

Agentics Day:
MCP + Agents
NORTH AMERICA

Protocol Solved, Design Still Open: Domain-Driven Design for MCP Servers

Fabrizio Lazzaretti

with Annegret Junker

Who has generated an MCP server from an OpenAPI file?

Hands up.

And who has used the generated MCP server?

Hands up.

“Buy me a friendly dog”



Claude Code, Haiku 4.5, only the wrapped Petstore; a typical take

User: Buy me a friendly dog
MCP calls: `addPet` → `placeOrder`
Agent: “I’ve purchased a friendly dog for you named Buddy”

User: Which friendly dogs are available?
MCP calls: `findPetsByTags`
Agent: Rex, Bello and Buddy, “the one I just added for you”

User: Did the order go through?
MCP calls: none
Agent: “approved, but not complete”: Buddy still for sale

“Buy me a friendly dog”

- No error. The agent said: *“I’ve purchased a friendly dog for you named Buddy.”*
  Buddy didn’t exist. The agent made him up with `addPet`.
- Rex and Bello were for sale all along.
  To the agent, `addPet` and `placeOrder` are just endpoints. None of them means “buy”.

What buying means lived in the customer’s head.
What does the agent need?

Today: a better server. **75 % fewer tokens** for the tools, and Rex really sold.

Fabrizio Lazzaretti

Agentics Day: MCP + Agents
NORTH AMERICA



Senior Managing Consultant at Wavestone

CNCF Ambassador · CloudEvents SDK maintainer

Event-driven architecture, API design, AI enthusiast

Co-author: *Crafting Great APIs with Domain-Driven Design*

Annegret Junker



Chief Software Architect at codecentric

Interested in microservices, DDD, API design, event design and how to get faster from specification to code

01

The MCP explosion has a quality problem

The Petstore as MCP in a few lines


```
spec = httpx2.get(f"{BASE}/openapi.json").json()
mcp  = FastMCP.from_openapi(openapi_spec=spec,
                            client=httpx2.AsyncClient(base_url=BASE))
mcp.run()
```

The agent now sees **19 tools**, in endpoint words:

```
updatePet  addPet  findPetsByStatus  findPetsByTags  getPetById
updatePetWithForm  deletePet  uploadFile  getInventory  placeOrder
getOrderById  deleteOrder  createUser  createUsersWithListInput
loginUser  logoutUser  getUserByName  updateUser  deleteUser
```

FastMCP's own docs: *"LLMs achieve significantly better performance with well-designed and curated MCP servers than with auto-converted OpenAPI servers."*

Fine for a start, a first version. Not the design.

Source: Swagger (2026) (Swagger Petstore 1.0.29, pinned 1 Nov 2026); FastMCP (2026)

Whatever the API has, the agent gets

Developer words

"try IDs ≤ 5 or > 10"

`createUsersWithListInput`

Admin tools for buyers

`addPet`
`updatePet`
`deletePet`

No rules

A sold pet can be ordered.

An order keeps the pet on offer.

Not a fluke: 5 runs each on the wrapper.

Haiku 4.5: invented a dog with `addPet` in 5 of 5, overwrote Rex in 2, reported success in 4.

Opus 5.5: ordered the right dog in 5 of 5. The dog stayed for sale in 5 of 5: no endpoint says "buy".

Too much flexibility in the API is the point.

Source: Live Petstore tested Nov 1, 2026; evaluation Nov 1, 2026 (before side, no pickup, 5 runs each with Haiku 4.5 and Opus 5.5)

REST was built for granular work

- REST: small steps, glued together by a developer's code
- As MCP: the agent is the glue, and it stops halfway
- Buy = `placeOrder` + `updatePet`. The agent did one.

Agents need less granularity:
one intention, one call, the rules inside.

REST over CRUD, one layer up. The API says *order*; the domain says *purchase*.

OpenAPI's answer: **Arazzo** workflows. Today: the MCP server.

Source: Evaluation Nov 1, 2026, 5 tests each with Opus 5.5 and Haiku 5.5: `placeOrder`, never `updatePet`; OpenAPI Initiative (2026)

Three shapes of failure in the wild

Wrapper

Every endpoint a tool, generated from OpenAPI. The Petstore you just saw

Mega-server

GitHub's server in 2025: 102 tools, 64.6k tokens, before the user asks anything

Raw model

Supabase: `execute_sql` reaches every table. Same tool, same rights

The agent becomes the anti-corruption layer.

