De prompts sueltos a agentes

- Hasta ahora vimos:
  - Transformers, atención, fine-tuning
  - Prompting basico y RA
- Limitación: todo eso suele ser **una llamada aislada** al modelo
- En producción necesitamos:
  - Múltiples pasos
  - Estado / memoria
  - Llamar APIs, BDs, RAG, etc.
  - Controlar errores, reintentos, límites

¿Qué es un agente?

Un agente es un sistema que usa un LLM para **tomar decisiones paso a paso**, llamar herramientas y **mantener estado** para cumplir un objetivo del usuario.

Componentes típicos:

- Objetivo o tarea (goal)
- Contexto / memoria
- Modelo de lenguaje (LLM)
- Tools (acciones externas)
- Ciclo de razonamiento:
  - Pensar → decidir → actuar → observar → repetir

Comparación:

- Prompt “clásico”: función pura `input → output`
- Agente: **bucle** `estado + input → nuevo estado + acciones + output`

¿Por qué agentes?

Casos donde un sólo prompt no alcanza:

- Flujos multi-paso:
  - “Reservá un vuelo, verificá clima, proponé 2 opciones y fijate mi presupuesto”
- Integraciones complejas:
  - CRM + base de tickets + documentación interna
- Ajuste incremental:
  - El usuario corrige, agrega contexto, cambia de idea
- Tareas largas:
  - Investigar, resumir, proponer plan, iterar sobre feedback

Beneficios:

- Automatizan workflows, no sólo respuestas.
- Hacen explícito el flujo de decisiones.

Anatomía de un agente LLM

Usuario → (Mensaje) → **Agente** → (Respuestas + acciones)

Dentro del agente:

- **State**:
  - Historial de mensajes
  - Memoria (resumen, entidades, preferencias)
  - Variables de trabajo (ej. “ticket_id”, “carrito_compra”)
- **Reasoner (LLM)**:
  - Interpreta el estado actual
  - Decide próxima acción (tool a llamar / respuesta / pregunta)
- **Tools**:
  - Funciones externas (APIs, RAG, bases de datos, etc.)
- **Policies**:
  - Reglas de seguridad, límites, fallback

¿Qué es una “tool”?

Tool = acción externa que el LLM puede pedir ejecutar.

Ejemplos:

- `buscar_productos(query)`
- `consultar_stock(product_id)`
- `run_sql(query_sql)`
- `rag_search(question)`
- `enviar_correo(destinatario, asunto, cuerpo)`
- `programar_reunion(fecha, asistentes)`

Características:

- Tienen **input bien tipado**
- Tienen **output estructurado**
- Hacen algo **fuera del LLM** (datos frescos, efectos en el mundo real)

Diseño de tools

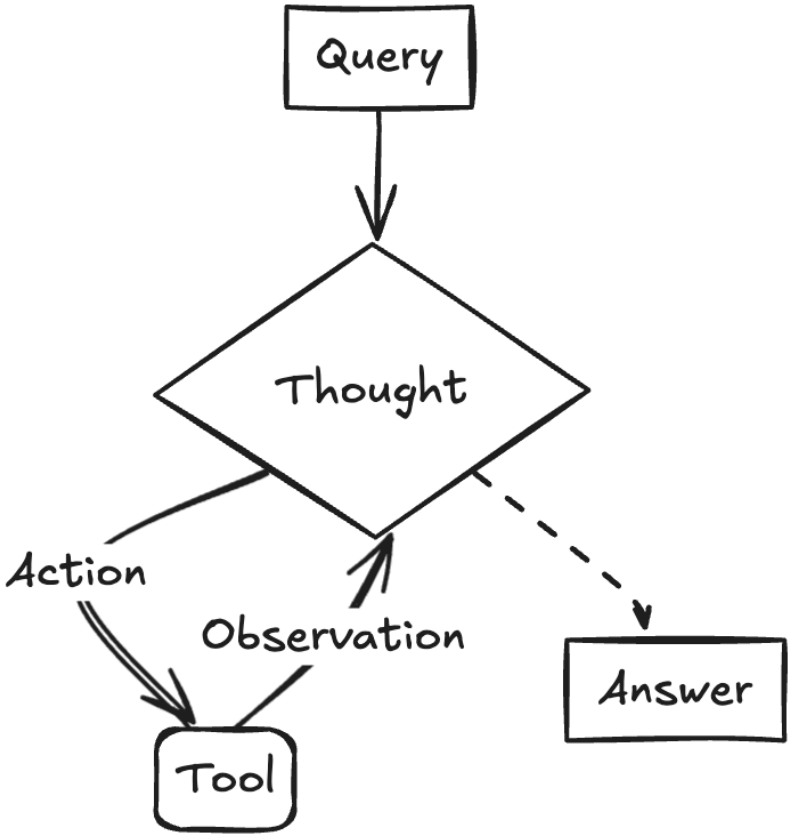
Buenas prácticas:

