From Tools to Platforms

MCP Patterns for Building Open Agent Ecosystems

Guangya Liu
JPMC / Executive Director

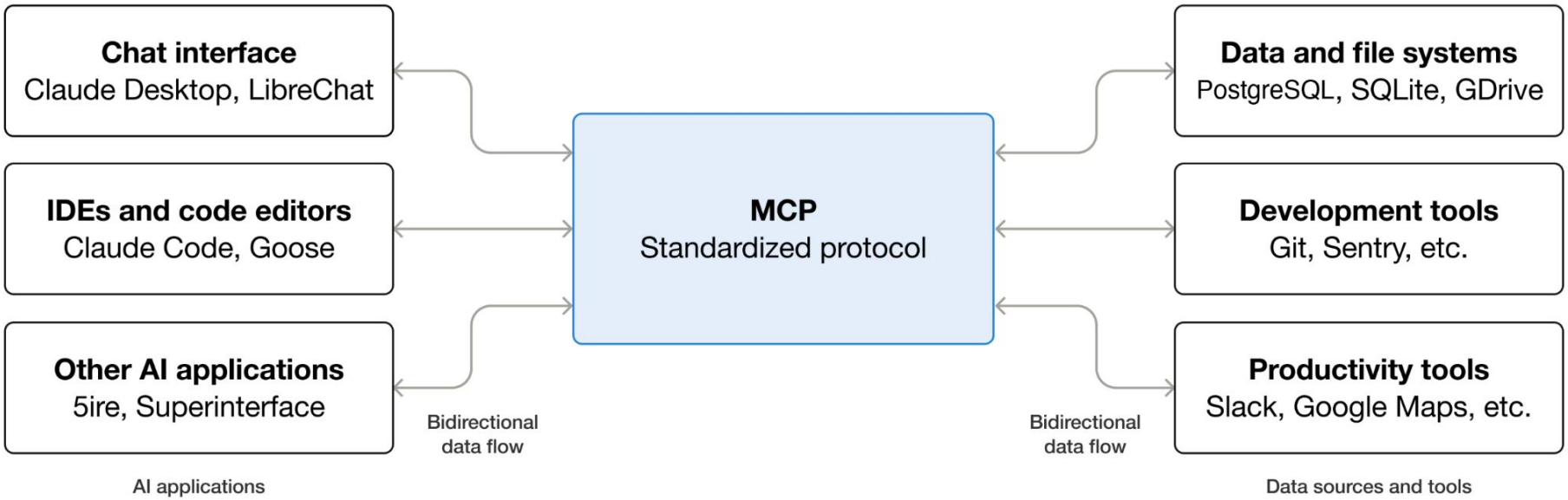
Self Intro

- Guangya Liu gyliu513@gmail.com
- Executive Director at JPMC (ex-RedHat, ex-IBM)
  - SRE Agent for JPMC
  - FullStack AI Observability with OpenTelemetry
  - GPU Optimization for AI/ML/HPC
- Open Source https://github.com/gyliu513
  - Contributor for llamastack(ogx), llm-d, kueue
  - Apache Mesos Committer and PMC Member
  - Former Maintainer for OpenTelemetry GenAI Semantic Convention
  - Former Maintainer for OpenStack

