AI < 💡 필요 시점에 정확한 데이터를 검색·제공하는 AI
- 학습된 지식만 활용 < 🔄 최신 정보에 즉시 접근하여 응답

결론: AI의 진정한 능력은 **"지식의 양"** 이 아니라 **"상황별 맥락 활용도"** 에서 결정됩니다.

1.2 Function Calling: 첫 번째 돌파구

초기 방식의 한계:

- LLM 프롬프트 엔지니어링으로 구조화된 출력 유도
- 텍스트 파싱(Text Parsing) 방식의 불안정성
- 원하는 형식과 다른 출력 빈번 발생

Function Calling의 등장 (2023년 7월, OpenAI):

- 자연어 요청을 외부 시스템의 함수 호출로 변환
- **출력 안정성과 신뢰성 대폭 향상**

작동 메커니즘:

1. 함수 스키마 정의 (`Name` , `Description` , `Parameters`)
2. 모델이 적절한 함수와 인자 선택
3. 실제 함수 실행
4. 결과를 Context로 추가
5. 최종 응답 생성

1.3 Function Calling의 구조적 한계

상태 관리의 어려움

- 단일 요청/응답 구조로 설계됨
- 멀티턴 대화에서 컨텍스트를 별도로 관리해야 하는 복잡성

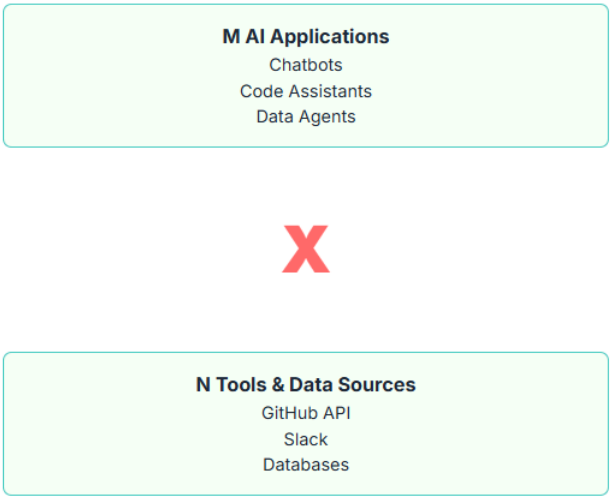
복잡한 워크플로우 처리 한계

- 다단계 작업이나 여러 데이터 소스를 동시에 활용하는 시나리오에 부적합

플랫폼별 파편화

- AI 제공자마다 상이한 함수 호출 방식
- 각 플랫폼별 별도 구현 필요

