CHAPTER 9 / 3 MINUTE BRIEF

Orchestration and Agentic Topologies

Read the full chapter → / Read this brief online →

The main idea

A forecast built on model tokens cannot explain an agent's cost: the memory, tools and runtime around each call carry their own bills and limits, and the agent's loops change the shape of consumption itself. Consider a hypothetical support assistant that looks up a customer's account, drafts a reply, checks a refund rule, updates its memory and writes a trace. The customer sees one answer, but several systems did work.

A memory store, a tool server and a gateway may begin as small additions, then acquire their own bills and scaling constraints. A loop rereads its growing history every turn, a lead agent fans work out to parallel workers, retries stack across layers, and a task waiting for a person can hold a sandbox for hours. Some failures interrupt the task; others leave it running with incomplete context or stale data. The capacity team needs to follow both the cost and the behavior back to the work these components support, then ask whether that work was worth what it cost.

Explore this section in the full chapter →

Topologies

An agent may work alone with tools, pass work through a pipeline, or delegate to parallel workers. Each arrangement changes context reuse, latency and coordination. None is reliably the cheapest: isolating a subtask can avoid replaying irrelevant context, while unnecessary delegation can duplicate it. Even a single agent can accumulate substantial input consumption by repeatedly sending its growing history.

Explore this section in the full chapter →

The operating inventory

This is the inventory I would keep for the layer; check your own stack against it for lines nobody owns.

Component	What it is	Consumption scales with	Bill shape
Model gateway	Routing, retries and fallbacks	Traffic × retries at the gateway	Gateway tier, request charges or self-run compute; model charges remain separate
Prompt cache	Reuse of identical prompt prefixes	Repeated prefixes within the cache lifetime	Premium on writes, discount on reads; reads may not count toward rate limits[1]
Session store	Conversation state with time-to-live expiry	Concurrent sessions × retention	Stored state and infrastructure; expiry is a policy choice
Memory service	Extracted facts, episodes and instructions	Writes × model calls per write; retained history	Extraction and embedding calls, retrieval queries, storage under the retention policy
Vector database	Similarity search over embeddings	Vectors × dimensions × replicas, plus query rate	RAM-heavy nodes or hosted tiers
Graph database	Entities and relationships, sometimes with validity intervals	Entities × traversal rate; superseded facts if kept	Licensing varies by engine; self-hosting is an operations obligation
Tool/MCP servers	Structured access to external systems	Agent activity × tools per step × retries	Per-seat or per-call hosted, or self-run processes; downstream charges separate
Sandbox runners	Isolated execution for generated code	Arrival rate × holding time, including waits	Compute time or session charges; startup and expiry affect cost
Observability	Traces of agent steps and tool calls	Agent activity × sampling rate	Ingestion plus storage; sampling is a cost decision

Explore this section in the full chapter →

[1] Anthropic, “Prompt caching” and “Rate limits”, checked October 4, 2026. Five-minute writes at 1.25 times the base input price, one-hour writes at 2 times, reads at 0.1 times (less on some newer models); hits need identical prefixes; on most models, cached reads do not count toward input-tokens-per-minute limits. Provider-specific and subject to change.

What the review inherits

Agent spend has the shape of Volume II, Chapter 15's long tail — thousands of small autonomous decisions, no single owner watching each one — inside the shape of Volume II, Chapter 17's relationships, because the workflow owners are the people who know what the agents should be doing. I would have the capacity team build an attribution record first, tying each event on each system to an agent, workflow, customer and task. With it, the monthly review in Volume II, Chapter 22 can ask what the agents completed and what that work cost across every layer, and the workflow owner can say what it was worth.

Questions I would bring to that review:

- Which events, on which systems, carry no agent, workflow or customer?
- What did an accepted task cost across all layers, counting failed and abandoned attempts, and how wide is the spread?
- Which resources are held rather than used, and what releases them?
- How many attempts can one failure produce, and where is the retry budget enforced?
- What does the memory store keep, and does deletion reach its indexes, history and backups?

Explore this section in the full chapter →