

Les 16 — MCP

Model Context Protocol — eigen server bouwen voor AI-clients

Vak:	AI-Assisted Development
Klas:	Klas A (3 uur fysiek)
Vervolg op:	Les 15 — Cursor + Vercel deploy
Volgende les:	Les 17 — Externe APIs in diepte

Leerdoelen

- MCP architectuur kennen (client/server/transport)
- Drie primitieven onderscheiden: tools, resources, prompts
- Eigen MCP server bouwen met TypeScript SDK
- Server laden in Cursor en Claude Desktop
- MCP Inspector gebruiken voor debug
- Verschil MCP vs Tool Calling weten
- Bestaande MCP servers kunnen installeren + gebruiken

Inhoud

1. Wat is MCP en waarom
2. Architectuur
3. De drie primitieven
4. Bestaande MCP servers
5. Eigen server bouwen
6. Laden in Cursor + Claude Desktop
7. Resources + Prompts
8. MCP Inspector
9. MCP vs Tool Calling
10. Productie + distributie
11. Bronnen

1

Wat is MCP en waarom

Model Context Protocol is een open standaard van Anthropic (november 2024) voor de communicatie tussen AI-assistenten en externe data/tools.

Probleem dat het oplost

Tools zoals 'zoek bands' wil je gebruiken in je eigen app, Cursor, Claude Desktop, of zelfs ChatGPT. Zonder MCP: vier integraties. Met MCP: één server.

USB-C voor AI

MCP is voor AI-tools wat USB-C is voor apparaten — één standaard connector.

2

Architectuur — client / server / transport

```
MCP Client    <-- JSON-RPC -->    MCP Server

(Cursor,      stdio /              (jouw code)
 Claude Desktop, HTTP/SSE          - tools
 custom apps)                      - resources
                                   - prompts
```

Drie rollen

- **MCP Client** — AI-assistent: Cursor, Claude Desktop, custom apps
- **MCP Server** — JOUW code, proces dat tools/data exposeert
- **Transport** — hoe ze communiceren (stdio default, HTTP/SSE remote)

3 De drie primitieven

Tools — uitvoerbare functies

```
server.tool(  
  "searchBands",           // naam  
  "Zoek bands op dag, stage of genre", // beschrijving  
  {                         // input schema  
    day: z.enum(["Vrijdag", "Zaterdag"]).optional(),  
    stage: z.string().optional(),  
  },  
  async ({ day, stage }) => {  
    const bands = await query(day, stage);  
    return { content: [{ type: "text", text: JSON.stringify(bands) }] };  
  }  
);
```

Resources — read-only data

```
server.resource("bands-list", "bands://list", async () => ({  
  contents: [{  
    uri: "bands://list",  
    mimeType: "application/json",  
    text: JSON.stringify(allBands),  
  }],  
}));
```

Prompts — herbruikbare templates

```
server.prompt("daily-recap", "Recap van een festivaldag",  
  { day: z.string() },  
  ({ day }) => ({  
    messages: [{ role: "user", content: {  
      type: "text", text: `Geef recap van ${day}...`  
    } }],  
  })  
);
```

4

Bestaande MCP servers

Officieel (Anthropic via npm)

- @modelcontextprotocol/server-filesystem — file system access
- @modelcontextprotocol/server-github — repos, issues, PRs
- @modelcontextprotocol/server-slack — Slack workspace
- @modelcontextprotocol/server-postgres — Postgres queries
- @modelcontextprotocol/server-puppeteer — browser automation

Community: linear, notion, asana, figma, stripe (500+ servers)

Voorbeeld Cursor config

```
// ~/.cursor/mcp.json
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": { "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_..." }
    }
  }
}
```

