Google Antigravity & Agents CLI by Michaël Bettan

Google Antigravity is Google's agent-first development platform. You hand an agent a task, review the **artifacts** it produces (plans, diffs, walkthroughs, browser recordings), and approve key milestones. You no longer have to watch every keystroke.

Agents CLI (`agents-cli` , package `google-agents-cli`) is a separate, open-source tool. It gives any coding agent (Antigravity, Claude Code, Codex, Gemini CLI) a machine-readable way to **scaffold, evaluate, deploy and publish ADK agents** on Google Cloud.

Layer	Tool	Role
Developer harness	Antigravity (App, CLI `agy` , IDE, SDK)	The coding agent you work with
Agent-building expertise	Agents CLI skills + `agents-cli` commands	Teach the coding agent ADK, evals and deployment
Agent framework	ADK (open source)	The code of the agent you ship
Production runtime	Agent Runtime, Cloud Run, GKE → Gemini Enterprise	Where the shipped agent runs and is distributed
Secure dev environment	Cloud Workstations, Agent Sandbox on GKE, Antigravity terminal sandbox	Where agent-run code is isolated

Rule 0: Antigravity is the **developer's** harness, not the production runtime of the agent you build. The agent you ship runs on Agent Runtime, Cloud Run or GKE.

01. Surfaces: App, CLI, IDE & SDK

Surface (status)	Install path	Auth & permission model	Quotas, cost drivers & operational traps	Best for
Antigravity App (`Antigravity 2.0` , GA)	Desktop installer at `antigravity.google/download`	Shared Google sign-in across local surfaces; Google Cloud ADC when invoking Cloud CLIs/MCPs	Bound by workstation CPU/RAM and Gemini / Agent Platform RPM & TPM quotas. **Trap:** requires an interactive GUI session; not suited for unattended CI runners unless paired with CLI or SDK	Supervising parallel subagents, Automations (scheduled runs), worktrees, and artifact review panes across projects
Antigravity CLI (`agy` , GA)	`curl -fsSL https://antigravity.google/cli/install.sh | bash` (settings in `~/.gemini/antigravity-cli/settings.json`)	Shared Google sign-in, `GEMINI_API_KEY` , or Google Cloud ADC (`gcloud auth application-default login`); tool rules enforce **`Deny > Ask > Allow`**	Bound by local or Cloud Workstation VM sizing plus Gemini / Agent Platform RPM & TPM quotas. **Trap: Gemini CLI (`geminicli.com`) was superseded by `agy` on `2026-06-18` ;** treat Gemini CLI as legacy, not an active peer	Keyboard-first TUI (`agy --help` , `/help`), SSH sessions on Cloud Workstations, and OS-sandboxed terminal loops
Antigravity IDE (GA)	`antigravity.google/download` (VS Code-based standalone IDE + editor extensions)	Shared local Google sign-in; inherits workspace `.agents/` rules and permissions	Local workstation memory/CPU plus model token/RPM quotas for autocomplete and side-panel turns	In-editor flow: Tab autocomplete/supercomplete, inline edits (`Cmd/Ctrl+I`), and side-panel agent assistance
Antigravity SDK (`google-antigravity` , GA)	`pip install google-antigravity`	`GEMINI_API_KEY` or Google Cloud ADC via `LocalAgentConfig(vertex=True, project=..., location=...)`	Billed on Gemini / Agent Platform input, thinking, and output tokens across parent + subagent turns. **Traps:** omitting `conversation_id` + `save_dir` makes runs ephemeral across CI stages; omitting `policies=[deny(...), allow(...)]` leaves write/shell tools unconstrained once enabled	Headless CI/CD automation, PR triage, batch migrations, scheduled pipelines, and programmatic `@hooks.*` gates

- **Shared local sign-in:** signing in on one local surface signs you in on the others.

- **Headless SDK defaults and persistence:** the SDK agent is **read-only by default**; pass `capabilities=CapabilitiesConfig()` to unlock write tools, pair `conversation_id` + `save_dir` in `LocalAgentConfig` to resume state across CI steps, and attach declarative `policies=[deny(...), allow(...)]` plus `@hooks.*` callbacks so unattended runs fail safely:

```python
from antigravity import Agent, CapabilitiesConfig, LocalAgentConfig, allow, deny, hooks
cfg = LocalAgentConfig(vertex=True, project="my-proj", location="us-central1",
                       conversation_id="ci-pr-42", save_dir="/var/lib/agy-state",
                       capabilities=CapabilitiesConfig(), policies=[deny("run_command"), allow("view_file")])
@hooks.on_tool_error
async def on_err(ctx, err): return {"retry": False, "reason": str(err)}
async with Agent(config=cfg) as agent:
    result = await agent.run("Audit diff for breaking API schema changes")
```

