READ MODE · ANSWER FIRST

How Clanker Cloud Works

An interactive, visual explanation of How Clanker Cloud Works, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

The whole story in 6 lines

How Clanker Cloud Works becomes easier to reason about when every stage is connected as one system.

1. Fast keyword inference handles the common path; only ambiguous prompts trigger an LLM tie-breaker.
2. Flags, config, env vars, and defaults have a strict precedence so requests do not drift across environments.
3. Backend-first auth enables cross-machine use, but local profiles still keep the system usable when the backend has no secret for that...
4. This is the grounding step: enough evidence to answer well, but not so much context that the model gets flooded.
5. The AI stage emits a constrained operation list, not free-form prose, so later execution stays observable and bounded.
6. Concurrency lowers wall-clock latency, but the system still tracks dependencies, failures, and aggregate metadata.

STAGE 1 OF 8

Transcript Turn Extraction

Start with the raw input. Clanker Cloud may receive a full chat transcript, not just the final question.

Before routing, it scans for speaker markers so assistant replies and wrappers are not mistaken for user intent.

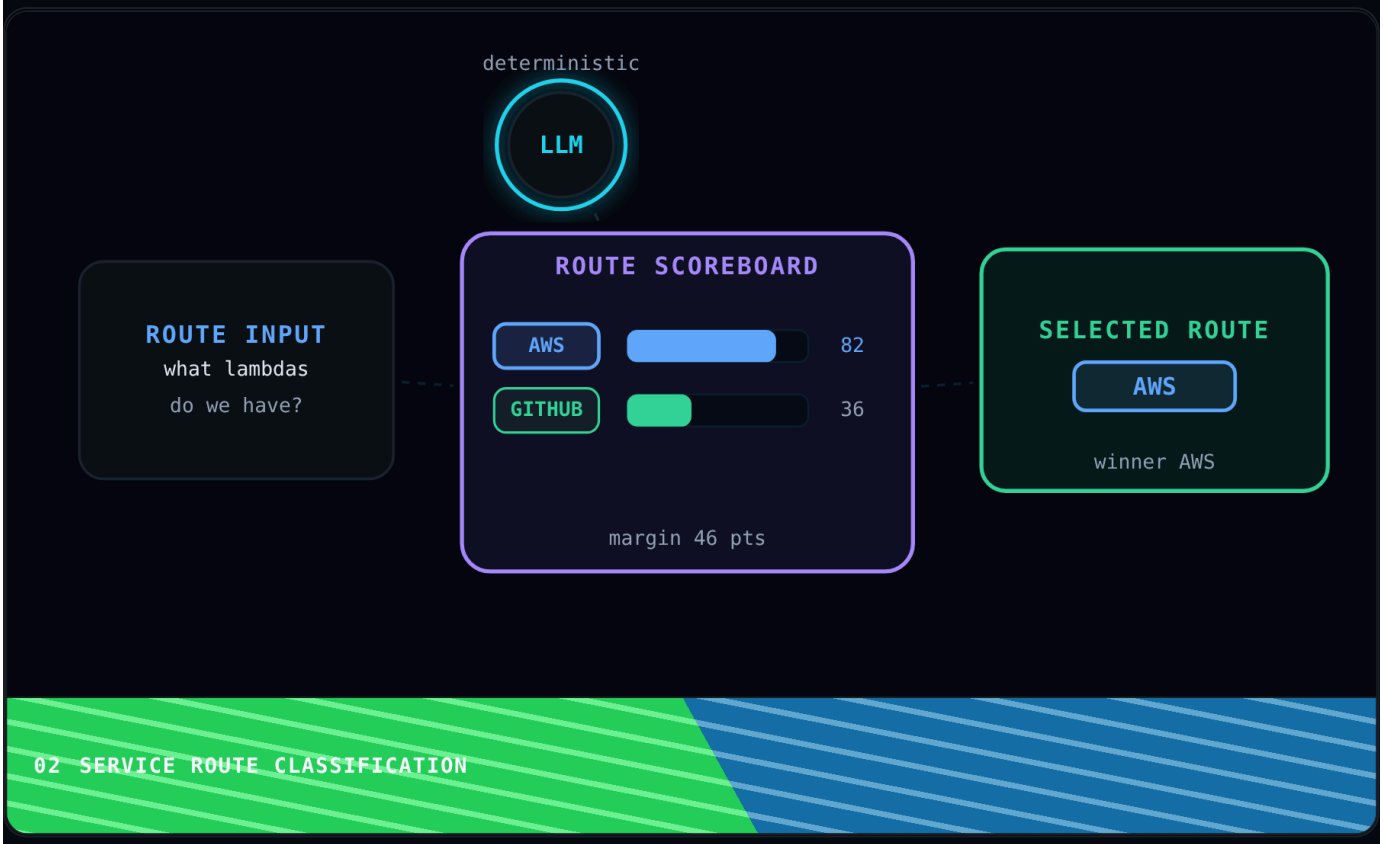
Then it isolates the newest explicit user turn, because that is the part that actually needs an answer.

