A RESOURCE FOR DESIGN SYSTEM AND ENGINEERING TEAMS

MCP integration patterns.

Single versus composed, and when to skip MCP.

Resources for context. Tools for decisions. Code for volume.

25
rules across the five patterns

05
patterns, from single server to no MCP

02
protocol revisions one server factory answers

Pick the pattern before you pick the server.

Most MCP trouble is a pattern problem: reference data behind a tool that dumps a whole file, three systems behind one credential, fifteen servers loaded for every task, a gateway relaying tool text it never checked. The protocol's primitives have different owners: tools are model-controlled, resources application-driven, prompts user-controlled [S01, S02, S12]. How you combine and split them decides cost, reliability and blast radius.

Five patterns follow: resources versus tools, one server per system, composed and gateway servers, code execution with MCP, and when MCP is the wrong answer. The essay says keep configs small and add servers per task; the patterns show why, and where composition earns its cost.

The kit is a working server on the official TypeScript SDK v2: `acme-ds-tokens` exposes the Field Guide 03 tokens as four resources and two read-only tools. Its smoke test negotiates 2026-07-28, the Inspector connects on 2025-11-25, and Field Guide 24's scanner flags nothing.

Version 1.0 / Sources checked 24 September 2026
Field Guide 27 of the Design × AI series. Pairs with Field Guides 24 (evaluation), 25 (starter config) and 26 (security review), which scan, configure and review this guide's server. Verified 24 September 2026: MCP specification 2026-07-28, @modelcontextprotocol/server and /client 2.1.0, zod 4.6.5, MCP Inspector 2.8.0, Node 22.

Practical guidance, not a specification. The token-saving figure for code execution is Anthropic's own example, quoted, not measured here. The SDK v2 API is recent; recheck the imports against its changelog before copying. Prepared with AI assistance and edited by hand.

START HERE

Find your pattern, then read its rules.

Most teams need the first two patterns and one of the last three. Start from the problem, not from the server catalogue.

If you need to	Pattern	Section
Give agents your design system's facts	Resources for reference data, tools for precise lookups	01
Connect one system of record	One server per system and per trust boundary	02
Reach many systems through one endpoint	Composed in the client first; a gateway only with proxy rules	03
Chain many calls or filter large results	Code execution with MCP, in a sandbox	04
Use a CLI, static guidance or one-off data	Probably not MCP	05

Building your first server

Section 01, then copy the kit. `npm install && npm run check` runs the smoke test on both protocol revisions and the Inspector.

Consolidating servers

Section 02 before section 03. Merging systems into one server merges their credentials and their blast radius.

Hitting context limits

Section 03's progressive discovery, then section 04. Count tool-definition tokens first.

Asked to wrap everything in MCP

Section 05. A wrapper around a CLI the agent can already run is a second thing to secure.

Label	Meaning
`PATTERN`	A structural choice: how many servers, which primitive, where code runs.
`SPEC`	Traces to the MCP specification 2026-07-28 or its official docs.
`KIT`	Demonstrated in the kit's server, with the command that shows it.
`CLIENT` / `RESEARCH` / `PRACTICE`	Client behaviour per its docs / published research / a working habit.

PATTERN 01

Resources for context, tools for decisions.

Resources are data the host or the person attaches. Tools are functions the model chooses to call. Put each thing where its owner can control it.

Suggested owners: Design-system lead + engineering lead

- [ ] **Match the primitive to who is in control**

  01.01 PATTERN SPEC

  Tools are model-controlled: the model discovers and calls them. Resources are application-driven: the host decides what to attach. Prompts are user-controlled templates. A design-token file is context; "what is the focus ring colour in dark mode" is a decision.

  Evidence: Each capability listed with its primitive and its controller.

  S01, S02, S12.

- [ ] **Serve reference data as resources with stable URIs**

  01.02 PATTERN KIT

  The kit serves `tokens://semantic/index` (Markdown), the light and dark DTCG files and the Button component tokens. Hosts can attach them once; nothing is re-fetched per question.

  Evidence: `npm run inspect:resources` lists 4 resources.

  S02. Kit.