- **Remote Control:** drive App and CLI sessions from a browser (`/remote-control` , `--remote-control` , or `agy remote-control start`).
- **Gemini CLI retirement (`2026-06-18`):** Antigravity CLI (`agy`) superseded Gemini CLI (`geminicli.com`) on `2026-06-18` . On first launch, `agy` detects legacy Gemini CLI settings, and `agy plugin import gemini` converts legacy Gemini CLI extensions into Antigravity plugins.

02. Agent-First Workflow: Plans, Artifacts & Verification

Execution modes

- **Planning mode:** the agent researches the codebase, groups its work and produces an **implementation plan** artifact before it edits anything. Use it for multi-file changes, new features, migrations and security-sensitive work.
- **Fast mode:** the agent acts right away. Use it for renames, one-off commands and small local fixes.

`/plan` **runs a 4-step workflow:** (1) analysis and discovery, (2) a clarifying interview if requirements are unclear, (3) an implementation plan with step-by-step tasks, **verification gates** and risks, (4) your review and approval, then execution. `/grill-me` interviews you one question at a time before work starts. Combine them with `/plan /grill-me <task>` .

Artifacts are structured deliverables: implementation plans, code diffs, diagrams, images and browser recordings. A **walkthrough** artifact is produced at the end. It summarizes the changes and often includes screenshots or recordings of browser tasks.

Steering: comment inline on a plan (like a doc). Then click **Proceed**, or click **Review** to send your comments as feedback.

Artifact review policy

Surface	Options	Default / guidance
App	Request review · Always proceed	Request review is recommended
CLI (`artifact ReviewPolicy`)	`asks-for-review` · `agent-decides` · `always-proceed`	`asks-for-review` is the default. `agent-decides` currently behaves the same way

The artifact review policy covers plans. File edits are controlled separately, by the **agent mode** (see §07).

Code refactoring, runtime optimization & vulnerability patching

- **Multi-file refactoring & runtime optimization:** use **Planning mode** (`/plan`) to inventory call sites across the repository, establish a baseline (existing test suite or latency/memory benchmark), apply incremental diffs, and run verification gates after each step before producing the final walkthrough artifact. The full playbook follows below.
- **Automated vulnerability patching:** pair security scanner findings (SAST or dependency audit) with a **security implementation plan** and a **Proof-of-Concept (PoC)** reproduction test in an isolated sandbox (the Antigravity terminal sandbox: Seatbelt profiles on macOS, kernel namespaces on Linux). The agent confirms the PoC reproduces the flaw before the fix, patches the root cause (e.g., parameterized queries, output encoding, path canonicalization, upgraded dependency), verifies that the PoC and regression tests pass, and records a security audit report artifact. Use `PreToolUse` / `PostToolUse` hooks to run linters or security scanners automatically on every edit.

Refactoring and runtime optimization playbook

Refactoring and runtime optimization are judged by different proofs than a security patch, so they need their own gates.

Aspect	Multi-file refactoring	Runtime optimization	Vulnerability patching (CodeMender-style)
Goal	Change structure, keep external behavior identical	Cut latency, cost or resource use against a measured baseline	Remove one verified flaw at its root cause
Scope	Wide: many modules and call sites	Hot paths found in traces or profiles	Narrow: the code path behind one finding
Proof of success	Unchanged test results plus a clean interface diff	Better benchmark or trace numbers and no eval regression	The exploit or PoC stops reproducing and regression tests pass
Typical tooling	Rules, skills, `research` subagents, `branch` worktrees, hooks	Cloud Trace, benchmarks, `agents-cli eval compare` , revision traffic split	CodeMender (`cm fix` on Agent Platform, Preview) or the security plan flow above

- **Key point:** CodeMender and the PoC flow above start from a finding and end with a small, reviewed diff. A refactor starts from a target design and can touch hundreds of files, so the safety net is the test suite and isolation, not an exploit reproducer.