- Inputs y outputs **explícitos** (tipos, campos, unidades)
- Hacer tools **pequeñas y composables**:
  - Mejor 3 tools simples que 1 mega-tool confusa
- Pensar en **idempotencia**:
  - ¿Qué pasa si el LLM la llama dos veces?
- Manejo de errores:
  - Mensajes claros de “no encontré X”, “timeout”, etc.
- Limitar poder:
  - Tools peligrosas (borrar datos, enviar dinero) con más checks / sandbox

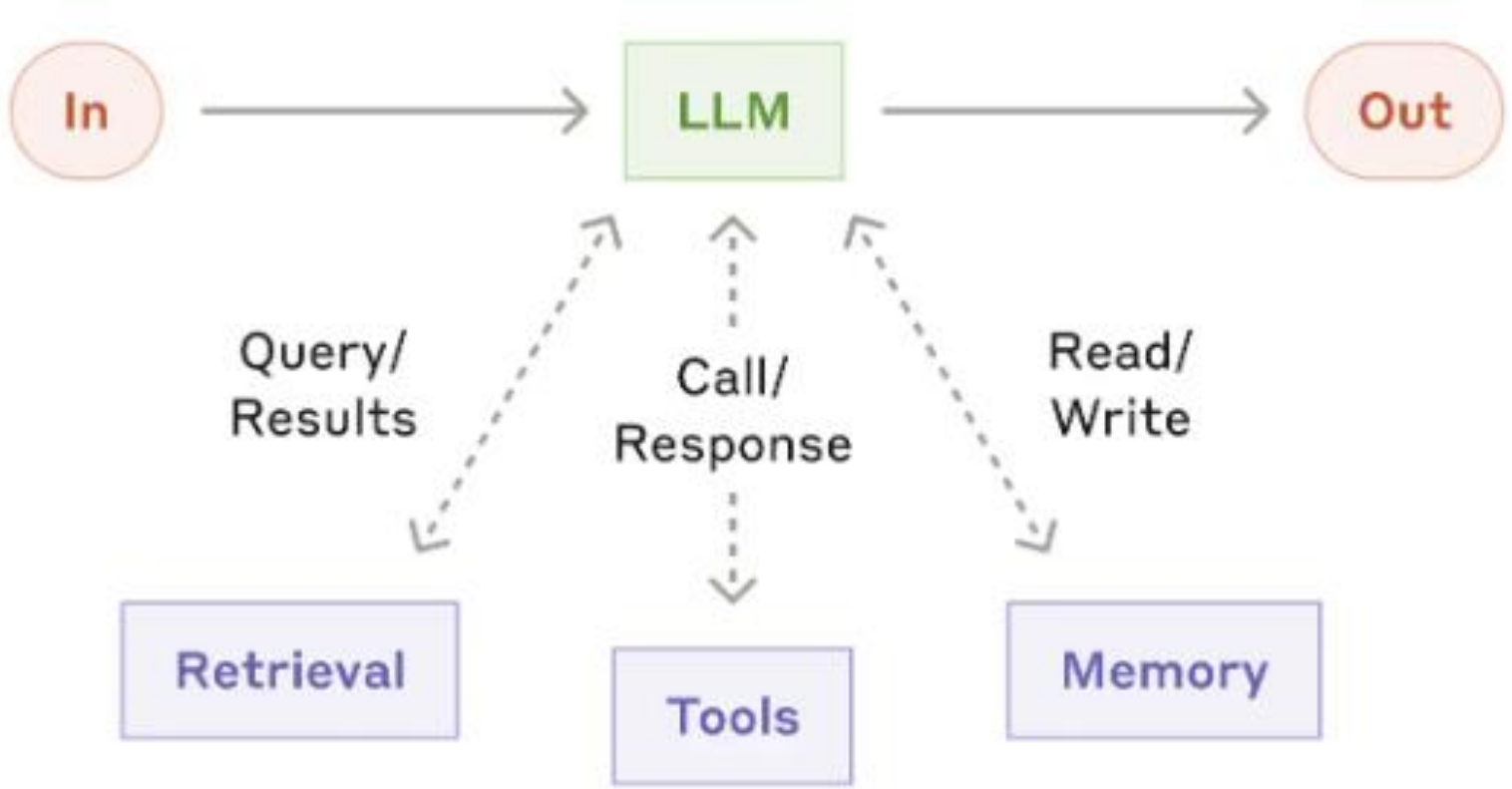
LLM + tools: ciclo de interacción