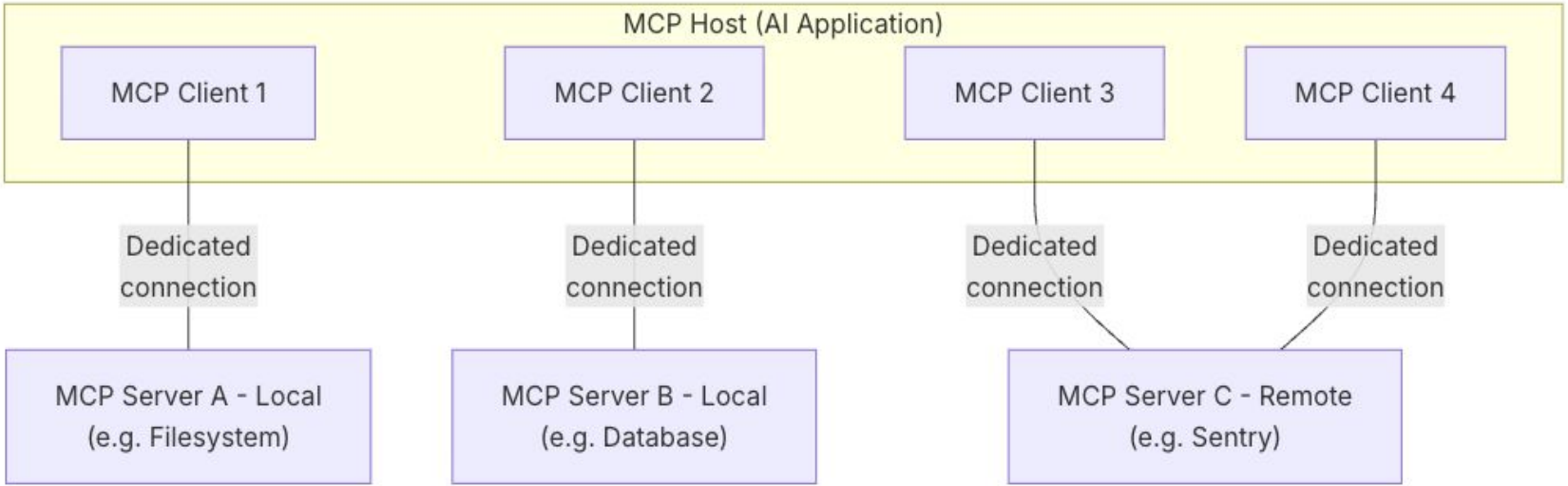
Agenda

- MCP Intro
- Single MCP Server
- Multiple MCP Servers

MCP Intro


Chat interface
Claude Desktop, LibreChat
IDEs and code editors
Claude Code, Goose
Other AI applications
5ire, Superinterface
AI applications
Bidirectional data flow
MCP
Standardized protocol
Bidirectional data flow
Data and file systems
PostgreSQL, SQLite, GDrive
Development tools
Git, Sentry, etc.
Productivity tools
Slack, Google Maps, etc.
Data sources and tools