Stage	Refactoring	Runtime optimization	Gate before the next stage
Contract and baseline	Write target structure and forbidden patterns as `.agents/rules/` files; package repeatable codemod steps as a skill	Record latency percentiles, token counts and Cloud Trace waterfalls before any edit	Characterization tests pass on the old code; baseline numbers are saved
Decompose	`research` subagents map imports, call graphs and public interfaces, then the plan splits the work into independent slices	The agent reads traces and ranks suspects (serial tool calls, repeated queries, uncached prompt prefixes)	You approve the implementation plan artifact
Isolated execution	One writer subagent per slice in `branch` mode (its own Git worktree), commands in the terminal sandbox	Apply one change at a time: parallel fan-out, caching, or a smaller model on simple steps	No writer shares a working tree; sandbox stays on
Automated checks	`PostToolUse` hooks format and lint each edit; a `Stop` hook refuses to finish while tests fail	Rerun the benchmark and the eval dataset after every change	Tests green, eval scores at or above baseline
Review and rollout	A fresh-context reviewer subagent checks the diff against the rules; merge through CI	Canary the new revision on a small traffic share, compare traces and quality, then promote	CI passes; canary metrics hold before 100%

Walkthrough: rename a module boundary across a monorepo

1. **Pin the contract:** add an `always_on` rule that names the new package path and bans imports of the old one; run the full suite and save the result as the baseline.
2. **Map the blast radius:** in `/plan`, have the agent delegate discovery to `research` subagents that list every importer and public symbol; the plan groups them into slices with no shared files.
3. **Fan out safely:** the main agent launches one subagent per slice, each in its own branch workspace so writers never share a working tree.
4. **Let hooks enforce quality:** the formatter runs after each edit and the `Stop` hook re-enters the loop with the failing test output until the slice is green.
5. **Review and merge:** a custom reviewer subagent whose `tools` list has only read tools audits each worktree diff; the walkthrough artifact summarizes the result before you merge.

Walkthrough: cut response latency of a deployed ADK agent

1. **Measure first:** open the agent's Cloud Trace spans and separate model time from tool and I/O time; save `agents-cli eval run` results as the quality baseline.
2. **Fix orchestration:** independent lookups that run one after another become a `ParallelAgent` fan-out.
3. **Fix tokens:** enable ADK context caching (`ContextCacheConfig`) for long, stable instructions on Gemini, or rely on prompt caching for Claude models; move simple classification steps to a smaller model.
4. **Prove both speed and quality:** rerun the benchmark, then `agents-cli eval compare` on the old and new result files.
5. **Canary:** on Agent Runtime, switch to a manual traffic split that keeps the current revision at 100% before you deploy, then move a small share to the new revision (revisions and traffic splitting are in Preview). The default is always-latest routing, which sends all traffic to a new revision as soon as it exists.

Hook setup used above (`.agents/hooks.json`; top-level keys are hook names, and `Stop` handlers sit directly under the event with no `matcher`):

```
{
  "format-on-edit": {
    "PostToolUse": [
      { "matcher": "write_to_file|replace_file_content|multi_replace_file_content",
        "hooks": [{ "type": "command", "command": "./scripts/format_changed.sh", "timeout": 20 }] }
    ]
  },
  "tests-before-stop": {
    "Stop": [
      { "type": "command", "command": "./scripts/stop_unless_green.sh", "timeout": 120 }
    ]
  }
}
```

- **Why the `Stop` hook carries the test gate:** a `PostToolUse` hook returns an empty object, so it cannot steer the agent. A `Stop` hook can print `{"decision": "continue", "reason": "..."}` and the reason is injected into the conversation, which sends the agent back to fix the failure.
- **Cost:** hooks run synchronously and block the loop (default timeout 30 seconds), so keep the `PostToolUse` script to changed files and run only the affected tests in the `Stop` script.

Anti-patterns

- **Anti-pattern (refactor and feature in one change):** behavior changes hide inside structural moves; ship the pure refactor first, gated by unchanged test results.
- **Anti-pattern (optimizing by feel):** tuning prompts or swapping models without a saved baseline and an eval comparison trades latency for silent quality loss.

- **Anti-pattern (shared working tree):** parallel writer subagents in `inherit` mode overwrite each other; use `branch` and non-overlapping slices.
- **Anti-pattern (self-grading):** asking the authoring agent to review its own diff reuses the same blind spots; use a separate reviewer subagent with a clean context.
- **Anti-pattern (sandbox off for builds):** benchmarks and build scripts written by the agent run with full host privileges outside the sandbox.
- **Anti-pattern (deploy equals rollout):** with always-latest routing, every Agent Runtime deploy is a 100% rollout; pin traffic with a manual split before deploying.

03. Rules, Skills & Workflows

Rules are persistent constraints and invariants, such as "always use zod" or "never import internal/".