- [ ] **Answer precise questions with a lookup, not a dump**

  01.03 PATTERN KIT

  `lookup_token` returns one token: CSS variable, resolved value in a mode, alias and usage note. A tool that returns the whole file makes the model search it in context, every time.

  Evidence: Smoke test: `color.border.focus` in dark mode returns `#80adff` via `core.color.blue.300`.

  Kit.

- [ ] **Annotate every tool, honestly**

  01.04 SPEC KIT

  Both kit tools declare read-only, non-destructive, idempotent and closed-world. Leave nothing to the defaults: a missing hint means destructive and open-world.

  Evidence: Field Guide 24's `inspect-tools.mjs`: 0 findings on the kit's tool list.

  S01. Field Guide 24.

- [ ] **Return structured content, and text for older clients**

  01.05 SPEC KIT

  Declare an `outputSchema`, return `structuredContent`, and also return the JSON as text, as the spec asks for backwards compatibility. Prefer optional fields to nullable ones: the Inspector's portability check flagged four `["string","null"]` types in the kit's first draft.

  Evidence: `mcp-inspector --cli ... --method tools/list --strict` reports no warnings.

  S01, S09. Working record in Appendix B.

- [ ] **Fail as a tool result with a next step**

  01.06 SPEC KIT

  An unknown name is a tool execution error (`isError: true`) that lists the three closest names; a core token is refused with the semantic tokens that use it. The model can recover without a human.

  Evidence: Smoke test: `color.text.primary` and `core.color.blue.600` both return actionable errors.

  Tools, error handling. S01.

- [ ] **Bound every result**

  01.07 KIT CLIENT

  `search_tokens` returns at most 10 matches with a `truncated` flag and tells the model to refine. Claude Code warns above 10,000 tokens per result and cuts at 25,000 by default.

  Evidence: Output schema includes `total` and `truncated`.

  S10. Kit.

Attach what is always true. Look up what the model needs to decide.

PATTERN 02

One server per system, per trust boundary.

The default. Each server wraps one system of record with one credential, so each can be evaluated, scoped and removed on its own.

Suggested owners: Engineering lead

- [ ] **Split by system and by credential**

  02.01 PATTERN

  The token server holds no credential and reads local files; GitHub needs a token and reads other people's content. In one server they would share a blast radius and a review. Apart, one is LOW risk and one is MEDIUM.

  Evidence: Field Guide 26's inventory rates them separately.

  Field Guide 26.

- [ ] **Serve the public API, not the internals**

  02.02 PATTERN KIT

  The kit lists and accepts semantic tokens only. `core.*` and `comp.*` names are refused with a pointer to the semantic tokens that use them, so an agent cannot reach past Field Guide 03's public layer through MCP.

  Evidence: Smoke test: `core.color.blue.600` is refused, naming `color.bg.brand`, `color.border.focus` and `color.text.link`.

  Field Guide 03, rule R03. Kit.

- [ ] **Name tools so they survive aggregation**

  02.03 SPEC

  Tool names are unique per server only. Clients and proxies that combine servers should disambiguate, for example by prefixing the server name; pick distinctive names so the prefix is a safety net, not the only difference.

  Evidence: No tool name in your server collides with another server in the same config.

  Tools, tool names. S01.

- [ ] **Stateless by default; explicit handles when not**

  02.04 SPEC

  The 2026-07-28 revision removed protocol sessions and the initialize handshake. A server that needs state across calls returns a handle as an ordinary result and takes it back as an argument: random, bound to the user, expiring.

  Evidence: No reliance on per-connection state; handles documented where used.

  S03. State handle hijacking. S06.

Answer both protocol eras during the transition

02.05 KIT SPEC

Clients will speak 2025-11-25 and 2026-07-28 side by side for a while. SDK v2's `serveStdio` takes one server factory and answers whichever era the client opens with.

Evidence: `npm run smoke` negotiates 2026-07-28; `npm run smoke:legacy` and the Inspector use 2025-11-25.

S03, S07. Kit.

Small servers are easy to trust, easy to scope and easy to remove.

PATTERN 03

Composed servers and gateways.

Composition happens in one of two places: in the client, which loads several servers side by side, or in a gateway server that fronts others. The second is a proxy, with a proxy's duties.