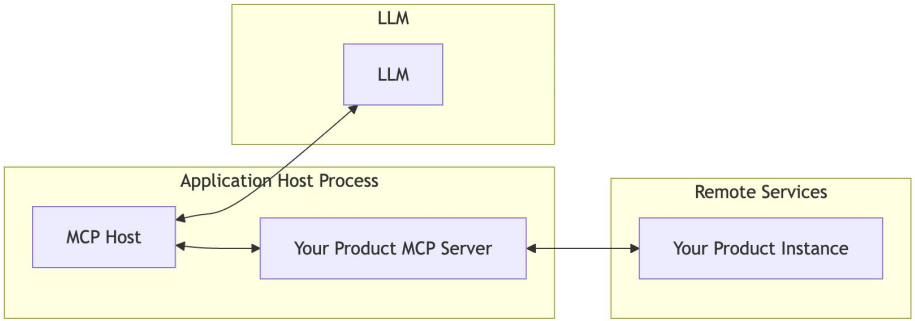
MCP Intro

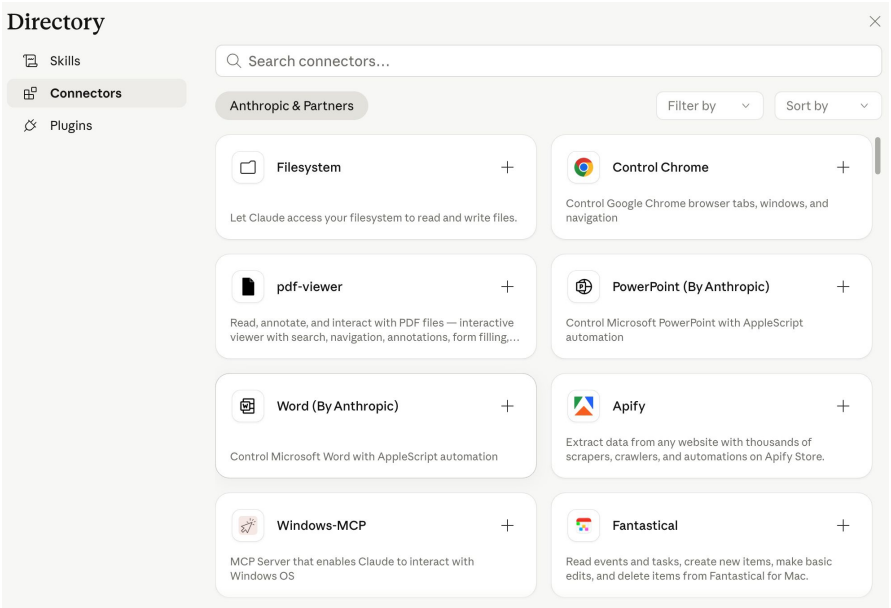

MCP Host (AI Application)
MCP Client 1
MCP Client 2
MCP Client 3
MCP Client 4
Dedicated connection
Dedicated connection
Dedicated connection
Dedicated connection
MCP Server A - Local (e.g. Filesystem)
MCP Server B - Local (e.g. Database)
MCP Server C - Remote (e.g. Sentry)

Product/Platform Level MCP Server

- External MCP Server
- Internal MCP Server

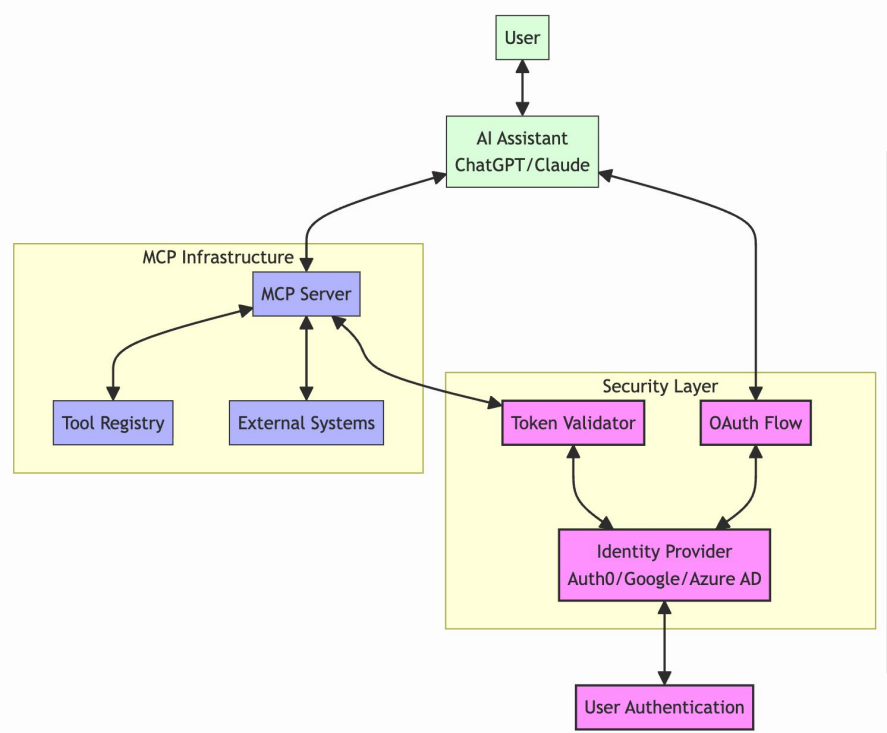
Platform/Product level External MCP Server