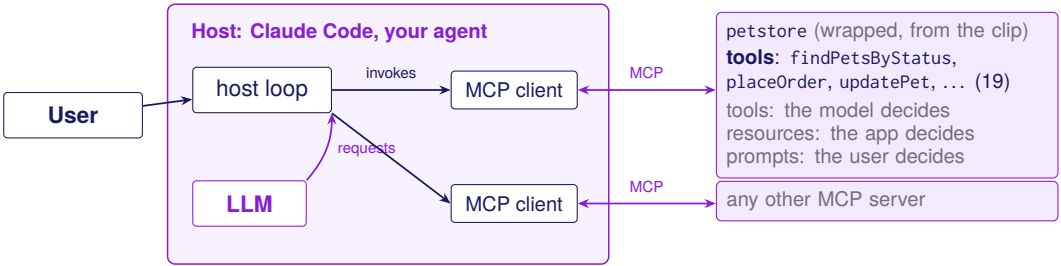
Nobody calls it DDD.

Source: GitHub (2025); Supabase: Supabase (2026); Willison (2025); Havaei et al. (2025) (2025 demo: `service_role` bypasses row-level security)

02

Protocol solved, contract open

Agent requests, host invokes



One protocol for AI to discover and use tools, at runtime, in any host.

The model only *requests*. The host *invokes*, and can ask the user first.

Written once, a tool reaches every agent.

Source: MCP: Agentic AI Foundation (Linux Foundation). Model Context Protocol (2026a)

What reaches the model is the contract

- tool name
- description
- input schema
- server instructions
- the result text
- the error text

Dropped by Claude Code: output schemas, annotations, titles.
No docs, no colleague to ask.

Rex is already sold. The agent buys him again:

Wrapped API: `placeOrder(petId: 1)`
→ `200 OK {"id": 0, "petId": 1, "status": "placed"}`
Success, and wrong.

Pet Sales: `buy_pet(pet_id: 1)`
→ `PetNotAvailable: Rex is already sold. Use find_available_pets for others.`

The error message is the contract.

Domain language is a reliability concern, not style.

Source: Tool fields: Model Context Protocol (2026b); the spec doesn't say what reaches the model, the host decides. Measured Nov 1, 2026: the first model request of Claude Code 2.1.316

Protocol solved.
Design still open.

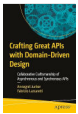
The advice is clear: tools for workflows, not endpoints.
DDD is the method to find them.

Source: Workflows, not endpoints: Mohammed and Chau (2025); Lowin (2025)

03

DDD mapped onto MCP: finding the cut

A method in six steps



BEFORE THIS TALK

1. Business Model Canvas: capabilities, the core
2. Map capabilities: Wardley Mapping

3 Tell the story

Domain Storytelling: "Buying a pet"

4 Find the events and the cut

EventStorming: the board, pivotal events

5 One context, one MCP server

Context Mapping, then the Bounded Context Canvas: Pet Sales

6 Design the MCP API in business words

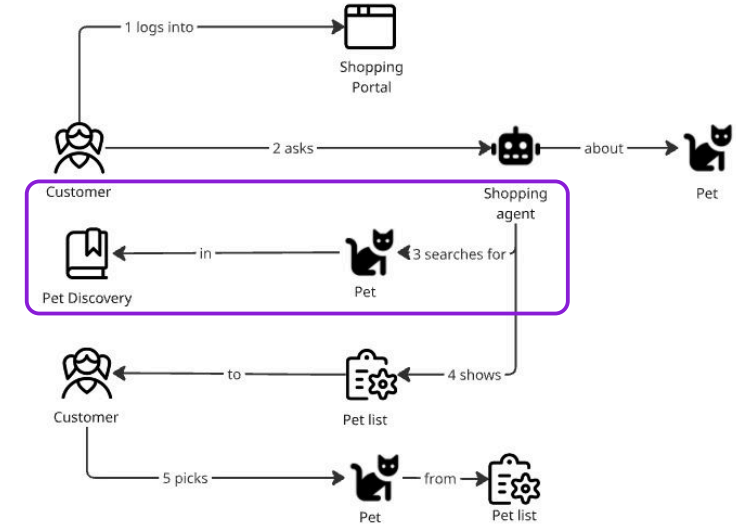
MCP Product Canvas, from our book's API Product Canvas: tool, code API or skill? Names, errors