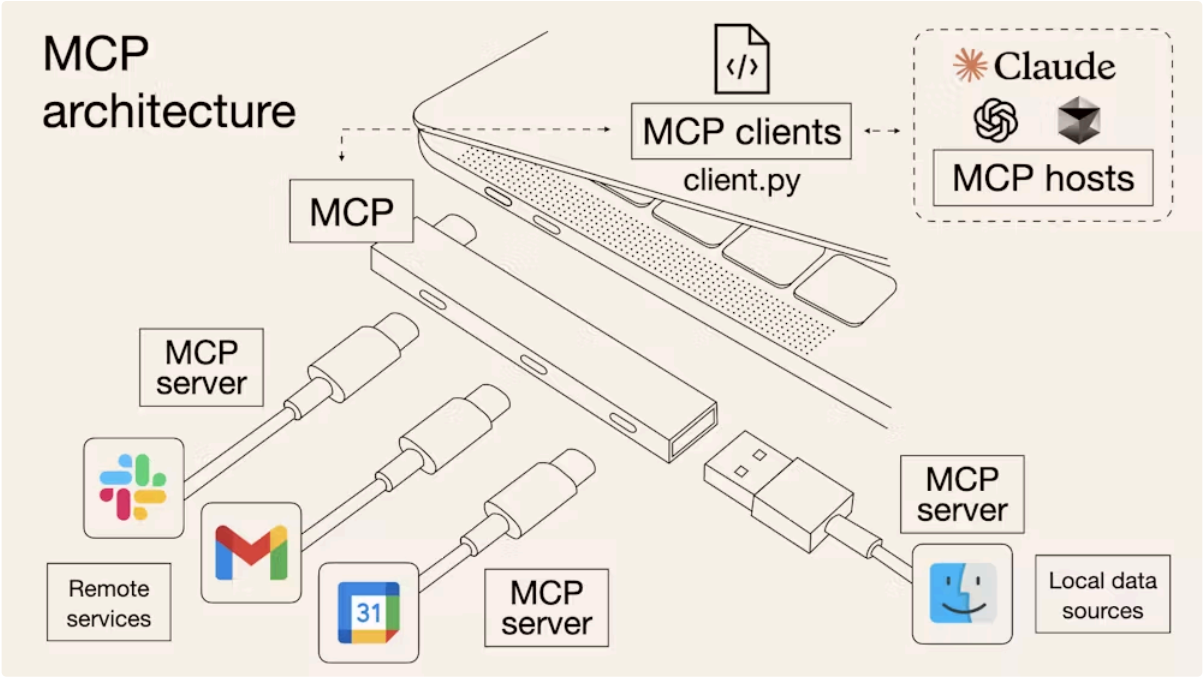
1.4 M × N 통합 복잡도의 본질

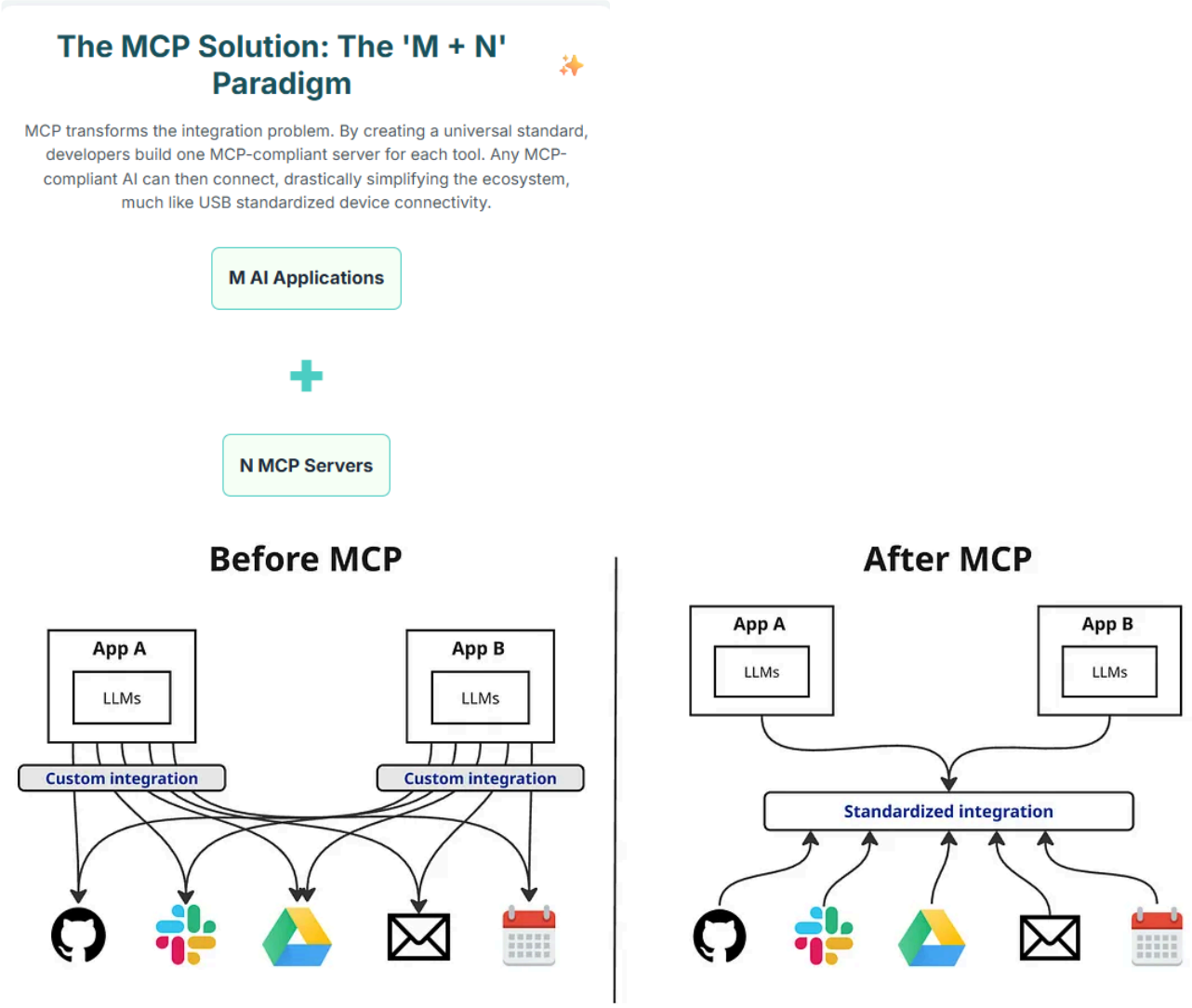


문제 상황:
M개의 AI 애플리케이션이 N개의 데이터 소스/도구와 연결될 때, 이론적으로 M × N개의 개별 통합이 필요합니다.

발생하는 이슈:

- 개발 시간 및 비용의 기하급수적 증가
- 시스템 전체 복잡도 상승
- 유지보수 부담 가중
- 확장성과 유연성 저하





2. MCP의 등장과 핵심 개념

2.1 개발 배경

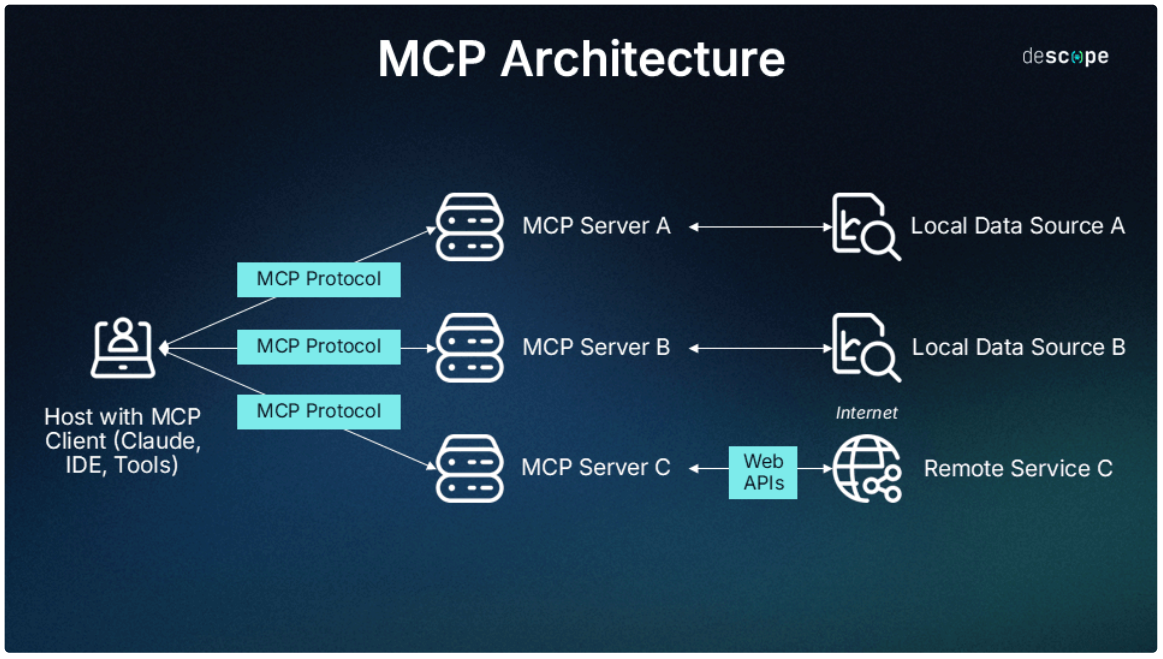
Anthropic의 제안 (2024년 11월):

