READ MODE · ANSWER FIRST

Getting Started with Clanker

An interactive, visual explanation of Getting Started with Clanker, from setup through the complete system.

CHEAT SHEET · 5 ESSENTIAL IDEAS

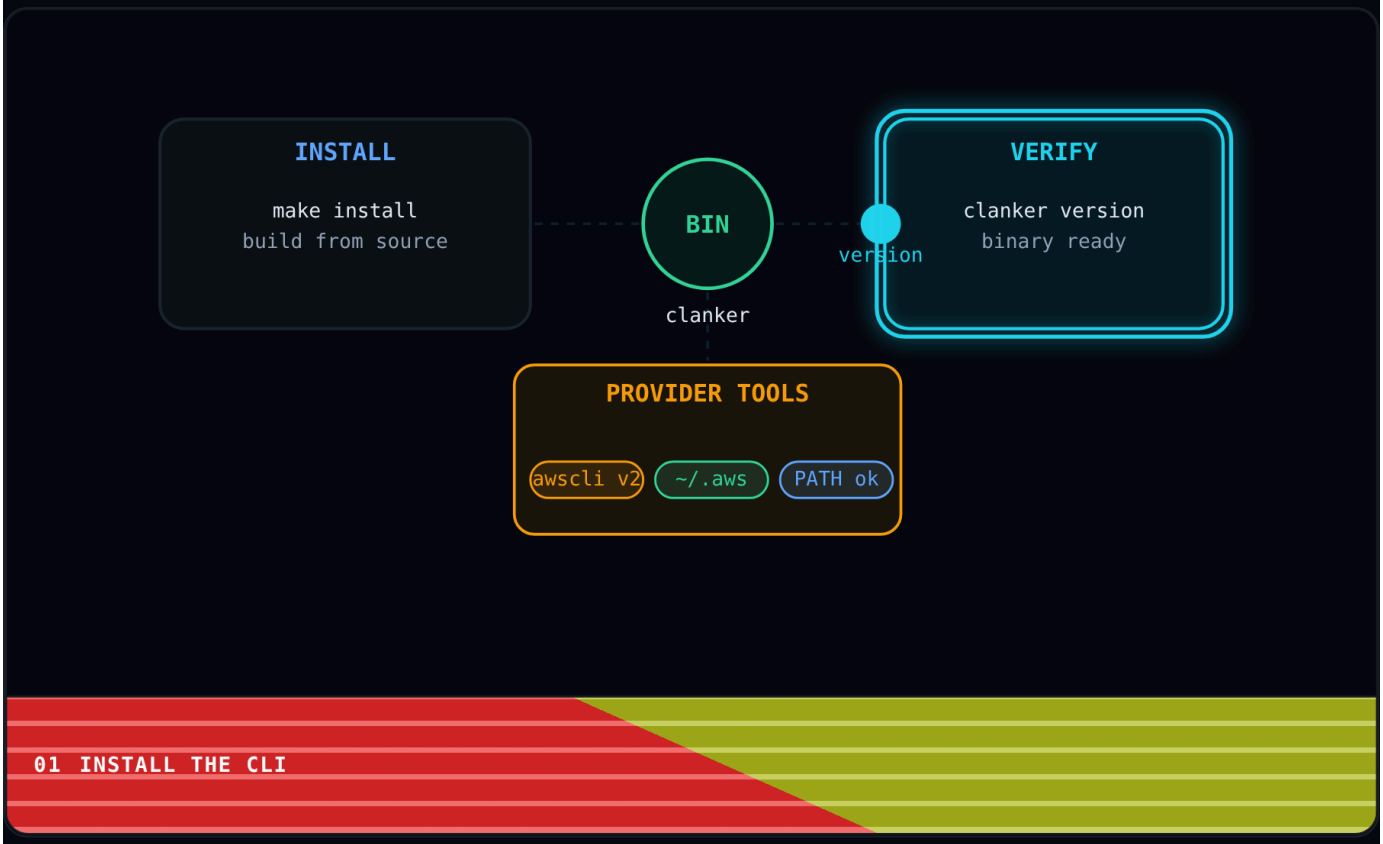
The whole story in 5 lines

Getting Started with Clanker becomes easier to reason about when every stage is connected as one system.