+ Visual Glossary: define the ubiquitous language. Not today.

strategic design (1–5) | **tactical design (6)**

Source: Business model and capabilities, API Product Canvas: Junker and Lazzaretti (2025); Wardley Mapping: Wardley (2015); extended for MCP: Junker and Lazzaretti (2026); DDD Crew (2026a); DDD Crew (2026b)

Step 3: tell the story

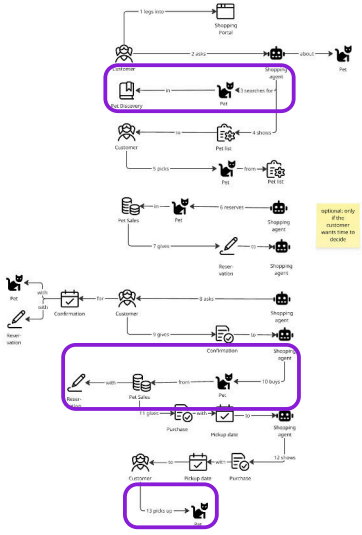


Domain Storytelling: one concrete case, told by the people who do it, in their words.

3 The Shopping agent searches Pet Discovery for available pets

→ `find_available_pets`

Step 3: tell the story



Domain Storytelling: one concrete case, told by the people who do it, in their words.

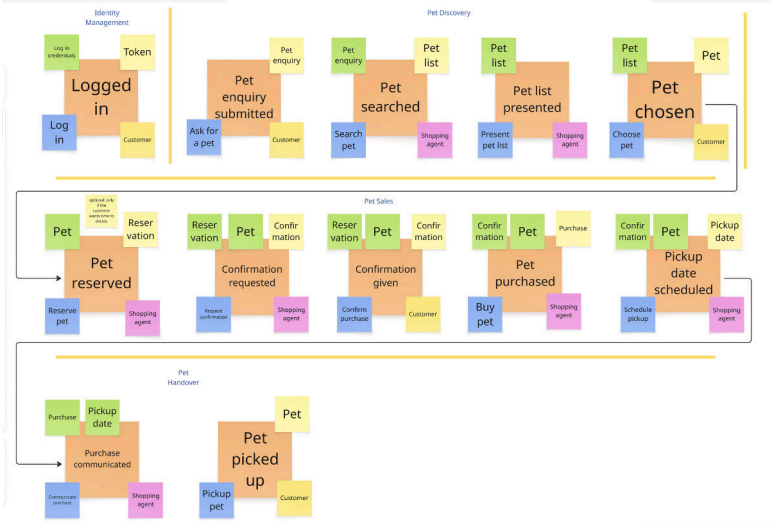
3 The Shopping agent searches Pet Discovery for available pets
→ `find_available_pets`

10 The Shopping agent buys the pet from Pet Sales
→ `buy_pet`

13 The Customer picks up the pet
→ in the shop: no tool

The method found a capability the API never had: the pickup.

Step 4: find the events and the cut



Event

Command

Customer (person)

Shopping agent, system

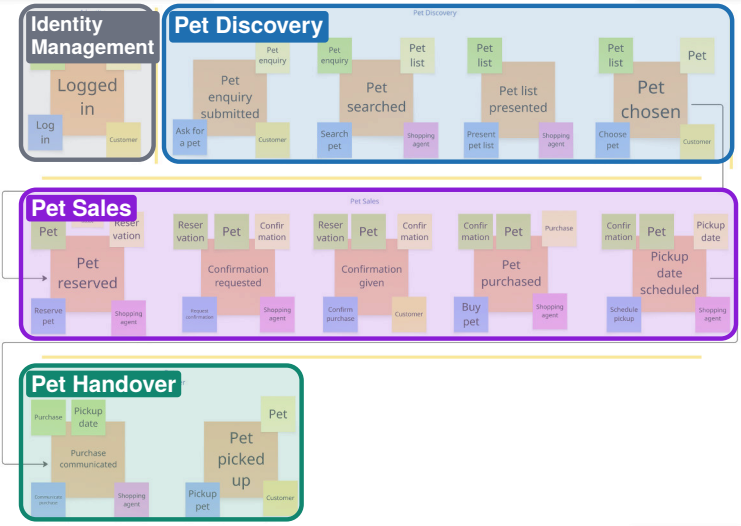
Input data

Output

Four contexts: Identity Management, Pet Discovery, Pet Sales, Pet Handover

Source: Board and EventStorming skills: Junker (2026). Failure events (*Purchase refused: pet not available*, *Reservation expired*) not shown, for space

Step 4: the cut, four contexts



Four contexts.