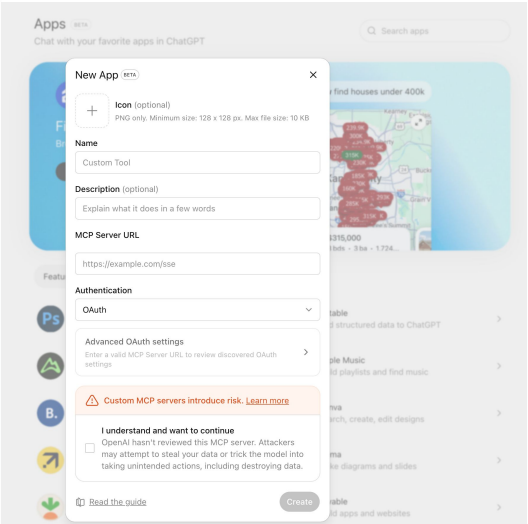




OAuth for External MCP Server

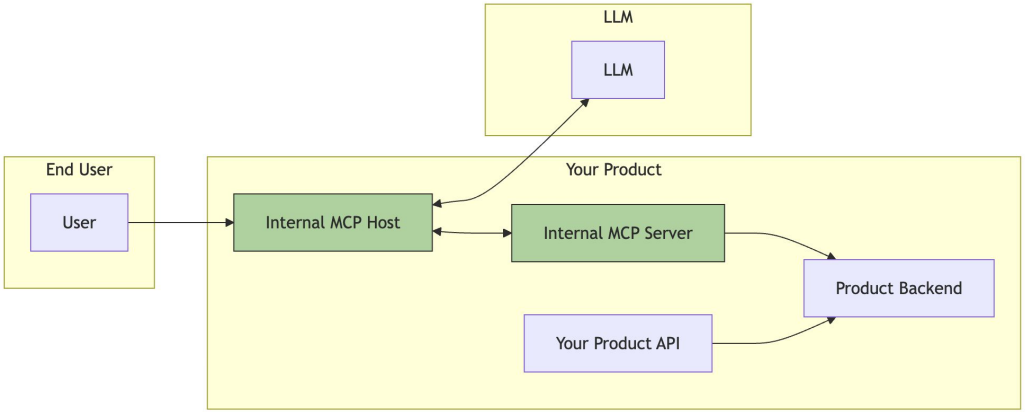
- Without OAuth
  - Copying access tokens by hand.
  - Managing token rotation and expiry.
  - Worrying about leaks in logs or environment variables.
- With OAuth
  - **Instant sign-in experience** — users click “Connect” and authorize via Google, GitHub, or SSO.
  - **No manual token management** — short-lived, auto-refreshed access tokens.
  - **Granular permissions** — only the scopes you need (`mcp.read`, `mcp.write`, etc.).