1. You only need a small baseline: an AI profile, an infra default, and the env vars or config keys that satisfy that profile.
2. Config defaults are only the starting point. Runtime flags like `--profile`, `--ai-profile`, and provider selectors win when they are...
3. Clanker does not shotgun every backend. It classifies the surface first, then runs only the evidence probes that matter for that question.
4. The safe path is prompt -> JSON plan -> human review -> `--apply`. Destructive requests stay behind `--destroyer` and explicit confirmation.
5. The path is local credential export -> backend vault -> remote ask with `--api-key`, but Clanker still falls back to local config when...

STAGE 1 OF 7

Install the CLI



01 INSTALL THE CLI

Start by choosing how to install the binary. Homebrew works, but `make install` usually tracks the freshest source.

Either way, the goal of this stage is simple: get one local `clanker` executable onto the machine.

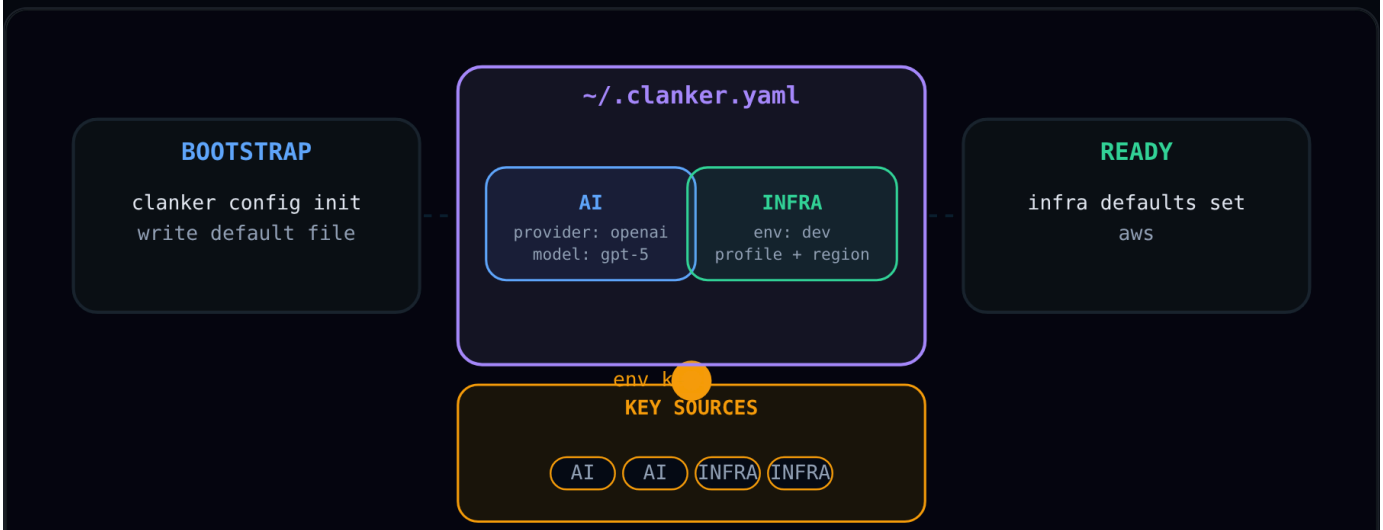
Clanker can only query real cloud surfaces if the matching provider CLIs, such as AWS CLI v2, are already available.

Run `clanker version` next. That quick check proves the binary is on your PATH and callable.

Once this passes, you have a working entrypoint for the rest of the workflow: `ask`, `config`, `credentials`, `talk`, and `mcp`.

STAGE 2 OF 7

Bootstrap ~/.clanker.yaml



02 BOOTSTRAP ~/.CLANKER.YAML

The next job is to give Clanker a baseline configuration instead of forcing every run to start from zero.

You can generate that baseline with `clanker config init` or copy the example file into `~/.clanker.yaml`.

Inside the file, choose the default AI provider and model family Clanker should reach for first.

Then wire in the provider key and the default infra environment so runs have a home context.

Think of `~/.clanker.yaml` as the starting state. Later flags can override it, but every run begins here.

STAGE 3 OF 7

Resolve Execution Context

03 RESOLVE EXECUTION CONTEXT

When you run `clanker ask`, the config file is only the starting point, not the final truth.