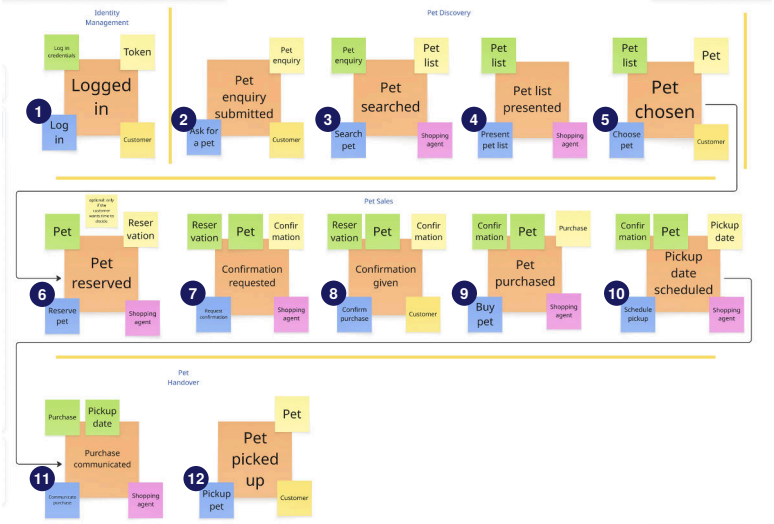
A context is cut where the words change meaning.

Pet is an offer in **Discovery**, an item for sale in **Sales**, an animal to hand over in **Handover**.

How the contexts relate (upstream, downstream, published language): **Context Mapping**. Not shown today.

Source: Board and EventStorming skills: Junker (2026)

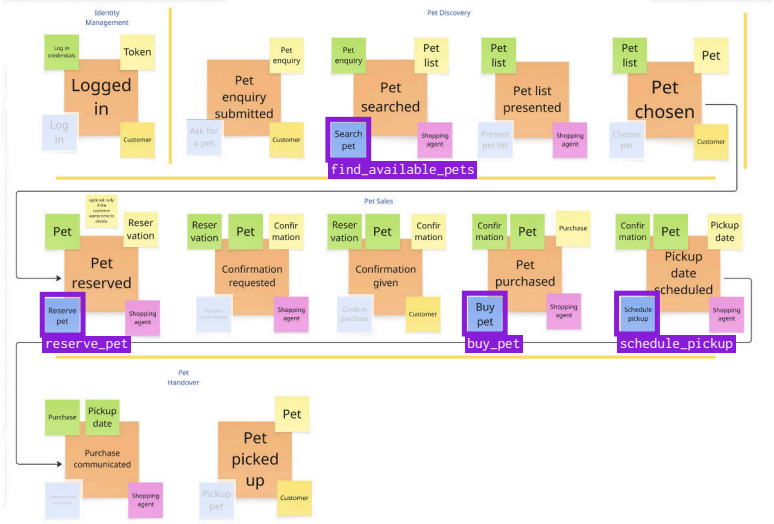
Step 4: board command → MCP surface



12 commands on the board.

How many become tools?

Step 4: board command → MCP surface



12 commands on the board.

4 tools.

A command becomes a tool when the agent issues it and only the context (the backend) can decide it.

The rest: chat, the host, OAuth, the shop.

+ `cancel_purchase` (story variation)
+ `my_purchases` (a read)

Step 5: one context, one MCP server

Identity Management	Pet Discovery	Pet Sales	Pet Handover
no server	`petstore-discovery`	`petstore-sales`	`petstore-handover`
Log in: MCP authorization (OAuth)	audience: buyer	audience: buyer	audience: buyer
	Search pet	Reserve pet Buy pet Schedule pickup Cancel purchase + the terms of sale (read)	Communicate purchase Pick up pet: in the shop

- One MCP server per context and audience: the buyer never sees the shop's admin commands
- Not a microservice: no separate deployment needed. The demo runs all three in one app
- One OAuth audience per server: a sales token doesn't open Pet Discovery

The bounded context is the security boundary.