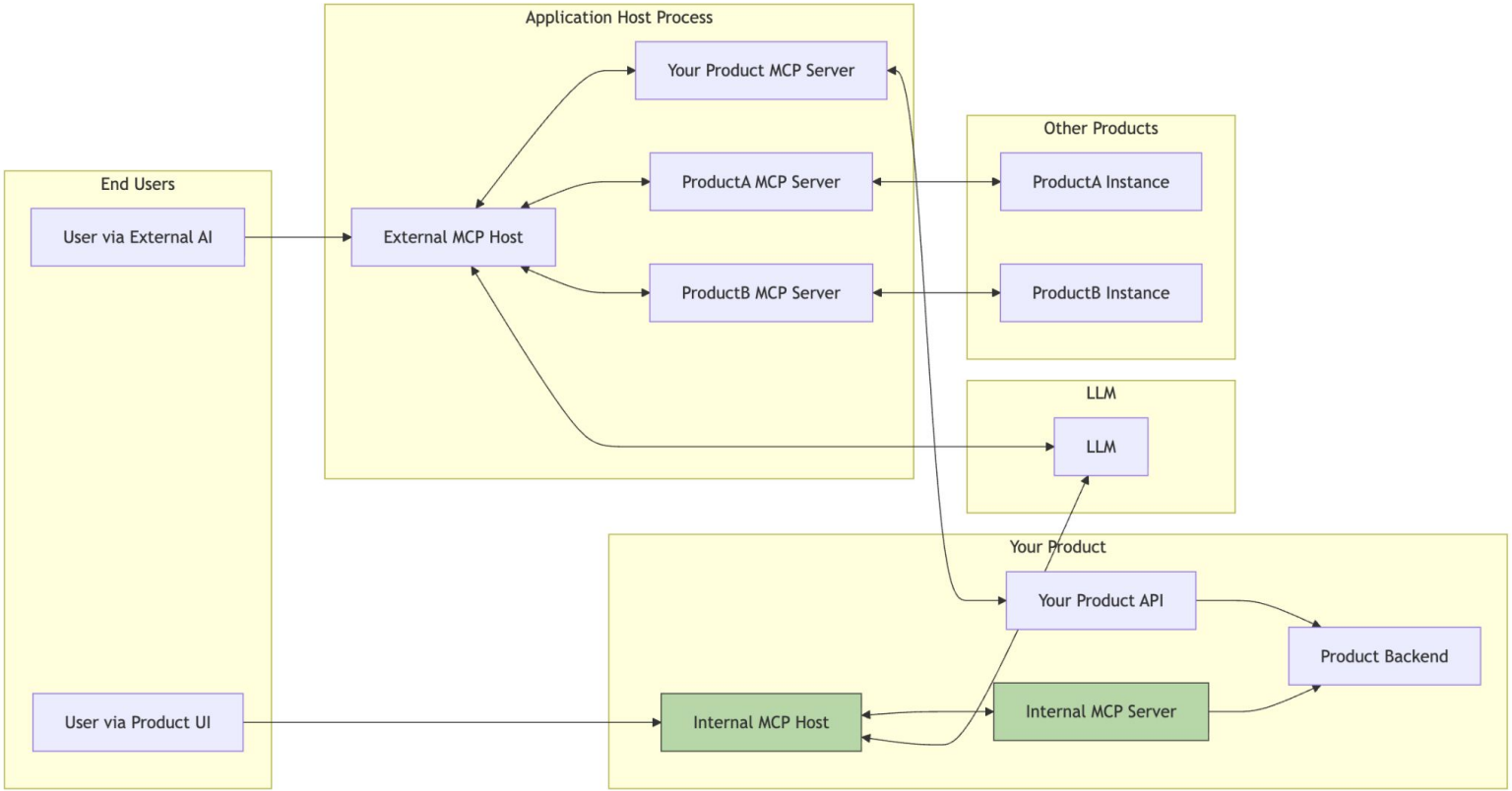


Platform/Product level Internal MCP Server

- Self Hosted MCP Host
  - Your own build-in chat bot for product
  - Dynatrace
  - DataDog
  - …