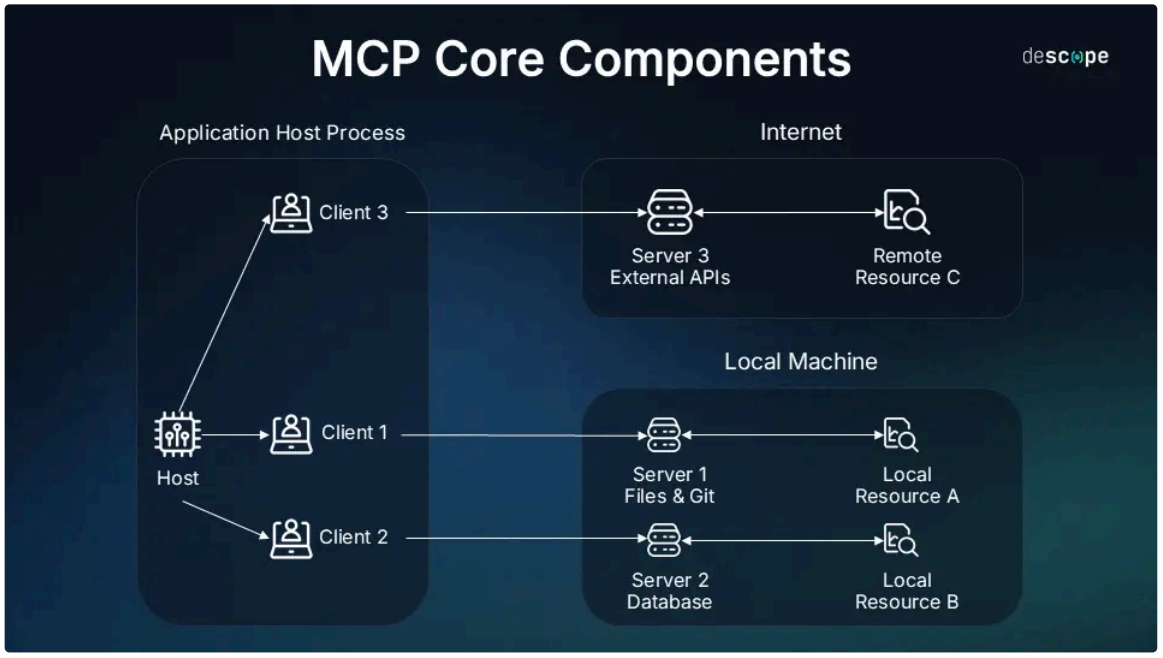
- Claude 개발 과정에서 내부 도구로 출발
- 오픈소스로 전환하여 생태계 확장 추구

업계 채택:

- OpenAI, Microsoft, GitHub 등 주요 AI 기업이 도입
- 사실상의 업계 표준으로 자리잡는 중

2.2 MCP 정의





Model Context Protocol (MCP)
AI 모델과 외부 리소스 간의 컨텍스트 정보를 일관된 방식으로 교환하기 위한 개방형 표준 규약

핵심 특징:

- 파일 접근, 함수 실행, 프롬프트 처리 등을 위한 범용 인터페이스 제공
- 다양한 외부 시스템과 AI 모델 간의 연결을 단일 표준으로 통일
- M × N 문제를 M + N 문제로 전환

제공 가치:
AI가 정보 검색을 넘어서 외부 세계와 상호작용하고 실제 작업을 수행할 수 있는 기반을 제공합니다.

2.3 MCP와 Function Calling의 관계

구분	Function Calling	MCP
역할	작업 지시	작업 실행
계층	LLM 출력 형식화	도구 실행 표준화
제공자	LLM 벤더	개방형 표준
초점	"무엇을" 할지 결정	"어떻게" 실행할지 정의

보완 관계:

- Function Calling: 자연어 → 구조화된 함수 호출로 변환
- MCP: 함수 호출 수신 → 실행 → 응답 처리를 표준화

Function Calling만으로는 각 도구/모델별 맞춤형 실행 로직이 필요하지만, MCP는 도구 로직을 독립시켜 표준화된 호출 방식을 제공합니다.