Patrón típico:

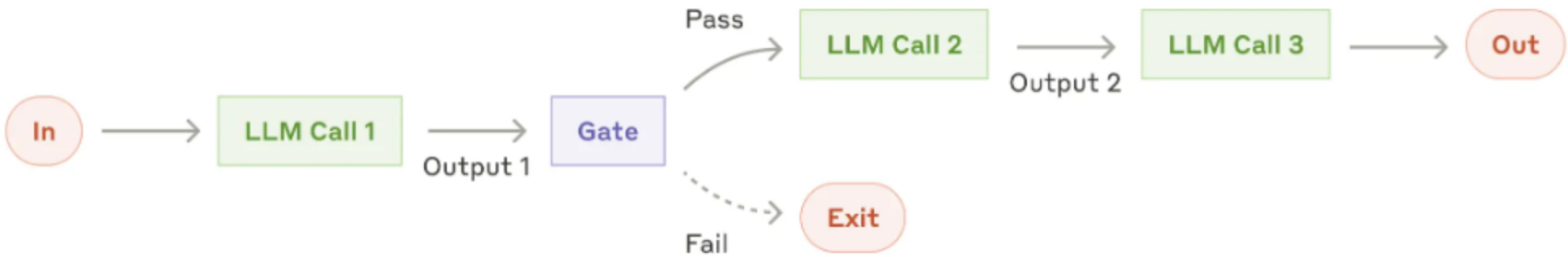
- Usuario manda mensaje
- Agente arma un “prompt de pensamiento”:
  - Historial relevante
  - Estado actual
  - Descripción de las tools disponibles
- El LLM decide:
  - Responder directamente
  - **O** pedir usar una tool con ciertos argumentos
- El runtime ejecuta la tool
- El resultado vuelve al LLM
- El LLM genera la respuesta final (o pide otra tool)
- Se actualiza el estado / memoria