Combining them Together



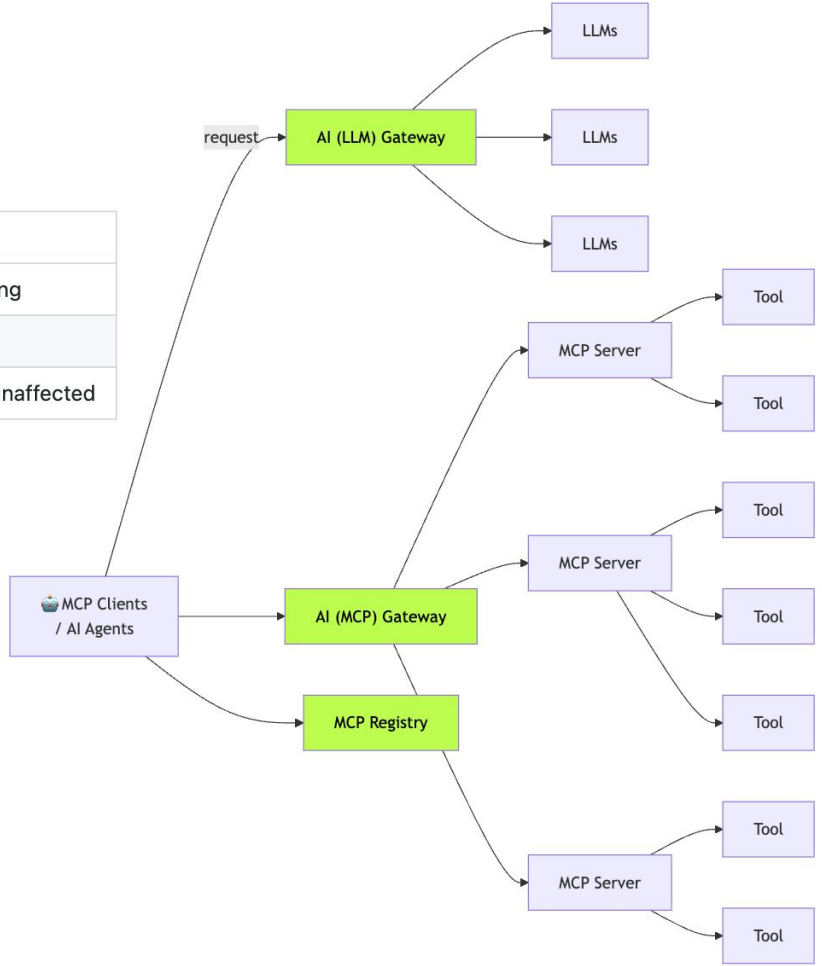
Multiple MCP Servers

- MCP Gateway
- Skill

MCP Gateway

Problem	Gateway's Solution
Agent needs to integrate dozens of Tools	Connect to one Gateway; it manages everything
Tool-level auth, rate limiting, auditing, observability	Handled centrally at the Gateway layer
Servers dynamically joining/leaving	Registry + Gateway handle discovery; Agent unaffected

```
Agent
  │
  ├─► MCP Registry        # 1. Discover available Servers/Tools
  │
  ├─► AI (LLM) Gateway    # 2. Call LLM when reasoning is needed
  │       └─► LLMs
  │
  └─► AI (MCP) Gateway    # 3. Call tools when action is needed
          ├─► MCP Server A ──► Tool 1, Tool 2
          ├─► MCP Server B ──► Tool 3, Tool 4, Tool 5
          └─► MCP Server C ──► Tool 6, Tool 7
```

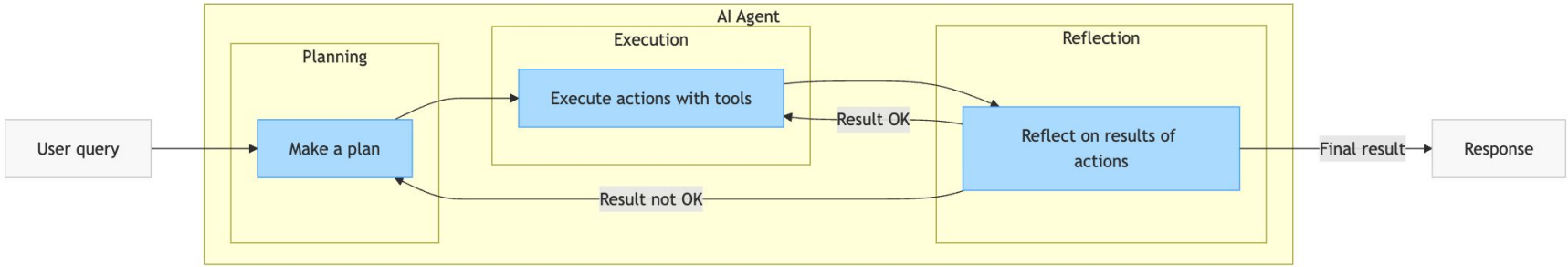


https://github.com/kgateway-dev/kgateway

https://github.com/obot-platform/obot

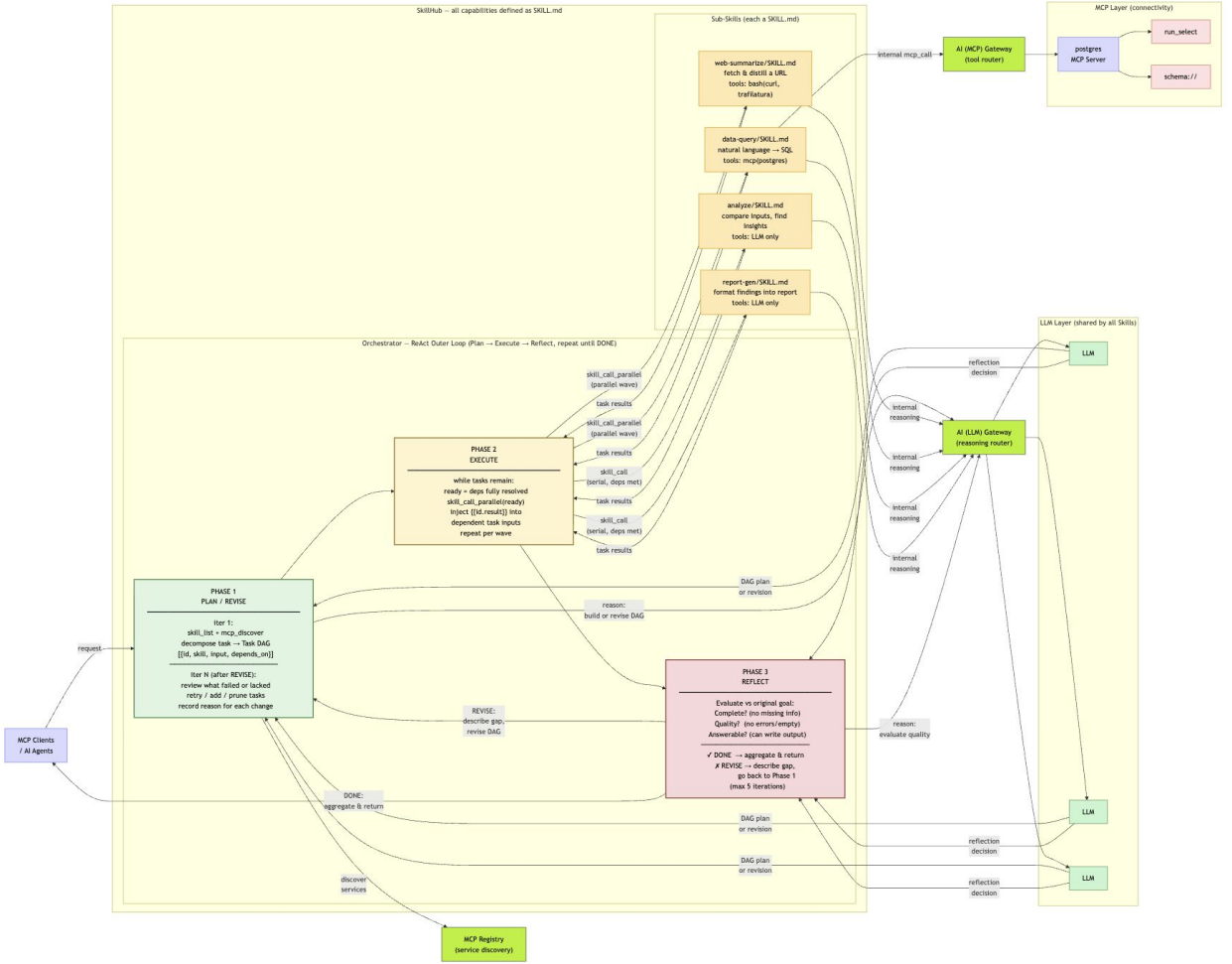
https://github.com/IBM/mcp-context-forge