Suggested owners: Engineering lead + security partner

- [ ] **Compose in the client first**

  03.01 PATTERN CLIENT

  Loading three small servers in one client is composition, and each keeps its own process, credential and approval. Field Guide 25's configs do exactly this. Check the combined set for the lethal trifecta: composition is where it forms.

  Evidence: Field Guide 26's `check-inventory.mjs` run over the client's servers.

  S11. Field Guide 26.

- [ ] **A gateway follows the proxy rules**

  03.02 PATTERN SPEC

  A server that fronts other APIs or servers must keep per-client consent when it uses a static OAuth client ID, and must never pass a client's token through to the systems behind it.

  Evidence: Consent per client and audience-checked tokens, shown in the gateway's design.

  Confused deputy and token passthrough. S06.

- [ ] **Prefix what you relay**

  03.03 SPEC

  Two upstream servers can both expose `search`. A gateway should disambiguate names, for example `github.search` and `tokens.search`; the server's own `name` field is not guaranteed unique and should not be relied on.

  Evidence: Relayed tool names carry a server prefix.

  Tools, tool names. S01.

- [ ] **A gateway must not launder rug pulls**

  03.04 RESEARCH

  If a gateway relays upstream tool descriptions, an upstream change reaches every client at once. Lock each upstream tools/list and diff it before relaying a change.

  Evidence: Field Guide 26's `diff-tools.mjs --verify` runs on each upstream.

  S13. Field Guide 26.

Switch to progressive discovery past a threshold

03.05 PATTERN SPEC CLIENT

When tool definitions take a significant share of the context, load them on demand through a search step. The MCP client guidance suggests a threshold of 1% to 5% of the context window; Claude Code's tool search is on by default.

Evidence: Tool-definition token count measured against the threshold.

Client best practices. S04. S10.

A gateway is a server that other servers' risks pass through. Review it that way.

PATTERN 04

Code execution with MCP.

When a task chains many calls or filters large results, let the model write code against the tools and run it in a sandbox, so intermediate data never passes through the context.

Suggested owners: Engineering lead + platform owner

- [ ] **Use code for volume, calls for single answers**

  04.01 PATTERN RESEARCH

  The host turns tool schemas into a typed API; the model writes one script; only the result returns to the model. Anthropic's worked example went from 150,000 tokens to 2,000, a 98.7% saving. That is their example, not a general rate.

  Evidence: Your own before-and-after token count on one real task.

  S05, S04.

- [ ] **Output schemas make the generated API typed**

  04.02 SPEC KIT

  With an `outputSchema`, the generated function returns a real type, for example `lookup_token(...): { token: { cssVar: string, value: string } }`. Without one, the code gets a string to parse.

  Evidence: Every tool meant for code execution declares an outputSchema.

  Client best practices, programmatic tool calling. S04.

- [ ] **The sandbox is the price of admission**

  04.03 PATTERN SPEC

  Running model-written code needs isolation, resource limits and monitoring. Keep network access denied inside the sandbox and route every tool call through the host, so the only exits are the tools you reviewed.

  Evidence: Sandbox runtime, limits and network policy written down.

  S04, S05.

- [ ] **Keep direct calls for single lookups**

  04.04 PATTERN KIT

  One token lookup is one small call with a bounded answer. Wrapping it in code adds a sandbox round trip and saves nothing.

  Evidence: Code execution is reserved for tasks with several calls or large intermediate data.

  Kit: `lookup_token` answers in one call.

Move the loop into code, and keep the sandbox between that code and everything else.

PATTERN 05

When not to use MCP.

MCP earns its cost when a model needs live, structured access to a system it cannot otherwise reach. Outside that, it is one more thing to review.

Suggested owners: Design-system lead + engineering lead

- [ ] **Not for a CLI the agent can already run**

  05.01 PRACTICE CLIENT

  In Claude Code, `git` and `gh` run through the built-in shell with its permission prompts. A server wrapping the same CLI adds a process, a credential and a tool list without adding a capability.

  Evidence: The task cannot be done with the client's built-in tools.

  Field Guide 24, check A.02.