File	Frontmatter	Activation
`AGENTS.md` / `GEMINI.md` (any directory)	None	Always on for that directory and below
`.agents/rules/*.md`	**Required:** `trigger`	`always_on` · `model_decision` (needs `description`) · `glob` (needs `globs`) · `manual`
Global: `~/.gemini/AGENTS.md`, `~/.gemini/GEMINI.md`, `~/.gemini/config/rules/*.md`	Modular rules need frontmatter	All projects on the machine

- Rules **add up**. When they conflict, the more specific directory rule wins. Each rule file is injected only once.
- A missing or invalid `trigger` (for example camelCase `alwaysOn`) means the rule is **silently discarded**.
- `.agents/rules/` is scanned **one level deep**. Files in nested folders are ignored unless you register them in `.agents/rules.json`.
- Budgets: each rule file is capped at **24 KB**. Always-on and global rules share a **20,000-token** budget. Rules over budget are reduced to a file-path pointer.

Skills are on-demand, multi-step procedures or runbooks.

- Layout: `skills/<name>/SKILL.md` with `name` and `description` in the frontmatter. Optional folders: `scripts/`, `references/`, `examples/`, `resources/`.
- **Progressive disclosure:** only the name and description go into the context. The full body loads when the task matches. **The description decides whether the skill ever activates.**
- Keep `SKILL.md` short. Move bulky docs into `references/`. Wrap long command sequences in `scripts/`. Include steps to validate the work.
- Rule or skill? A **rule** is a constraint. A **skill** is a workflow ("how to run DB migrations").

Workflows (legacy, IDE): Markdown prompt sequences you run as `/workflow-name`. The docs say current workflows stop working on **October 19, 2026**. Migrate them to skills with `/migrate-workflows`.

04. MCP, Hooks & Plugins

MCP servers: `mcp_config.json` (`mcpServers` map)

- Locations: global `~/.gemini/config/mcp_config.json`, workspace `.agents/mcp_config.json`, plugin `plugins/<name>/mcp_config.json`.
- **stdio** (local): `command`, `args`, `env`. **Streamable HTTP** (remote): `url` (`serverUrl` is also accepted).
- The legacy HTTP+SSE transport is **not supported**. For an SSE-only server, run a stdio bridge such as `mcp-remote`.
- Discovered tools are added to the agent's toolset. Plugin tools get a namespace prefix when needed to avoid name clashes.

Hooks: `hooks.json` (named hooks → event handlers)

Event	Fires	Typical use
`SessionStart`	Once per new session	Inject startup context
`PreToolUse` (needs `matcher`)	Before a tool runs	Gate, block, audit, or rewrite arguments
`PostToolUse` (needs `matcher`)	After a tool runs	Lint, format, analyze
`PreInvocation`	Before each model call	Inject reminders
`PostInvocation`	Immediately after each model invocation completes	Inject follow-up steps; force the loop to continue or terminate
`Stop`	When the loop ends	Block stopping until the goal is met

- `PreToolUse` returns `decision`: `allow` · `deny` · `ask` · `force_ask` (ignores cached approvals). It can also return `overwrite` (a shallow merge into the tool arguments).
- `Stop` can return `decision: "continue"` to re-enter the loop.
- Only `type: "command"` is supported. Hooks run **synchronously and block the loop**, so keep them fast. Set `"enabled": false` to turn a hook off.

Plugins are namespaced bundles: `plugins/<name>/plugin.json`, plus optional `skills/`, `rules/` (a consolidated `AGENTS.md` is recommended), `hooks.json` at the plugin root, `mcp_config.json` and `agents/`.

- Discovered plugins are on by default. Your on/off choice is stored in `config.json`, never inside the plugin.
- Install: `agy plugin install <name>@antigravity-plugins-official`, a GitHub URL or a local directory. Or use `/plugin` in the TUI.