ReAct Agent


AI Agent
Planning
Execution
Reflection
User query
Make a plan
Execute actions with tools
Result OK
Reflect on results of actions
Final result
Response
Result not OK

Skill

```
---
name: orchestrator
description: ReAct meta-skill — Plan → Execute → Reflect loop until goal is met
trigger: "orchestrate {task}"
tools:
  - bash: [skill_list, skill_call, skill_call_parallel]
  - mcp:  [mcp_discover, mcp_call]
permissions:
  skills: read
  mcp: all
max_iterations: 5
---

# Orchestrator Skill (ReAct Loop)

## Outer Loop — repeat until Reflect says DONE (max 5 iterations)

### Phase 1 · PLAN / REVISE
**iter = 1 (first time):**
- `skill_list()` + `mcp_discover()` — understand available capabilities
- Decompose task into Task DAG: `[{id, skill, input, depends_on}]`

**iter > 1 (after a REVISE):**
- Review: what results came back? what was empty/wrong/missing?
- Revise DAG: retry failed tasks with different skill, add missing tasks, prune useless ones
- Record reason for each change (for next reflection to reference)

### Phase 2 · EXECUTE
while tasks remain in DAG:
    ready = [tasks where all depends_on are resolved]
    results = skill_call_parallel(ready)
    inject {{id.result}} into dependent task inputs

### Phase 3 · REFLECT
Evaluate results against the **original goal** (not just "did tasks finish"):

1. **Complete?** — do we have all the information needed?
2. **Quality?** — any results empty, errored, or too shallow?
3. **Answerable?** — can we write a correct, complete final answer right now?

→ All pass: **DONE** — exit loop, go to Aggregate
→ Any fail: **REVISE** — describe the gap clearly, go back to Phase 1

## Aggregate (after loop exits)
Merge all results → return final markdown output.
```



Tools or MCP

- With Skills, should I use tools or MCP
- Tools
  - Short-lived execution
  - Agent-local capabilities
  - Lightweight interactions
- MCP
  - Long-lived integrations
  - External systems/services
  - Reusable and standardized connections

MCP Pattern

Single MCP Server

MCP Server per product/platform

Multiple MCP Servers

MCP Orchestration with Agent and MCP Gateway: Agent + MCP Gateway + MCP Servers

Skills + Multiple MCP Servers

MCP Orchestration with meta-skill: Agent + Meta-Skill + MCP Gateway + MCP Servers + Tools

Simple Calls: Skills + Tools

Complex System: Skills + MCP Servers

Thanks