5 Eigen server bouwen

```
pnpm init
pnpm add @modelcontextprotocol/sdk zod
pnpm add -D typescript @types/node tsx
```

Minimal server (src/index.ts)

```
#!/usr/bin/env node
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import { z } from "zod";

const server = new McpServer({ name: "polderfest", version: "1.0.0" });

server.tool("searchBands", "Zoek bands",
  { day: z.string().optional() },
  async ({ day }) => {
    const bands = MOCK_DATA.filter(b => !day || b.day === day);
    return { content: [{ type: "text", text: JSON.stringify(bands) }] };
  }
);

const transport = new StdioServerTransport();
await server.connect(transport);
```

Build + run

```
pnpm tsc
node dist/index.js # wacht op JSON-RPC over stdio
```

Best practices

- Logging via `console.error` (stdout = protocol)
- Errors als data: `{ content: [...], isError: true }`
- Zod schemas zo strict mogelijk
- Descriptions zijn cruciaal — AI kiest op basis hiervan

6

Laden in Cursor + Claude Desktop

Cursor (~/.cursor/mcp.json)

```
{
  "mcpServers": {
    "polderfest": {
      "command": "node",
      "args": ["/Users/jouw-naam/dev/mcp-polderfest/dist/index.js"],
      "env": { "SUPABASE_URL": "...", "SUPABASE_KEY": "..." }
    }
  }
}
```

Belangrijk

Gebruik **absolute paths** (geen ~ of relative). Restart Cursor na config change.

Claude Desktop

~/Library/Application Support/Claude/claude_desktop_config.json (macOS) — identieke JSON structuur, ander pad. Op Windows: %APPDATA%\Claude\.

7

Resources + Prompts in praktijk

Resources gebruik-cases

- **Documentatie** — handleidingen, RFCs, design docs
- **Database snapshots** — read-only views
- **API responses** — gecacheerde externe data
- **Logs** — recent errors voor debugging

In Cursor: `@polderfest:resource-name`

Prompts gebruik-cases

- Code review template
- Bug report opmaak
- Vergader-recap structuur

Cursor command palette — 'MCP: prompt-name' — AI start met die exacte prompt.

8

MCP Inspector — debug tool

```
npx @modelcontextprotocol/inspector node dist/index.js
```

Opent browser-UI op localhost:5173:

- Linker panel: tools/resources/prompts lijst
- Hoofdvenster: parameters invullen + call uitvoeren
- Result + raw JSON-RPC zichtbaar

Workflow: schrijf tool → build → test in Inspector → werkt → restart Cursor. Niet werkt? Inspector toont errors direct.

9 MCP vs Tool Calling

Aspect	Tool Calling (Les 12)	MCP (Les 16)
Waar leeft tool	In jouw Next.js app	Aparte server
Clients	Alleen jouw chat	Alle MCP-clients
Distributie	Niet	npm publish
State	Per-request	Server proces
Best voor	App-features	Herbruikbare tools

Mentale model

Tool calling = function in mijn app.

MCP server = library die elke AI-client kan gebruiken.

10 Productie + distributie

npm publish

```
// package.json
{ "name": "@org/mcp-polderfest",
  "version": "1.0.0",
  "bin": { "mcp-polderfest": "./dist/index.js" },
  "files": ["dist"] }
```

```
pnpm publish --access public
```

Anderen installen via npx:

```
"command": "npx",
"args": ["-y", "@org/mcp-polderfest"]
```

Security

- Auth-token in env-var voor gevoelige data
- Zod schemas strict voor input validation
- Audit log voor elke tool-call
- Geen secrets in code — env vars only

11 Bronnen

-
- MCP — modelcontextprotocol.io
 - TS SDK — github.com/modelcontextprotocol/typescript-sdk
 - Inspector — github.com/modelcontextprotocol/inspector
 - Cursor MCP — docs.cursor.com/context/model-context-protocol
 - Claude Desktop MCP — docs.anthropic.com/en/docs/build-with-claude/mcp
 - Anthropic launch — anthropic.com/news/model-context-protocol
 - Awesome MCP — github.com/punkpeye/awesome-mcp-servers