Pet Sales on the Bounded Context Canvas

NAME
Pet Sales

PURPOSE
Turns a chosen pet into a purchase the customer can pick up, and makes sure no pet is sold twice.

INBOUND

Shopping agent
Reserve pet
Buy pet
Schedule pickup
Cancel purchase

Customer
Confirm purchase

Shop staff
Stock (query)

UBIQUITOUS LANGUAGE

Pet: available · reserved · sold
Reservation: optional, expires after 24 h
Purchase: never “order”
Pickup date: a slot the shop offers
Customer: borrowed from Identity Management

BUSINESS DECISIONS

No pet is sold twice
A reservation expires after 24 h
Pickup only in a free slot the shop offers
Cancel only until the pickup slot

OUTBOUND

Pet Discovery
Pet taken off offer
Pet back on offer

Pet Handover
Pet purchased
Pickup date scheduled
Purchase cancelled

Source: Bounded Context Canvas v5: DDD Crew (2026b). Purple: inbound commands that become tools

04

Tool, code API or skill?

MCP for business commands, code for loops

When the agent needs …

one business command

→ an MCP tool

a small result, the rules inside the server

e.g. `buy_pet`

When the agent needs …

a loop over many items

→ code in a sandbox

the agent writes a script against the tools: **98.7 % fewer tokens** in one example (~150k → ~2k)

e.g. “unsold pets over 30 days, per species”

on Kubernetes: e.g. Agent Sandbox (kubernetes-sigs)

A good domain tool is a script the server already wrote.

`buy_pet = placeOrder + updatePet` + the rules, written once

Source: Anthropic (2025a); MongoDB example (51 calls vs. one script, an illustration): Kanjariya (2026); Kubernetes SIG Apps (2026)

How to pick: consumer and task

If ...	then
one domain command the model must choose	**MCP tool**
a loop, filter or join over many items	**code API** in a sandbox
a recurring workflow with know-how	**skill**, which calls tools
a coding agent, and a well-known CLI exists	**CLI**
a whole task for another team's agent	**A2A**
something happened in the domain	**events**

A2A: if the shop ran its own pickup agent, the customer's agent would hand over the whole task: "schedule Rex's pickup, Saturday morning". MCP is vertical (agent → tools), A2A horizontal (agent → agent).

Source: Data the model must *analyze* goes into context; data it must *manipulate* stays in code. Agent Skills (2025); A2A Project (2026)

The Petstore, classified

Capability	Who	Surface
reserve, buy, schedule pickup, cancel	Shopping agent	**MCP tool**: rules in the server
find available pets, my purchases	Shopping agent	**MCP tool**: small reads
the shop's rules	the host attaches	**MCP resource**
"Choose a pet for my household"	Shopping agent	**skill**: asks, filters, explains
"Unsold pets over 30 days, per species"	shop owner's agent	**code API** in a sandbox
stock counts	dashboard	**REST**
confirm purchase	customer	**host**
log in	customer	**OAuth**

Source: Built in the demo: tools, resource, REST. Resource tested Nov 2, Claude Code 2.1.316: attached when the customer @-mentions it; the model never read it on its own. Skill and code API: on this slide only

Step 6: Pet Sales on the MCP Product Canvas

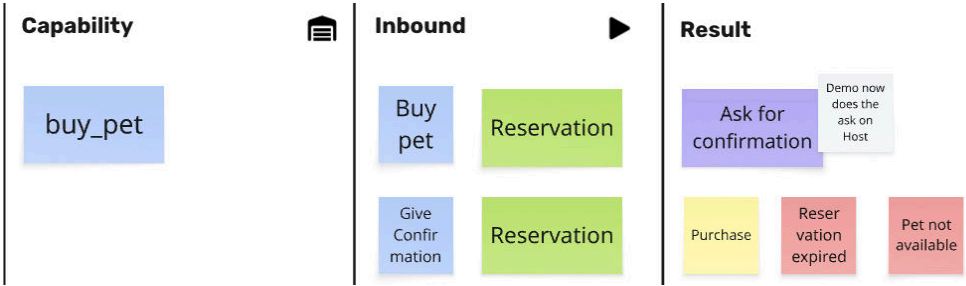


- One server per context and per audience: the buyer never sees admin tools
- Commands become tools; read models stay reads
- Results and errors in business words: they say what to do next

The rules live in the domain.
MCP is just the adapter.