05. Discovery & Precedence

Discovery scopes and source priority (highest → lowest). The harness walks up from the CWD (and from each read/edited file for directory rules) to the repo root before checking machine-global paths. When two customizations share the same name, the higher-ranked source shadows the lower one:

#	Source	Scope & sharing
1	Workspace project (`.agents/`, plus `.agent/`, `_agents/`, `_agent/`, and directory `AGENTS.md` / `GEMINI.md`)	Committed to Git; shared across the repository team
2	Workspace declared configs (`.agents/skills.json`, `.agents/plugins.json`, `.agents/rules.json`)	Repo-tracked manifests referencing shared folders

#	Source	Scope & sharing
3	Global discovery (`~/.gemini/config/` , `~/.gemini/AGENTS.md` , `~/.gemini/GEMINI.md`)	Local developer workstation only
4	Built-in customizations (mounted by name)	Shipped with the Antigravity binary

#	Source	Scope & sharing
5	Global declared configs	Machine-wide inherited manifests

Customization loading & scope precedence (`Workspace > Global`)

Customization	Workspace path (wins on name match)	Global path	Loading & execution behavior
Skills	`.agents/skills/<name>/SKILL.md`	`~/.gemini/config/skills/<name>/SKILL.md`	Progressive disclosure: loads only YAML `name` + `description` at startup; reads full `SKILL.md` body only when the task matches
Rules & Workflows	`.agents/rules/<name>.md` (plus directory `AGENTS.md` / `GEMINI.md`)	`~/.gemini/GEMINI.md` , `~/.gemini/config/rules/<name>.md`	Pinned `always_on` or `glob` / `model_decision` invariants that override general prompt instructions (legacy IDE workflows retire `2026-10-19` ; migrate with `/migrate-workflows`)
Hooks	`.agents/hooks.json` (or SDK `@hooks.*` decorators)	`~/.gemini/config/hooks.json`	Deterministic synchronous `PreToolUse` / `PostToolUse` / `Stop` shell commands; a `deny` decision or non-zero exit blocks execution or injects feedback
Subagents	`.agents/agents/<name>.md`	`~/.gemini/config/agents/<name>.md`	Isolated child context windows with optional `branch` (Git worktree) or `share` workspace isolation
Plugins	`.agents/plugins/<name>/`	`~/.gemini/config/plugins/<name>/`	Atomic namespaced bundle packaging skills, rules, subagents, `hooks.json` , and `mcp_config.json`
MCP Servers	`.agents/mcp_config.json`	`~/.gemini/config/mcp_config.json`	Local `stdio` or remote `StreamableHTTP` tool endpoints

- **Name collision vs. Permission conflict:** artifact/customization name collisions resolve **`Workspace > Global`** , whereas tool/MCP/command permission rules and `PreToolUse` verdicts across all scopes strictly evaluate **`Deny > Ask > Allow`** (an explicit `deny` always wins, regardless of scope).
- **`skills.json`** / **`plugins.json`** : `entries` (paths scanned one level deep, with `include_only` / `exclude`) and `inherits` (other config files). Paths can be absolute, `~/` -relative, or relative to the workspace/repo root.
- **Team pattern:** commit shared skills to `tools/agents/skills/` and register `{"entries":[{"path":"tools/agents/skills"}]}` in `.agents/skills.json` so every clone picks them up automatically.

06. Subagents, Browser Agent & Parallel Work

- The main agent calls `invoke_subagent` with a role and a prompt. Each subagent **starts with a clean context window** and does not see the parent's history. Subagents run in parallel, in the background.
- **Workspace modes:** `inherit` (same folder), `branch` (an isolated **Git worktree**), `share` (shared repo storage).
- **Built-in subagents:** `research` (codebase exploration), `browser` (only through `/browser`), `self` (a clone of the caller).
- **Custom subagents:** `.agents/agents/<name>.md` (workspace), `~/.gemini/config/agents/` (global) or `plugins/<name>/agents/` . YAML frontmatter: `name` , `description` , `tools` , `model` (`inherit` / `flash` / `pro`), `commandExecutionPolicy` (default `sandbox`), `mcpServers` , `skills` , `mainAgent` , `subagent` .
- Known issue: a misspelled tool name in `tools` can make the subagent hang.
- **Lifecycle:** Running → Idle (sent its result; wakes up with full context when it gets a message) → Killed (permanent; temporary worktrees are cleaned up).
- **CLI:** the `/agents` panel (Agent Manager) shows ID, role, state and current step. `Alt+J` jumps to the next pending approval. `Ctrl+K` approves in place.
- **SDK:** dynamic self-cloning (`CapabilitiesConfig(enable_subagents=True)`) or static `SubagentConfig` . A custom tool given to a subagent must also be in the parent's `tools` .
- **Browser subagent (`/browser`):** drives Google Chrome through the Chrome DevTools MCP server. Its context is isolated, every session is recorded as a WebM artifact, and it is hidden from the normal subagent list. It is not available in the IDE, and admins can turn it off.
- **Automations** (App) run prompts on a schedule. `/schedule` sets a timer or recurring check inside a conversation.