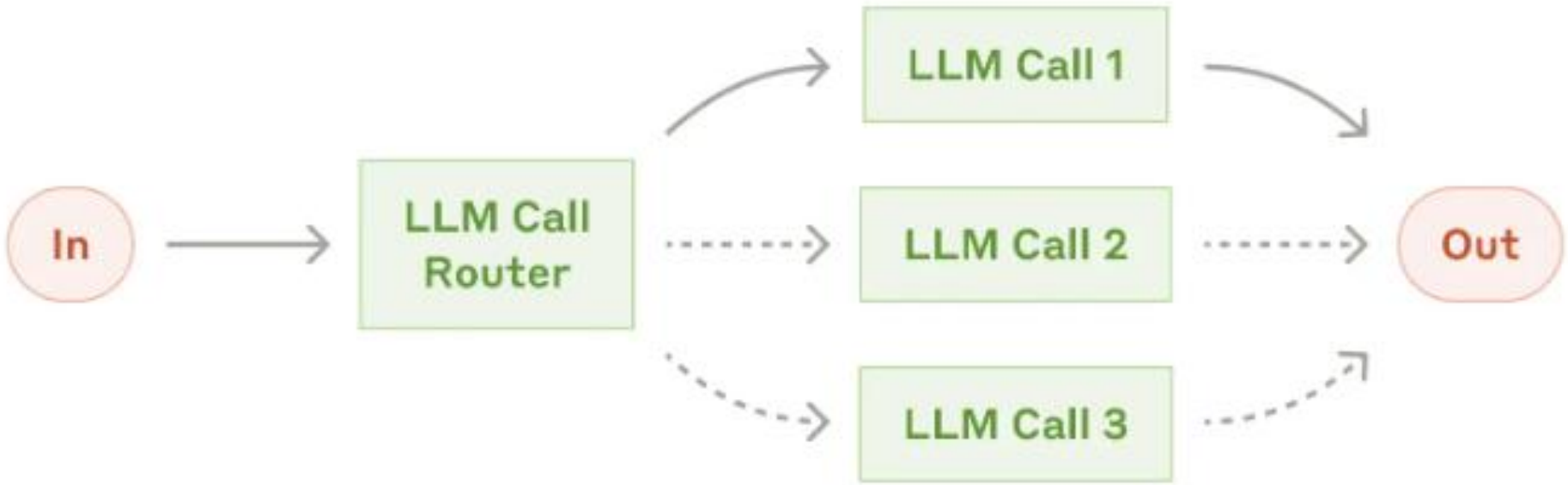
Patrones comunes de agentes – Basic LLM



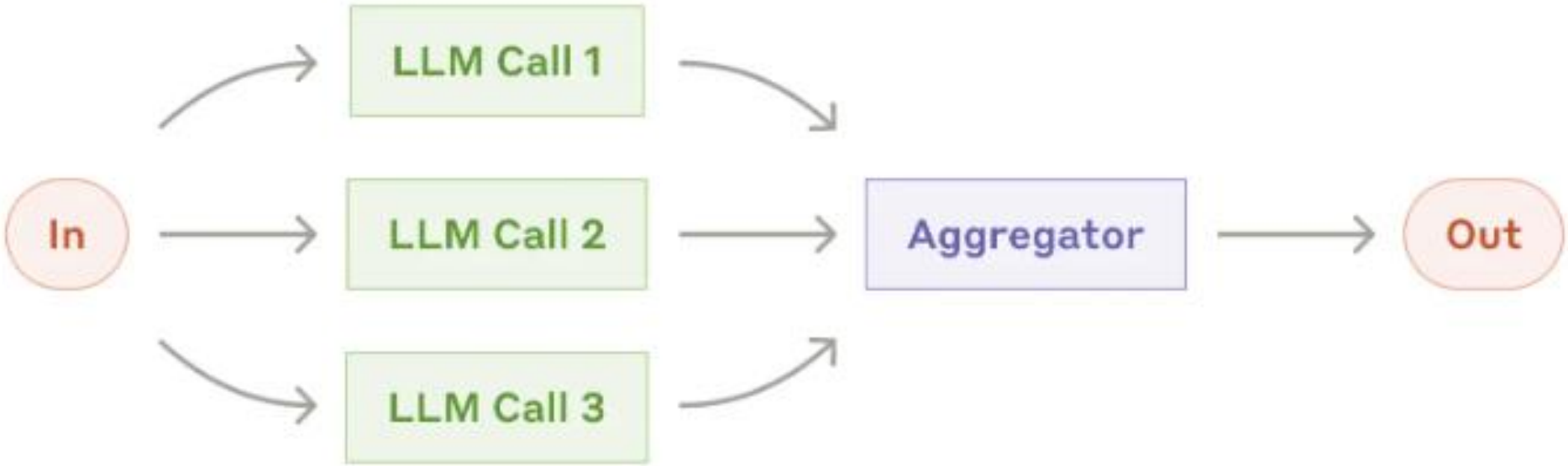
Patrones comunes de agentes - Prompt chaining