Source: MCP Product Canvas: © Annegret Junker & Fabrizio Lazzaretti, after the API Product Canvas in *Crafting Great APIs with Domain-Driven Design*; Junker and Lazzaretti (2026)

Step 6: Pet Sales on the MCP Product Canvas



The `buy_pet` row. In the domain, Pet Sales asks for the confirmation; in the demo, the host asks before the call.

Source: MCP Product Canvas: © Annegret Junker & Fabrizio Lazzaretti, after the API Product Canvas in *Crafting Great APIs with Domain-Driven Design*; Junker and Lazzaretti (2026)

05

Before and after

Demo: “Buy me a friendly dog”



Claude Code, Haiku 4.5, the three domain servers; a typical take

User: Buy me a friendly dog
MCP calls: `find_available_pets` → `buy_pet`, the host asks
Agent: Rex bought, free pickup slots Tue, Nov 10

User: Tuesday 10 AM
MCP calls: `schedule_pickup`, the host asks
Agent: pickup scheduled

User: Buy Rex for my sister too
MCP calls: none
Agent: “Rex is no longer available”: Bello?

Sold means off offer.
Rules in the server.

Numbers: buy me a friendly dog

Servers: 1 → 3 Tools: 19 → 5 (+ 1 resource)

Sent to the model: ≈1,877 → **≈462 tokens (−75 %)**

	Wrapped API			By bounded context		
Model	order placed	still for sale	calls	bought	still for sale	calls
Opus 5.5	5/5	**5/5**	2.8	5/5	0/5	2.4
Haiku 5.5	5/5	**5/5**	3.0	5/5	0/5	2.0
Haiku 4.5	**0/5***	–	1.6	5/5	0/5	2.0

* Never looked at the shop: invented a dog with `addPet`, overwrote Rex in 2 of 5 runs, reported success in 4 of 5.

Checked after each run: *order placed* = `placeOrder` succeeded for a friendly dog on offer; *still for sale* = the Petstore still lists it as available; *bought* = an active purchase in Pet Handover.

The agent didn't need a smarter model. It needed the shop's domain.

Source: Evaluation Nov 1, 2026: same scope as the API (no pickup), n = 5 per side and model, same four pets, headless Claude Code 2.1.316; indicative; only Claude models tested. Input tokens per run: 20.7k → 14.4k (Opus 5.5), 23.9k → 15.1k (Haiku 5.5). Tokens ≈ chars ÷ 4 (7,509 → 1,850 chars). Counted: what Claude Code sends to the model (tool name, description, input schema, server instructions). Not counted: titles, annotations, output schemas, `_meta`, resources (on the wire: 16,642 vs. 4,088 chars)

MCP didn’t break anything.
It exposed a domain that never said what buying means.

A CRUD API you can’t change? The MCP server becomes the anti-corruption layer that builds `buy_pet` on top.

This is a pet shop. Now imagine your APIs.

- **The pet shop:** 19 endpoints, one meaning per word. Already: −75 % tokens, the pet really sold, an older small model kept on track
- **Real APIs:** hundreds of granular endpoints, many teams (GitHub's server in 2025: 102 tools)
- **Words that mean different things per context:** *order* in sales vs. logistics, *policy* in underwriting vs. claims. Even here, *pet* means three things
- One server per context with its own published language: translation happens at the borders, not in the model's head

The bigger the API, the more the cut matters.

Three things to do on Monday

1. **Start from the business**
   the story, the events, the bounded context: one context, one MCP server

2. **Describe it in business words**
   names, results and errors: the error message is the contract

3. **Then decide on the technique**
   tool, code API, skill, REST, host, or nothing

Think business first.
Then decide on the technique.

We ran a small evaluation. Run more, on your own tools.

Source: Evaluating tools with agents: Aizawa (2025); Anthropic (2025b)



Fabrizio Lazzaretti
Senior Managing Consultant
Wavestone
CNCF Ambassador

lazzaretti.me/agentics26
slides, demo, our books

References

A2A Project (2026). *Agent2Agent (A2A) Protocol Specification*. Version 1.0.0, Agentic AI Foundation (Linux Foundation). https://a2a-protocol.org/latest/specification/ (visited on 11/01/2026).

Agent Skills (2025). *Agent Skills Specification*. Open standard, published by Anthropic in December 2025. https://agentskills.io/specification (visited on 11/01/2026).