Everything older gets trimmed away. The goal is to keep context noise from steering the route.

What leaves this stage is one clean question string, ready for downstream routing.

STAGE 2 OF 8

Service Route Classification



02 SERVICE ROUTE CLASSIFICATION

Now the system has a clean question but still does not know which cloud surface should answer it.

Keyword and pattern rules light up the likely service families, such as AWS, Kubernetes, or Cloudflare.

If one family is clearly dominant, Clanker can route immediately without spending model budget.

If the signals conflict, an LLM breaks the tie by choosing the most plausible route.

The output is one winning route. Everything else is discarded so later stages stay focused.

STAGE 3 OF 8

Backend Endpoint Resolution

03 BACKEND ENDPOINT RESOLUTION

Routing alone is not enough. Clanker still has to decide which backend endpoint and auth context this run should use.

It checks explicit flags first, because user-supplied overrides should win over saved defaults.

Anything still unresolved falls through config files and environment variables in a fixed order.

If there is no custom URL, the named environment maps to a known backend base URL.

By the end of this stage, the request is pinned to one backend and one auth source.

STAGE 4 OF 8

Credential Hydration Fallback

With the backend resolved, each provider branch asks the cloud backend for credentials before trying local state.

On a hit, Clanker can hydrate the provider client immediately and keep moving.

On a miss or error, it falls back to local profiles or machine config so the run can still succeed.

The important idea is that both paths converge on the same result: a ready-to-use provider client.

Once that client exists, Clanker can gather live infrastructure evidence without credential guesswork.

STAGE 5 OF 8

Provider Context Harvest

05 PROVIDER CONTEXT HARVEST

Now the system starts grounding the question in real infrastructure state.

Only the clients on the selected route run queries, so unrelated providers stay idle.

Each client fetches targeted slices of context rather than dumping a full inventory of the account.

If the question is exploratory, discovery mode widens the search before the final filter runs.

Low-value fragments are pruned and the survivors are merged into one prompt-ready evidence bundle.

STAGE 6 OF 8

Operation Plan Compilation

06 OPERATION PLAN COMPILATION

Clanker still does not answer immediately. First it decides which follow-up operations are actually worth running.

The planner looks at the cleaned question and the fresh evidence bundle together.

From there, the model proposes candidate operations that could close the remaining information gaps.

Confidence and budget gates trim that list so weak or redundant probes do not run.

The result is a compact execution plan that later stages can run and observe deterministically.

STAGE 7 OF 8

Parallel Probe Execution

07 PARALLEL PROBE EXECUTION

Execution begins only after the plan is fixed and validated.

Independent operations are scheduled across workers so wall-clock time drops without losing structure.