Patrones comunes de agentes - Routing



Patrones comunes de agentes - Parallelization



Patrones comunes de agentes - Orchestrator

Patrones comunes de agentes - Evaluator

La pila de protocolos

Contexto actual:

- Grandes players (Google, Anthropic, OpenAI, Microsoft…) están empujando **estándares** para:
  - Conectar agentes a datos y herramientas
  - Hacer que agentes **hablen entre sí**
  - Conectar agentes con **interfaces de usuario** ricas y en tiempo real

Tres capas clave que vamos a mirar:

- **MCP** – Model Context Protocol → agentes ↔ datos / herramientas
- **A2A** – Agent2Agent Protocol → agentes ↔ agentes
- **AG-UI** – Agent–User Interaction Protocol → agentes ↔ frontends

MCP – Model Context Protocol

¿Qué es MCP?

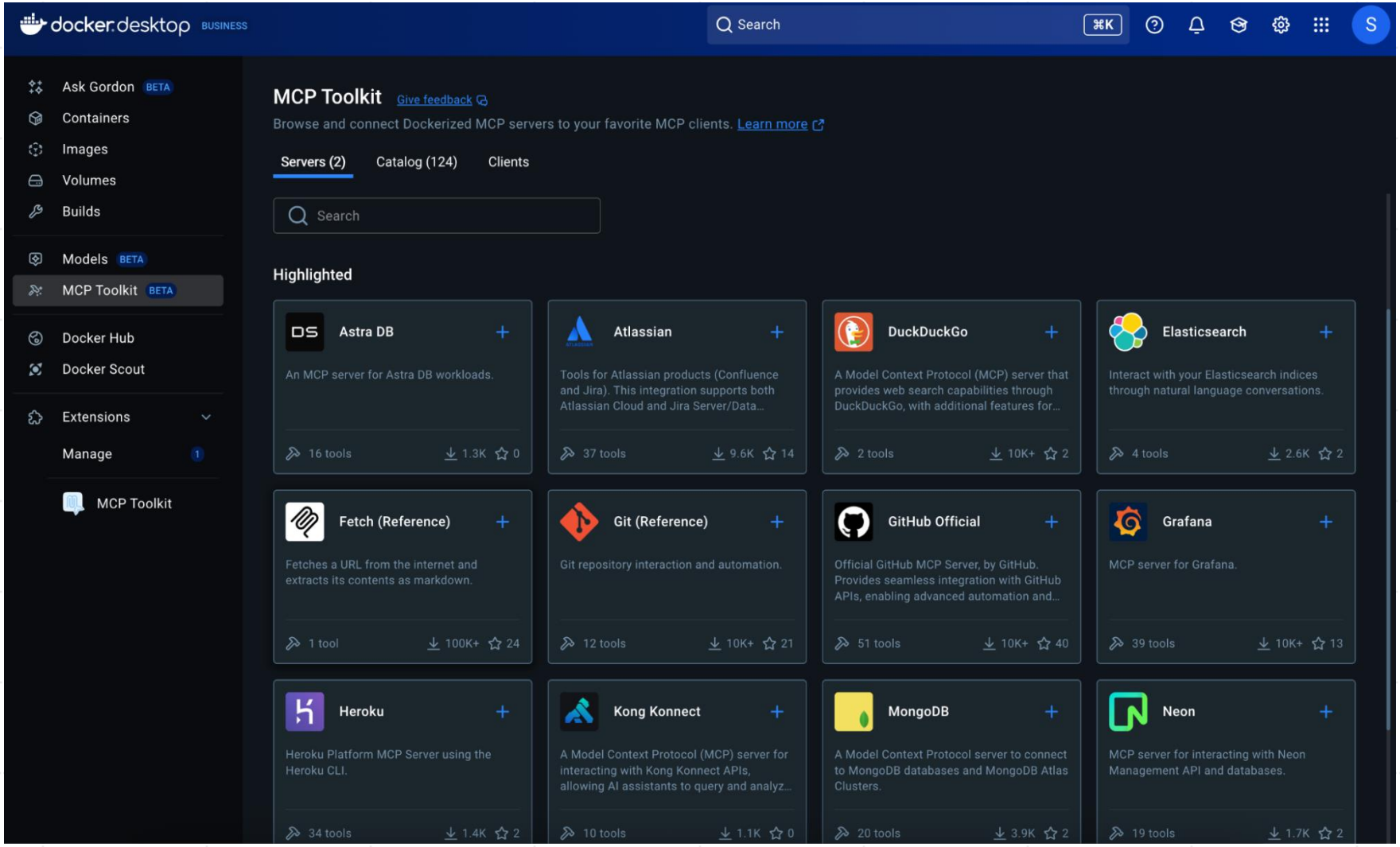
Estándar abierto para conectar aplicaciones de IA con fuentes de datos y herramientas externas.