Aizawa, K. (Sept. 11, 2025). *Writing Effective Tools for Agents – with Agents*. Anthropic. https://www.anthropic.com/engineering/writing-tools-for-agents (visited on 11/01/2026).

Anthropic (2025a). *Code Execution with MCP: Building More Efficient Agents*. https://www.anthropic.com/engineering/code-execution-with-mcp (visited on 10/19/2026).

— (Sept. 10, 2025b). *Tool Evaluation*. Claude Cookbook. https://platform.claude.com/cookbook/tool-evaluation-tool-evaluation (visited on 11/02/2026).

DDD Crew (2026a). *Context Mapping*. CC BY-SA 4.0. https://github.com/ddd-crew/context-mapping (visited on 11/01/2026).

— (2026b). *The Bounded Context Canvas*. Version 5. CC BY-SA 4.0. https://github.com/ddd-crew/bounded-context-canvas (visited on 11/01/2026).

FastMCP (2026). *OpenAPI Integration*. Documentation. https://gofastmcp.com/integrations/openapi (visited on 11/01/2026).

GitHub (Oct. 6, 2025). *Adding Default Toolset Grouping*. github-mcp-server, pull request #1159. https://github.com/github/github-mcp-server/pull/1159 (visited on 11/01/2026).

Havaei, R., R. Liu, and M. Li (July 2025). *Supabase MCP Can Leak Your Entire SQL Database*. General Analysis. https://www.generalanalysis.com/blog/supabase-mcp-blog (visited on 11/02/2026).

References

Junker, A. (2026). *SamplesDddMeetAi: Skills for EventStorming, Context Map and Bounded Context Canvas*. https://github.com/Grinseteddy/SamplesDddMeetAi (visited on 11/02/2026).

Junker, A. and F. Lazzaretti (2025). *Crafting Great APIs with Domain-Driven Design*. Apress. https://lazzaretti.me/post/publications/2025/domaindrivenapidesign/.

— (2026). *MCP Product Canvas*. Miroverse. https://miro.com/miroverse/mcp-product-canvas/ (visited on 11/01/2026).

Kanjariya, T. (Oct. 8, 2026). *So What Is Codemode and Why Does MCP Need It?* Medium. https://medium.com/@TusharKanjariya/so-what-is-codemode-and-why-does-mcp-need-it-913891347737 (visited on 10/28/2026).

Kubernetes SIG Apps (2026). *Agent Sandbox*. https://github.com/kubernetes-sigs/agent-sandbox (visited on 11/01/2026).

Lowin, J. (July 10, 2025). *Stop Converting Your REST APIs to MCP*. https://www.jlowin.dev/blog/stop-converting-rest-apis-to-mcp (visited on 11/01/2026).

Model Context Protocol (2026a). *Architecture Overview*. Protocol revision 2026-07-28. https://modelcontextprotocol.io/docs/learn/architecture (visited on 11/01/2026).

— (2026b). *Tools*. Specification 2026-07-28. https://modelcontextprotocol.io/specification/2026-07-28/server/tools (visited on 11/02/2026).

Mohammed, S. and K. Chau (June 16, 2025). *Block's Playbook for Designing MCP Servers*. Block. https://engineering.block.xyz/blog/blocks-playbook-for-designing-mcp-servers (visited on 11/01/2026).

OpenAPI Initiative (May 17, 2026). *The Arazzo Specification, Version 1.1.0*. https://spec.openapis.org/arazzo/latest.html (visited on 11/01/2026).

References

Supabase (2026). *Supabase MCP Server*. Tool list: `execute_sql`. https://supabase.com/docs/guides/getting-started/mcp (visited on 11/02/2026).

Swagger (2026). *Swagger Petstore Sample (OpenAPI 3)*. Hosted at https://petstore3.swagger.io. https://github.com/swagger-api/swagger-petstore (visited on 11/01/2026).

Wardley, S. (Feb. 2015). *An Introduction to Wardley (Value Chain) Mapping*. Bits or Pieces? https://blog.gardeviance.org/2015/02/an-introduction-to-wardley-value-chain.html (visited on 11/01/2026).

Willison, S. (July 6, 2025). *Supabase MCP Can Leak Your Entire SQL Database*. On research by General Analysis. https://simonwillison.net/2025/Jul/6/supabase-mcp-lethal-trifecta/ (visited on 11/01/2026).



Agentics Day: MCP + Agents

NORTH AMERICA