- [ ] **Not for static guidance**

  05.02 PRACTICE

  Rules that never change per request belong in an instruction file or the component docs agents already read (Field Guide 04). A server that only returns the same text adds latency and a trust decision.

  Evidence: The server returns something that changes, or answers a question.

  Field Guide 04.

- [ ] **Not for one-off data**

  05.03 PRACTICE

  A spreadsheet you need once is an attachment. Build a server when the question repeats every week.

  Evidence: Recurring use named in the evaluation record (Field Guide 24, check A.01).

  The essay's rule: add servers for a specific problem.

- [ ] **Not where you cannot break the trifecta**

  05.04 RESEARCH

  If connecting a system would give one session private data, untrusted content and a way out, and nothing can break the chain, do not connect it. No server is worth an unbounded leak path.

  Evidence: Field Guide 26's inventory check for the client stays free of unbroken trifectas.

  S11. Field Guide 26.

The best integration is sometimes the one you did not build.

APPENDIX A

The token server, in full view.

About a hundred lines of SDK code in `src/server.mjs`, plus pure token logic in `src/tokens.mjs` that tests without installing anything. The tokens come from Field Guide 03.

ds-tokens-mcp/src/server.mjs (excerpt)

```
import { McpServer } from "@modelcontextprotocol/server";
import { serveStdio } from "@modelcontextprotocol/server/stdio";
import * as z from "zod/v4";

server.registerTool("lookup_token", {
  title: "Look up a design token",
  description: "Return one semantic design token ... Only semantic names are accepted.",
  annotations: { readOnlyHint: true, destructiveHint: false,
                 idempotentHint: true, openWorldHint: false },
  inputSchema: z.strictObject({
    name: z.string().min(3).max(80),
    mode: z.enum(["light", "dark"]).default("light"),
  }),
  outputSchema: z.object({ token: TokenShape }),
}, async ({ name, mode }) => {
  const result = lookupToken(tokens, name, mode);
  if (result.error) return { isError: true, content: [{ type: "text", text: result.error }] };
  return { content: json(result), structuredContent: result };
});

server.registerResource("semantic-index", "tokens://semantic/index",
  { title: "Semantic token index", mimeType: "text/markdown", cacheHint },
  (uri) => text(uri, "text/markdown", indexMarkdown(tokens)));

// One factory answers 2026-07-28 and 2025-era clients.
serveStdio(() => createServer(tokens));
```

Condensed from the kit file. The kit keeps tool titles, descriptions and annotations in `src/definitions.mjs`, so a test can compare them with the captured tools/list.

Primitive	Name	Controlled by	Why this primitive
Resource	`tokens://semantic/index`	Host or person	Always true; attach once
Resource	`tokens://semantic/light`, `/dark`	Host or person	Source files for review or diffing
Resource	`tokens://component/button`	Host or person	Scoped to the Button's own code
Tool	`lookup_token`	Model	A decision per value written
Tool	`search_tokens`	Model	Finding a token by intent, bounded to 10

APPENDIX B

Install it, run it, read the output.

Verified by copying the kit to an empty directory, installing from npm and running every script. The output below is from that run.

terminal

```
$ npm install && npm run smoke
protocol 2026-07-28, server acme-ds-tokens 1.0.0
resources: tokens://semantic/index, tokens://semantic/light, tokens://semantic/dark,
  tokens://component/button
tools: lookup_token, search_tokens
lookup_token color.border.focus (dark): #80adff via core.color.blue.300,
  var(--ds-color-border-focus)
lookup_token color.link: deprecated, "Use color.text.link. Removed in 3.0.0."
lookup_token core.color.blue.600: isError: core.color.blue.600 is not a public token.
  Application code uses semantic tokens only; semantic tokens that use it:
  color.bg.brand, color.border.focus, color.text.link.
lookup_token color.text.primary: isError: No semantic token color.text.primary.
  Closest: color.text.link, color.text.default, color.text.muted.
search_tokens "focus": 2 matches: color.border.focus, stroke.focus
smoke: OK

$ npx mcp-inspector --cli node src/server.mjs --method initialize --format json
{"result":{"serverInfo":{"name":"acme-ds-tokens","version":"1.0.0"},
  "protocolVersion":"2025-11-25","capabilities":{"resources":{"listChanged":true},
  "tools":{"listChanged":true}}, ...}}
```

Long lines wrapped for print. `npm run check` runs the smoke test on both eras and the Inspector tools/list, and exits 0.