- Nace del ecosistema Anthropic / Claude, pero es abierto.
- Un puerto estándar para enchufar:
  - BDs, APIs, sistemas internos
  - Herramientas (search, cálculos, workflows)
  - Repositorios de prompts / workflows

Conceptos clave:

- **MCP server**: expone tools y recursos (archivos, queries, etc.)
- **MCP client**: tu agente / aplicación que consume esos tools
- Tools tienen:
  - Nombre único
  - Esquema de inputs/outputs tipado
  - Metadata de permisos y contexto

MCP – Model Context Protocol



A2A – Agent2Agent Protocol

¿Qué es A2A?

Protocolo abierto para que agentes de distintos vendors y frameworks puedan comunicarse y coordinarse entre sí.

- Originalmente impulsado por Google y donado a la Linux Foundation.
- Objetivo: que un agente de:
  - Google / Vertex,
  - OpenAI / LangGraph,
  - Vendor X en otra nube
    - puedan hablar un mismo “idioma” estructurado.

¿Qué resuelve?

- Descubrimiento de agentes (“qué sabes hacer vos?”)
- Mensajes estructurados (intents, respuestas, errores)
- Seguridad / autenticación entre organizaciones
- Coordinación:
  - Agente “usuario” ↔ agente “comercio”
  - Agente “soporte” ↔ agente “facturación”

AG-UI – Agent–User Interaction Protocol

¿Qué es AG-UI?

Protocolo abierto y ligero para estandarizar cómo los agentes se conectan con aplicaciones de usuario (web, mobile, etc.).

- Se centra en el **canal tiempo real** entre:
  - Backend agéntico (LangGraph, Agent Framework, etc.)
  - Frontend (React, Next.js, móviles…)

Qué define AG-UI:

- Eventos estándar:
  - Mensajes de chat, streaming de tokens
  - Tool calls (y su render en UI)
  - Actualización de estado compartido (p. ej. un panel, formulario)
- Threads / sesiones desde el lado UI
- Patrones para:
  - Human-in-the-loop
  - Mostrar planes, pasos, errores.

Cómo encajan MCP, A2A y AG-UI

Capa UI (User ↔ Agente)

- AG-UI: eventos, estado compartido, streaming
- Frameworks: React + CopilotKit, Agent Framework, etc.

Capa Agente / Orquestación

- LangGraph, LangChain, Vertex ADK, OpenAI Agents SDK
- Definen:
  - Estado del agente
  - Tools internas (RAG, BDs, APIs)
  - Memoria, seguridad, routing

Capa Conectores / Datos / Otros agentes

- MCP: conectar con datos y tools externos estándar
- A2A: hablar con otros agentes, internos o de terceros
- Otros protocolos emergentes (pagos AP2, credenciales, etc.)