- **FastAPI** → A **Python web framework** to build **HTTP APIs** (REST/JSON) for humans, services, browsers, etc. It's built on Starlette/Pydantic with automatic OpenAPI docs. [fastapi.tiangolo.com], [fastapi.tiangolo.com]
- **FastMCP** → A **Python framework for Model Context Protocol (MCP)** to build **servers and clients that expose tools/resources/prompts to LLM hosts** (e.g., Claude Desktop, IDE assistants). It focuses on LLM-to-tool integration and can even generate OpenAPI/FastAPI surfaces from MCP tools. [fastmcp.wiki], [pypi.org]

Core intent & audience

Aspect	FastAPI	FastMCP
Primary goal	Build high-performance **HTTP APIs**	Build **MCP servers/clients** that let **LLMs** call your tools and read resources
Main consumers	Browsers, microservices, mobile apps, human developers	**LLM hosts/clients** (Claude Desktop, editors, agent runtimes)
Typical output	JSON over HTTP with OpenAPI docs	MCP primitives: **Tools** (actions), **Resources** (read-only data), **Prompts** (templates) exposed via JSON-RPC over stdio/SSE transports

- FastAPI's charter is conventional API development with standards like **OpenAPI/JSON Schema**, interactive Swagger/ReDoc docs, and dependency injection for request handling. [fastapi.tiangolo.com], [fastapi.tiangolo.com]
- FastMCP's charter is **LLM integration** using the MCP standard (JSON-RPC 2.0, stdio and SSE transports) so models can **discover** and **invoke** your capabilities safely. [modelconte...rotocol.io], [en.wikipedia.org]

Architecture & transport

- **FastAPI** runs an **ASGI** app (often with Uvicorn) serving HTTP endpoints. Everything is request/response over HTTP. [devdocs.io]
- **FastMCP** implements MCP's client/server protocol. Communication is **JSON-RPC** over **stdio** (local) or **SSE/HTTP** (remote). It manages capability discovery, resource listing, and tool execution semantics tailored for LLMs. [modelconte...rotocol.io], [apxml.com]

Primitives vs. endpoints

- **FastAPI**: You define @app.get, @app.post routes; models validated by Pydantic; OpenAPI is auto-generated. [fastapi.tiangolo.com]

- **FastMCP**: You decorate Python functions with @mcp.tool (actions), expose **resources** (read-only context, like GET), and **prompts** for reusable interaction templates—all discoverable by the LLM host. [fastmcp.wiki]

Documentation & generation

- **FastAPI** gives you **interactive docs** (Swagger UI/ReDoc) out of the box for your HTTP API. [fastapi.tiangolo.com]
- **FastMCP** provides MCP docs and, notably, **can generate OpenAPI/FastAPI** layers from MCP definitions when needed—handy if you want both LLM-facing MCP **and** human-facing REST. [pypi.org]

Security & auth focus

- **FastAPI**: Built-in patterns for OAuth2/JWT, dependency injection for auth, typical web API security concerns. [geeksforgeeks.org]
- **FastMCP**: Emphasizes **enterprise authentication** (Google, GitHub, Azure, Auth0, etc.) for MCP apps and **user-governed tool execution** (approval flows) typical in LLM environments. [pypi.org], [modelconte...rotocol.be]

Typical use cases

Choose FastAPI when you need:

- A REST/JSON API for your web/mobile app or microservices.
- Automatic docs, validation, and high performance for conventional HTTP workloads. [fastapi.tiangolo.com]

Choose FastMCP when you need:

- An **LLM-aware interface** so assistants can call your functions (tools), read your data (resources), and use prompts—e.g., an AI DevOps agent that triggers pipelines or reads logs. [fastmcp.wiki], [datacamp.com]

How they can complement each other

You can **combine** them:

- Implement your AI-facing layer in **FastMCP** (expose tools/resources to LLMs).

- Expose parts of the same functionality to humans or services via **FastAPI** (or generate an OpenAPI/FastAPI surface from FastMCP). [pypi.org]

Quick decision guide

- **Are your primary callers humans/services over HTTP? → FastAPI**. [fastapi.tiangolo.com]
- **Are your primary callers LLM hosts/assistants needing tool/resource discovery & safe invocation? → FastMCP**. [modelconte...rotocol.io]
- **Need both?** Start with **FastMCP**, then generate an HTTP surface (or handcraft with FastAPI). [pypi.org]