Clanker first loads the defaults it can find from config and environment variables.

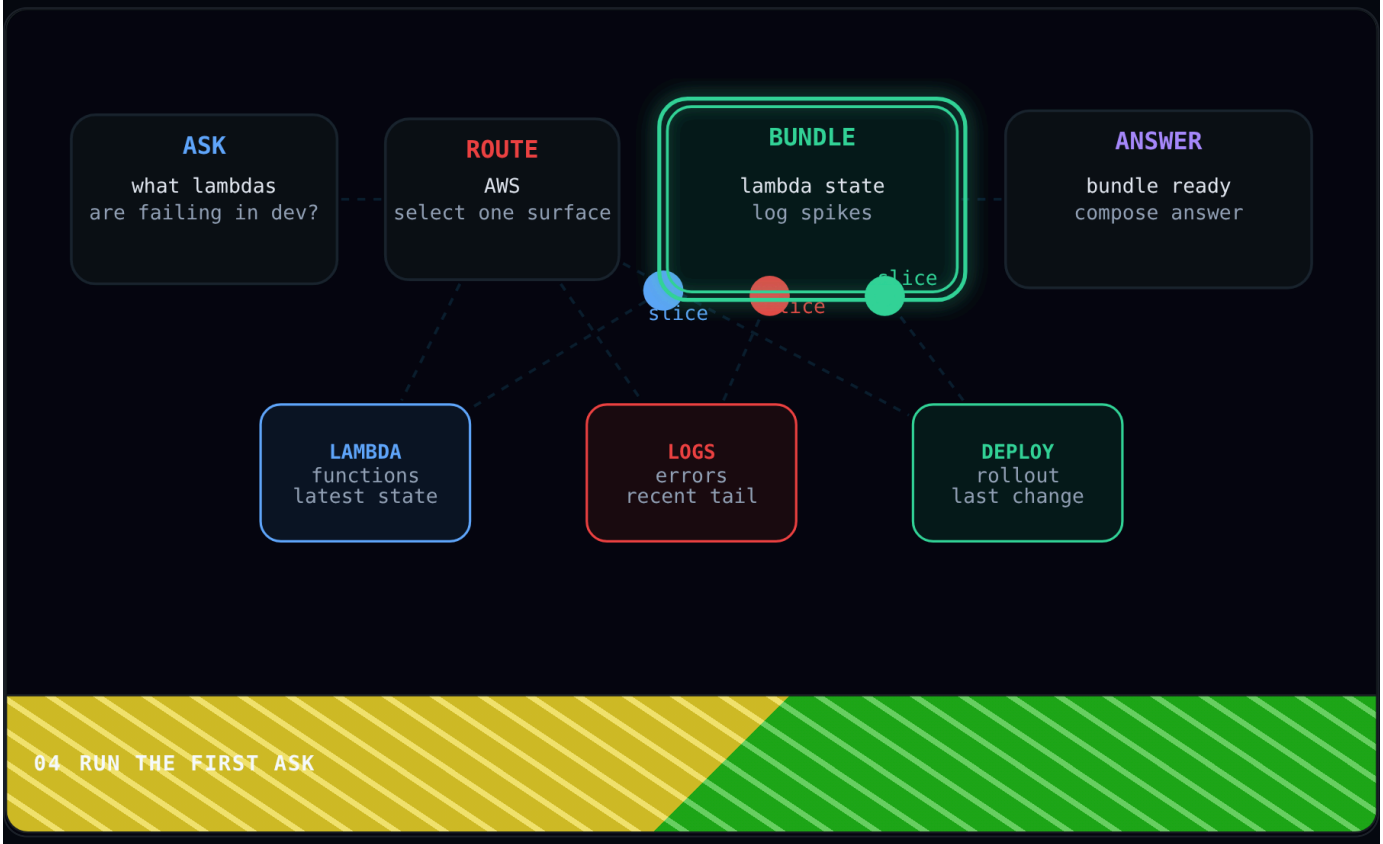
Then runtime flags like `--aws`, `--profile`, or `--ai-profile` override that baseline for this specific command.

The command does not proceed until provider, credentials source, model, and backend endpoint are all resolved.

Only after that context is pinned down does the question move into routing and evidence collection.