---

3. MCP 아키텍처

3.1 Host-Client-Server 3계층 구조

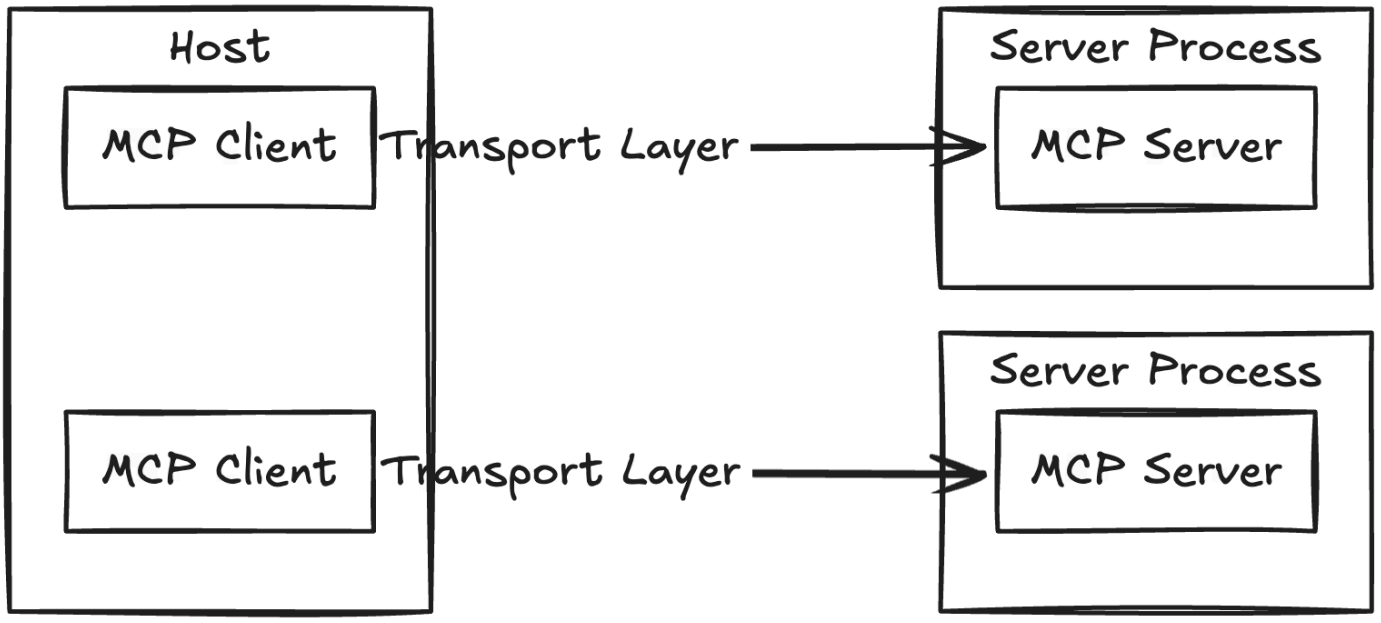
작동 원리:

- Host(AI 애플리케이션)가 여러 Client 인스턴스를 생성 및 관리
- 각 Client는 개별 Server와 1:1 관계로 연결
- 필요한 컨텍스트와 도구를 동적으로 조율

모듈형 구조의 이점:

- AI 애플리케이션 수정 없이 새로운 Server 추가 가능
- 각 Server는 독립적으로 개발 및 운영 가능

3.2 각 계층의 역할



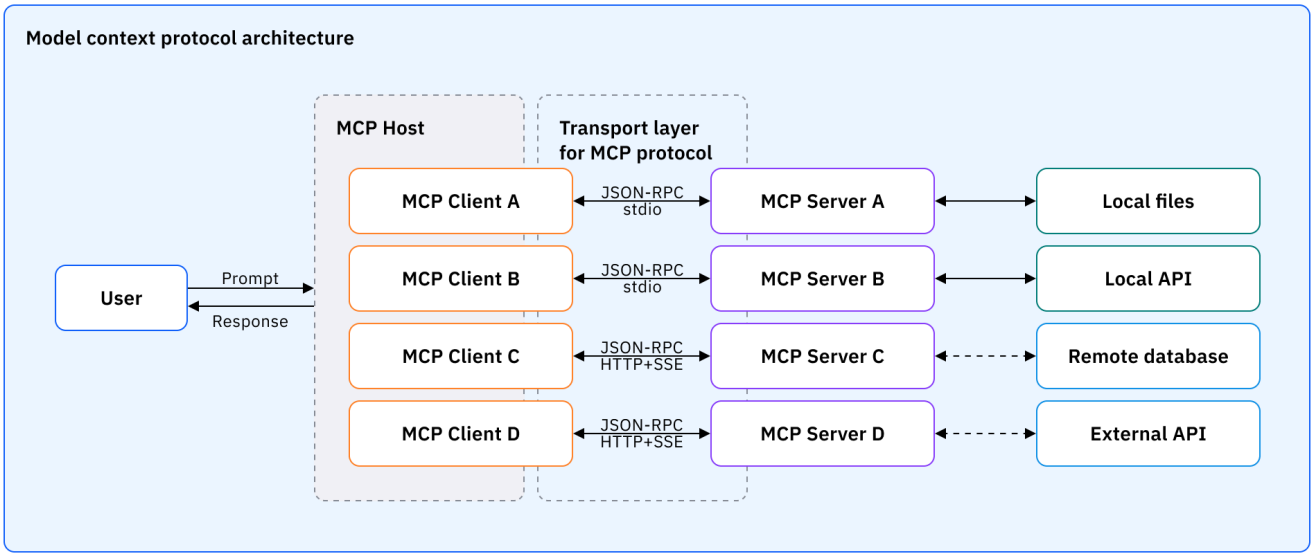
구성 요소	역할	예시
Host	LLM 통합 조율 및 보안·승인 관리 주체	· Claude Desktop, Cursor AI IDE · Client 생성 및 수명주기 관리 · 보안 정책 시행, LLM 응답 통합
Client	프로토콜 어댑터 및 메시지 라우터	· Host 내부에서 생성된 인스턴스 · 개별 Server와 1:1 통신 · JSON-RPC 세션 유지 · Capability Negotiation 수행

구성 요소	역할	예시
Server	특화된 기능 및 데이터 제공자	· 파일 시스템, DB, API 등 · 표준 MCP Primitives로 기능 노출 (Resources, Prompts, Tools) · 독립적 운영, 자신의 Context만 담당

---

4. MCP 통신 프로토콜

4.1 JSON-RPC 2.0 기반 통신



JSON-RPC 2.0 이해하기

JSON-RPC란?
Remote Procedure Call(원격 프로시저 호출)을 위한 경량 프로토콜입니다. 한 프로그램이 원격 서버의 함수를 로컬 함수처럼 호출할 수 있게 하며, 모든 메시지는 JSON 형식으로 전송됩니다.

통신 예시

AI 애플리케이션이 MCP Server의 `add` 함수를 호출하는 상황:

요청 (Client → Server):

```
{
  "jsonrpc": "2.0",
  "method": "add",
  "params": [5, 10],
  "id": 1
}
```

응답 (Server → Client):

```
{
  "jsonrpc": "2.0",
  "result": 15,
  "id": 1
}
```

오류 발생 시 `result` 대신 `error` 객체를 반환하여 문제 상황을 전달합니다.

4.2 전송 계층 (Transport)

Transport	설명
STDIO	· 표준 입출력 활용 로컬 전송 · Client가 Server를 자식 프로세스로 실행 · 간단하고 빠른 로컬 연결에 적합
Streamable HTTP	· HTTP POST + Server-Sent Events(SSE) 결합 · 양방향 통신 및 스트리밍 지원 · 실시간 진행 상황 업데이트 가능

인증 (Authorization):

- 원격 Server 구현 시 권장 (SHOULD)
- OAuth 2.1 표준 활용 가능

4.3 메시지 유형

"JSON-RPC 메시지" 이(가) 아직 생성되지 않았습니다. 클릭해서 생성해보세요.