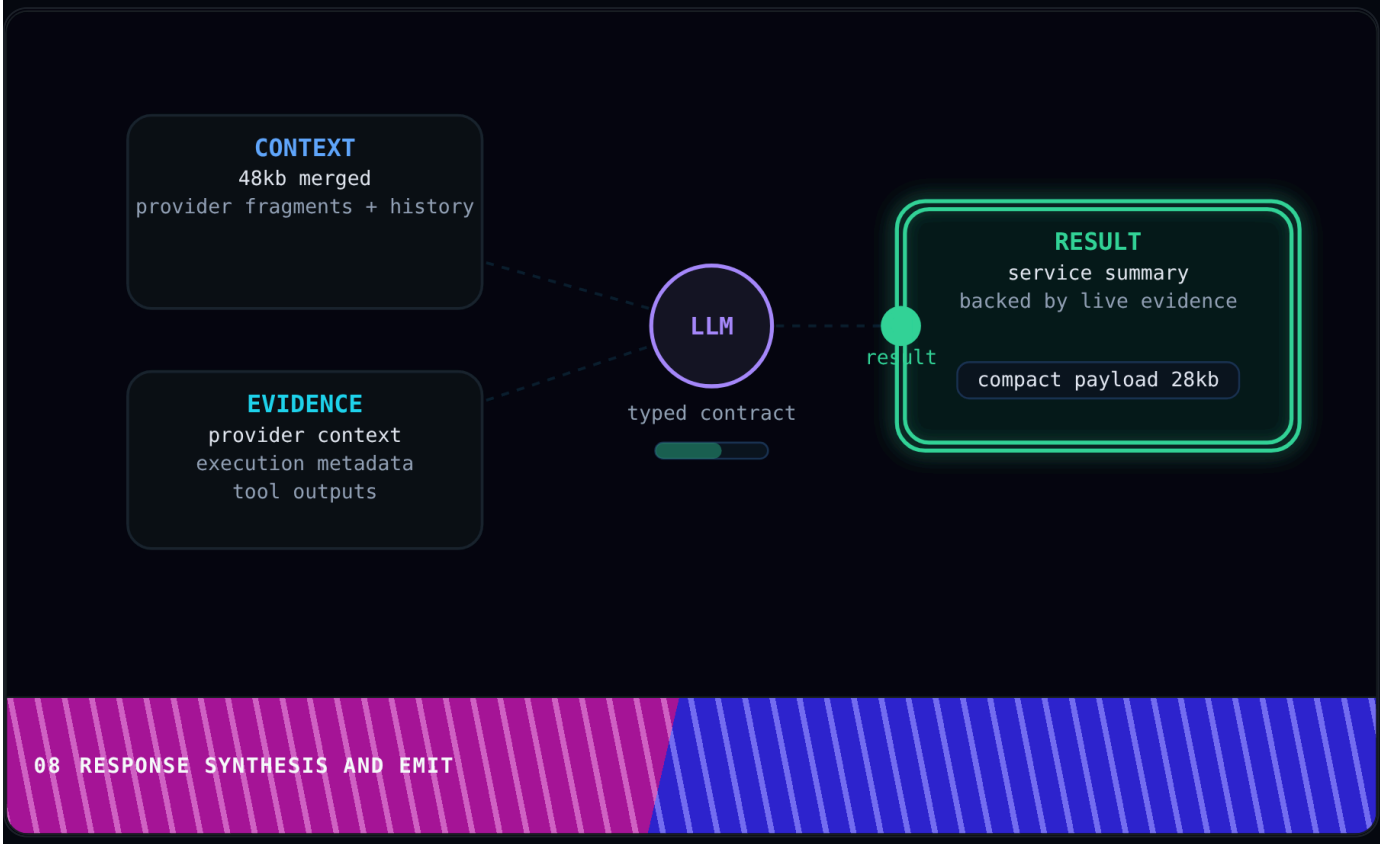
As workers finish, evidence streams back even while slower probes are still running.

Timeouts and local pacing rules keep the system bounded instead of letting probes fan out uncontrollably.

Everything that completes is folded into one aggregate evidence payload with status metadata attached.

STAGE 8 OF 8

Response Synthesis And Emit



08 RESPONSE SYNTHESIS AND EMIT

The final model step starts from merged inputs: harvested context plus whatever live evidence the workers found.

If that payload is too large, Clanker compacts it before synthesis so the last step stays within budget.

The compaction logic keeps the strongest facts and drops detail that no longer helps answer the question.

Instead of emitting loose prose, Clanker formats the result as a typed contract the client knows how to render.

That typed payload is what goes back to the UI or API caller as the final answer, plan, or error.