STAGE 4 OF 7

Run the First ask



04 RUN THE FIRST ASK

Your first useful run should be one concrete natural-language question, not a pile of hand-written provider commands.

Clanker reads the question and classifies which surface it belongs to before touching any live clients.

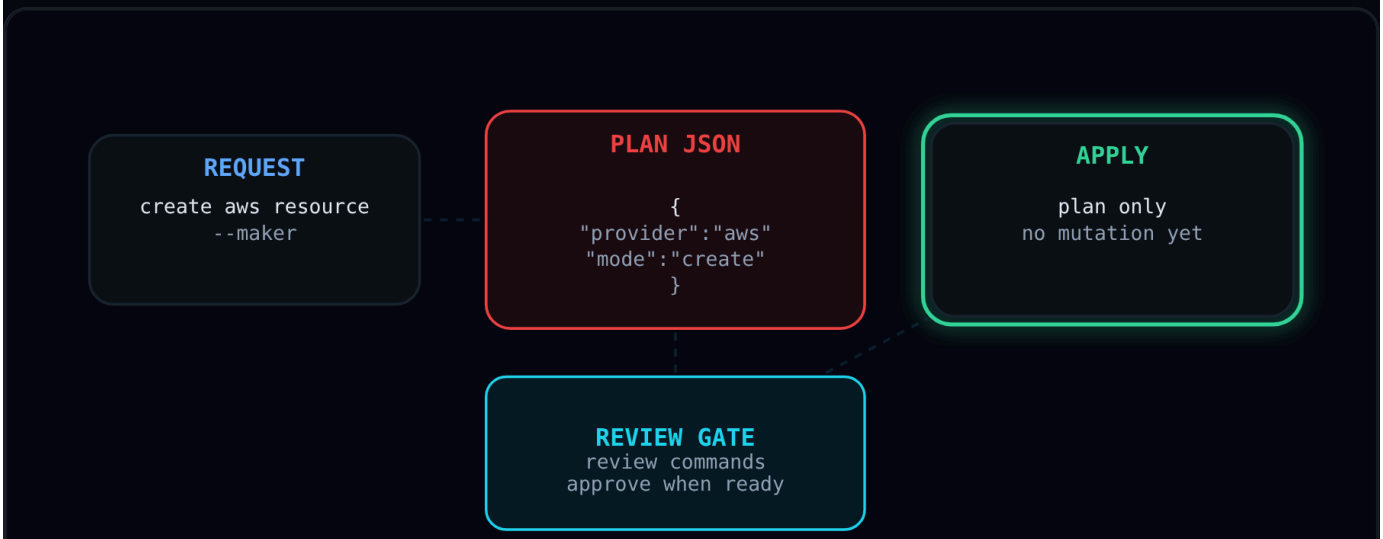
That routing step matters because only the matching probes run. Unrelated AWS, GitHub, or K8s calls stay out of the path.

The returned fragments are compacted into grounded evidence rather than dumped raw into the terminal.

What you finally see is a typed answer, or route JSON if you asked for `--route-only`.

STAGE 5 OF 7

Turn Prompts into Plans



05 TURN PROMPTS INTO PLANS

Read-only answers are the default, but Clanker can also turn a prompt into an execution plan.

Adding `--maker` changes the first output from prose into structured plan JSON for the target provider.