JSON-RPC 2.0 Specification

Request (요청):

```
{
  "jsonrpc": "2.0",
  "id": "string | number",
  "method": "string",
  "params": { }
}
```

Response (응답):

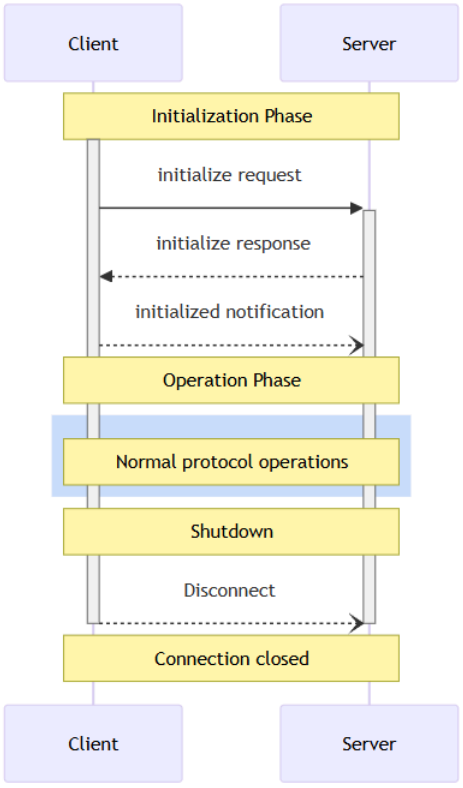
```
{
  "jsonrpc": "2.0",
  "id": "string | number",
  "result": { /* 성공 시 */ },
  "error": { /* 실패 시 */ }
}
```

code	message	meaning
-32700	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-32600	Invalid Request	The JSON sent is not a valid Request object.
-32601	Method not found	The method does not exist / is not available.
-32602	Invalid params	Invalid method parameter(s).
-32603	Internal error	Internal JSON-RPC error.
-32000 to -32099	Server error	Reserved for implementation-defined server-errors.

Notification (알림):

```
{
  "jsonrpc": "2.0",
  "method": "string",
  "params": { }
}
```

4.4 연결 생명주기



초기화 (Initialization):
전송 메커니즘 설정, 서버 및 클라이언트가 `initialized` 알림 전송

1. 클라이언트가 initialized 요청
2. 서버가 응답
3. 클라이언트가 initialized 알림 전송

운영 (Operation):
정상적인 프로토콜 통신, 협상된 기능 사용

종료 (Shutdown):
전송 메커니즘을 통한 연결 종료

---

5. MCP Primitives: 3가지 핵심 제공 방식

5.1 개요

MCP Server는 세 가지 Primitive를 통해 기능을 노출하며, 각각 다른 제어 주체를 갖습니다:

Primitive	제어 주체	설명	예시
Resources	**Application 제어**	LLM이 읽을 수 있는 컨텍스트 데이터 REST API GET과 유사	파일 내용 Git 로그 DB 레코드
Tools	**Model 제어**	LLM이 작업 수행을 위해 호출하는 함수 Function Calling과 유사	날씨 API 호출 DB 쿼리 실행 파일 쓰기
Prompts	**User 제어**	사용자가 트리거하는 사전 정의된 템플릿	슬래시 명령어 메뉴 선택

제어 흐름:

- **Resources**: AI에 정보 제공 (Informing)
- **Tools**: AI의 행동 실행 (Acting)
- **Prompts**: 사용자의 작업 시작 (Initiating)

5.2 Resources (리소스)

서버가 제공하는 데이터 컨텍스트
파일 내용, DB 결과, API 응답 등

특징:

- 애플리케이션 제어 기능으로 고유 URI로 식별
- 클라이언트/사용자 통제 하에 `resources/read`로 조회
- `notifications/resources/updated`로 실시간 변경 알림 가능

5.3 Tools (도구)

서버가 노출하는 실행 가능한 함수/동작
검색, 업데이트, 파일 생성 등

특징:

- 모델 제어 기능으로 AI가 필요 시 스스로 호출
- 사용자 승인 필요
- `readOnly`, `destructive` 등 힌트로 특성 명시 가능
- 기존 Function Calling을 표준화하고 안전하게 확장

5.4 Prompts (프롬프트)

서버가 제공하는 재사용 가능한 프롬프트 템플릿

특징:

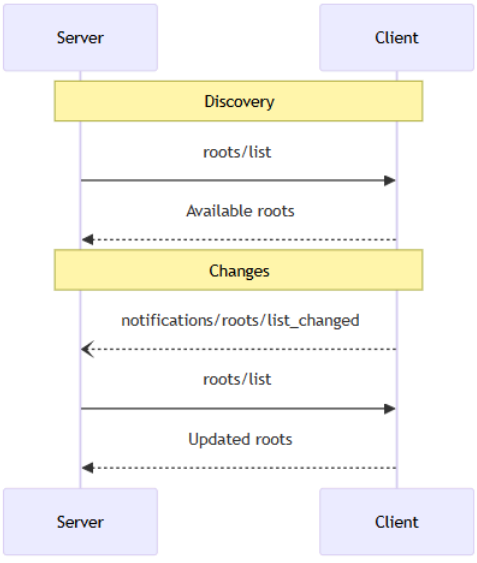
- 사용자의 요청으로 실행 가능
- 동일 반복 작업 자동화
- 일관된 사용자 경험 제공

---

6. 클라이언트측 고급 기능

6.1 Roots (작업 범위 제한)

서버가 작업할 리소스의 범위나 위치를 지정하는 기준 경로 URI



루츠 – 모델 컨텍스트 프로토콜(MCP)

역할:
클라이언트가 서버에게 루트 경로를 제공하여 작업 범위를 제한

가치:
데이터 및 리소스 범위를 제한하여 보안성 및 통제력 강화

6.2 Sampling (역방향 LLM 호출)

서버가 역으로 클라이언트에게 LLM 응답 생성을 요청하는 기능

작동 방식:
서버가 어려운 질문을 받으면 클라이언트의 LLM에 답변 생성을 호출

기대효과:
서버가 단순 데이터 제공을 넘어 LLM의 추론 능력을 활용
샘플링 – 모델 컨텍스트 프로토콜(MCP)

---

7. MCP의 핵심 가치

컨텍스트 파편화 해결

- 각 AI 모델의 상이한 컨텍스트 처리 방식 통합
- 일관된 관리 체계 제공

M × N 통합 복잡도 제거

- M × N개의 개별 통합을 M + N 문제로 단순화
- 개발 및 유지보수 부담 감소

동적 컨텍스트 접근

- LLM이 외부 시스템 정보에 실시간으로 접근하여 Context 주입
- RAG와 유사하게 환각(Hallucination) 현상 완화

컨텍스트 윈도우 제약 완화

- 외부 시스템과의 효율적 정보 교환
- Context Window 길이 제한 문제 개선

상호운용성 확보

- 서로 다른 공급자의 도구들이 통일된 방식으로 동작
- 표준화된 컨텍스트 교환 방법 제공

공식 리포지토리:

- https://github.com/modelcontextprotocol

커뮤니티 리소스:

- https://github.com/punkpeye/awesome-mcp-clients/
- https://github.com/punkpeye/awesome-mcp-servers
- https://github.com/punkpeye/awesome-mcp-devtools

MCP 서버 코드 검증 도구:

- https://repomix.com/
- https://gitingest.com/

> 서드파티 MCP 서버 사용 시
>
> gitingest나 repomix로 코드를 검증한 후 사용하세요. 보안 위협을 사전에 차단할 수 있습니다.

다음 챕터에서는 이론을 바탕으로:

- ✅ Chapter 2의 Custom Component를 MCP Server로 전환
- ✅ `core_services` 기반 코드 재사용 (DRY 원칙)
- ✅ STDIO/SSE 두 가지 Transport 방식 실습
- ✅ Langflow MCP Tools로 연동 테스트

실습 예시: "Samsung SDS 종목 분석"

- → GDELT 검색 + News Content Extractor 자동 연동

- → MCP 표준 프로토콜로 모든 도구 재사용

✓ 축하합니다! 🎉

MCP의 이론적 배경과 핵심 개념을 모두 학습했습니다.
이제 실제로 MCP Server를 만들고 Langflow와 연동해볼 준비가 되었습니다!

참고자료:

- MCP 공식 문서: https://modelcontextprotocol.io
- Anthropic Claude MCP 가이드: https://docs.anthropic.com/mcp
- FastMCP 문서: https://github.com/jlowin/fastmcp
- awesome-mcp 리소스: https://github.com/punkpeye/awesome-mcp-servers