> **Working record: the first draft was portable only in theory.**
>
> The first Inspector run reported four schema-portability warnings, and `--strict` showed why: zod's `.nullable()` produced `["string","null"]` type arrays, which clients that map schemas onto single-type dialects may reject. Switching those fields to optional and omitting absent values cleared all four. The inputs also became `z.strictObject`, so unknown arguments fail.
>
> Captured with MCP Inspector 2.8.0 on 24 September 2026.

package.json (dependencies)

```
"dependencies": { "@modelcontextprotocol/server": "2.1.0", "zod": "^4.6.5" },
"devDependencies": { "@modelcontextprotocol/client": "2.1.0",
                     "@modelcontextprotocol/inspector": "2.8.0" }
```

The SDK is pinned exactly; a server is a dependency of every client that runs it. v2 replaces the single `@modelcontextprotocol/sdk` package, which stays on the 1.x line.

KEEP WITH THE DESIGN

Record the integration decision.

One record per integration. It explains to the next team why this system is one server, part of a gateway, behind code execution or not on MCP at all.

System / owner	**Pattern chosen (01 to 05)**
Resources exposed (URIs)	**Tools exposed (names, annotations)**
Credential and trust boundary	**Clients and neighbouring servers**
Output bounds / largest result	**Protocol revisions supported**
Evaluation and review records (FG 24, 26)	**Revisit when**

> **Pattern 05 is a valid answer.**
> If the record ends with "not MCP", write down what the team uses instead, so the question does not come back next quarter without the answer.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. Kit output comes from a clean install from the public npm registry on Node 22.

S01 / MCP specification 2026-07-28, Tools
Tools, annotations, names, structured content, errors.
https://modelcontextprotocol.io/specification/2026-07-28/server/tools

S02 / MCP specification 2026-07-28, Resources
Application-driven resources identified by URI.
https://modelcontextprotocol.io/specification/2026-07-28/server/resources

S03 / MCP specification 2026-07-28, Key changes
Stateless protocol, removed sessions, explicit handles.
https://modelcontextprotocol.io/specification/2026-07-28/changelog

S04 / MCP docs, Client best practices
Progressive discovery at 1% to 5% of context; code mode in a sandbox.
https://modelcontextprotocol.io/docs/2026-07-28/develop/clients/client-best-practices

S05 / Anthropic, Code execution with MCP
150,000 to 2,000 tokens in their example; sandbox caveats (4 November 2025).
https://www.anthropic.com/engineering/code-execution-with-mcp

S06 / MCP, Security Best Practices
Confused deputy, token passthrough, state handle hijacking.
https://modelcontextprotocol.io/docs/2026-07-28/tutorials/security/security_best_practices

S07 / MCP TypeScript SDK
v2 packages /server and /client, the stable line for 2026-07-28.
https://github.com/modelcontextprotocol/typescript-sdk

S08 / npm, @modelcontextprotocol/server
Version 2.1.0, published 23 September 2026 (npm view).
https://www.npmjs.com/package/@modelcontextprotocol/server

S09 / MCP Inspector, CLI client
Scriptable tools/list, resources/list, tools/call, initialize.
https://modelcontextprotocol.io/docs/2026-07-28/tools/inspector/cli

S10 / Claude Code, Connect Claude Code to tools via MCP
Tool search on by default; output warning at 10,000 tokens and 25,000 cap.
https://code.claude.com/docs/en/mcp

S11 / Simon Willison, The lethal trifecta
Combining tools creates leak paths (16 June 2025).
https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/

S12 / MCP docs, Understanding MCP servers
Who controls tools, resources and prompts.
https://modelcontextprotocol.io/docs/2026-07-28/learn/server-concepts

S13 / Invariant Labs, MCP tool poisoning attacks
Rug pulls after approval (1 April 2025).
https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

Maintenance: rerun `npm install && npm run check` in a clean directory at each SDK minor release and specification revision. Update the guide and the server together.