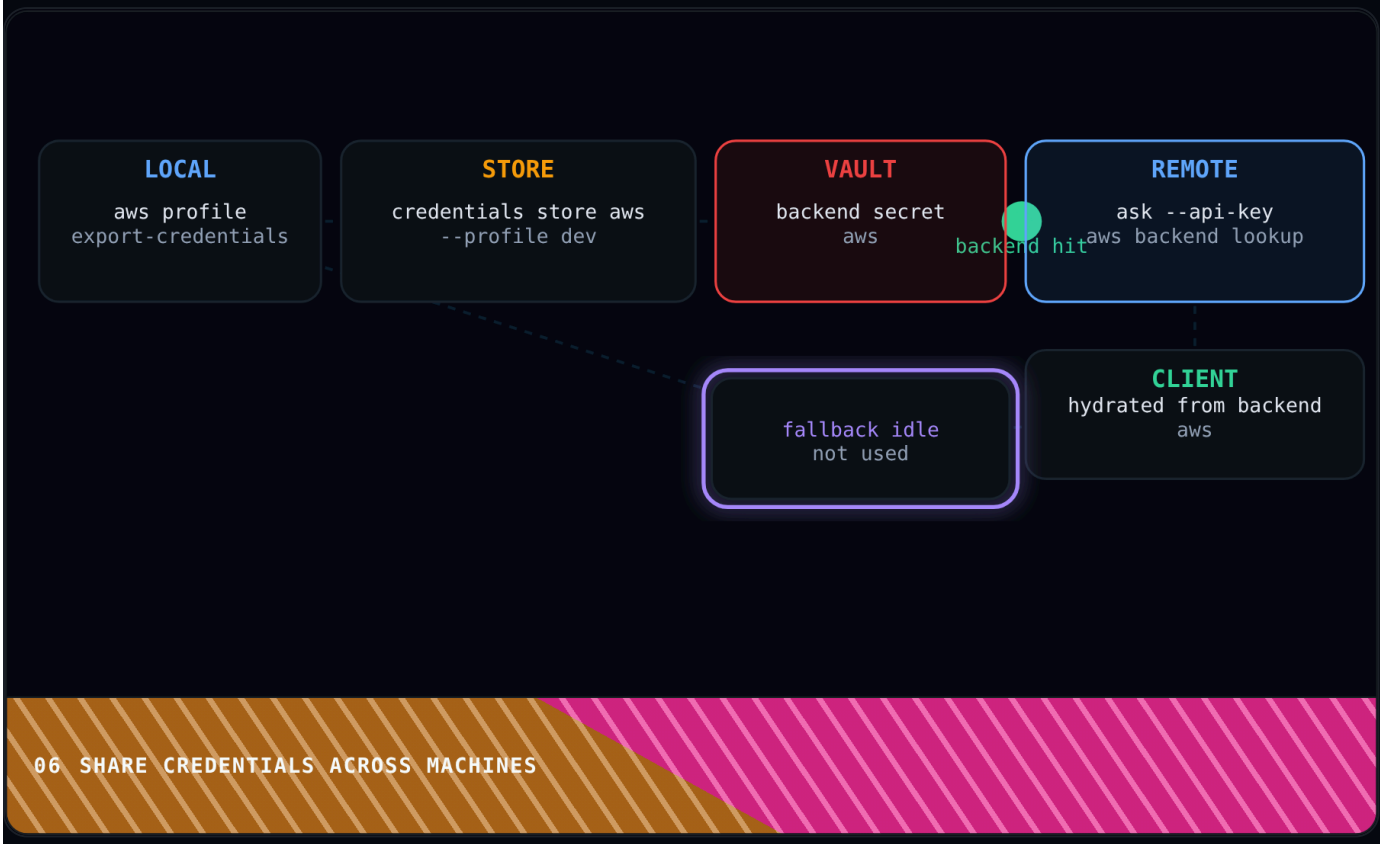
That plan is meant to be reviewed, because the safe workflow is prompt, then plan, then approval.

Only after review do you feed the plan back through `--apply`, either from stdin or a saved plan file.

If the request is destructive, Clanker keeps it behind `--destroyer` plus explicit confirmation.

STAGE 6 OF 7

Share Credentials Across Machines



06 SHARE CREDENTIALS ACROSS MACHINES

Now consider the cross-machine case, where the current laptop does not have the provider profile you need.

The `credentials store` flow uploads a provider credential set from the local machine into the backend vault.

That vault ties the secret to your Clanker API key so a different machine can request the same provider context.

A remote `clanker ask --api-key ...` run checks the backend first, then falls back locally if the secret is missing.

Either way, the stage ends with a hydrated provider client that can gather real evidence for the answer.

STAGE 7 OF 7

Use talk or MCP

07 USE TALK OR MCP

Once the core CLI works, you can expose the same engine through either a chat loop or an MCP server.

Use `clanker talk` when you want an interactive terminal conversation instead of one-shot commands.

Use `clanker mcp` when another client needs tool access over HTTP or stdio.

That MCP surface publishes a small toolset, including version lookup, route selection, and command execution.

At that point, Clanker is no longer just a CLI. It can sit inside a terminal chat, another agent, or a larger toolchain.