07. Permissions, Execution Modes & Terminal Sandbox

Permission rules: `allow` / `ask` / `deny` lists with typed matchers: `command(npm test)` , `read_file(path)` , `write_file(src/)` , `read_url` , `execute_url` , `mcp` . **Deny > Ask > Allow.**

Defaults when no rule matches: reading and writing inside the workspace is allowed. Web actions (`read_url` , `execute_url`) default to **Ask**. Everything else (commands, MCP, files outside the workspace) defaults to Ask.

CLI `agentMode` (cycle with Shift+Tab)

Mode	Behavior
`default`	Asks you to review a diff before every file write
`accept-edits`	Auto-approves file edits in the workspace. Commands still follow `toolPermission`
`auto` *(Preview)*	Auto-approves edits and runs commands **in the sandbox**. Commands that need to run outside the sandbox are auto-evaluated: safe, reversible ones run, risky ones pause

Mode	Behavior
`plan`	Plans before editing (adds `/plan` to the prompt)

CLI `toolPermission`: `request-review` (default) · `proceed-in-sandbox` · `strict` (asks before every tool call, even reads) · `always-proceed` (also turned on by `--dangerously-skip-permissions`). `strict` and `always-proceed` override `auto`.

Terminal sandbox (`enableTerminalSandbox: true` / `--sandbox`; off by default in CLI, turned on automatically in `auto` mode; see the 4-way sandbox decision matrix in §08)

- Isolates shell commands using OS-level primitives: macOS **Seatbelt** (`sbpl`) profiles and Linux `seccomp` + kernel namespaces. Project folders are mounted read-write, `read_file` paths read-only, and `write_file` paths read-write; sensitive host paths such as `~/.ssh` and `.env` are blocked and invisible.
- Network: in the CLI, sandboxed commands get **no network by default**. Domains allowed under `read_url` are added to the outbound allowlist.
- App security presets (v2.23.0+): individual developer defaults enable the terminal sandbox automatically on `Default` and `Auto` presets, while `Request Review` and `Turbo` leave the sandbox off. In Enterprise managed profiles, presets (`Default`, `Full machine`, `Turbo`) leave the terminal sandbox off by default so organizations can configure and mandate sandboxing via centralized policy. A command running outside the sandbox has full host privileges.

Other settings: non-workspace file access (off by default), internet access policy, command allow/deny lists, browser domain allowlist. Project settings override global settings.

08. Secure Dev Environments: Cloud Workstations & GKE

Cloud Workstations (GA): managed Compute Engine dev environments on Google Cloud.

- **Cluster:** region + VPC (not related to GKE clusters). **Configuration:** the template (VM type, CMEK-encrypted persistent `/home` disk, custom container image, IDE, idle and running timeouts; access through IAM). **Workstation:** an instance started or stopped on demand.
- Access through a browser IDE, local VS Code or JetBrains IDEs, or SSH. You can bring your own Linux container image, as long as its entrypoint runs a blocking process.
- **Lock it down:** place inside a VPC Service Controls perimeter, use a **private cluster** (clients connect through Private Service Connect), **turn off public IPs** (org policy `constraints/compute.vmExternalIpAccess`), and restrict the Compute Engine API alongside the Cloud Workstations API.
- **Agent-optimized development:** set `idleAction=SUSPEND` with an `idleTimeout` (for example `1800s`). Add **keep-alive lifecycle hooks**: start `/google/scripts/keep_alive.sh` when the agent starts working and kill it when the agent waits for input. Samples for Claude Code and Antigravity CLI ship in `/google/samples/agents/keepalive/`. A suspended workstation preserves RAM, running processes, and live agent context while stopping vCPU and memory charges.

Agent Sandbox on GKE (GA): a managed GKE isolation pattern providing pod-level user-space kernel (`gVisor`) or micro-VM (Kata) sandboxing for untrusted, LLM-generated code and custom tool sidecars. Pieces: `SandboxTemplate` (the blueprint) and `SandboxWarmPool` (pre-warmed pods). A non-overridable core security policy enforces `gVisor` and blocks `hostNetwork` and `hostPath`.

Agent Runtime Managed Sandboxes: serverless execution environments invoked as tools by deployed runtime agents without cluster operations: **Code Execution** (**GA**, stateful Python/pandas sandbox retaining state up to **14 days**, with **VPC-SC** and **CMEK** support) and **Computer Use** (**Preview**, isolated headless browser/UI automation).

4-way sandbox decision matrix

Sandbox option (stage)	Isolation boundary & workload fit	Secrets, VPC-SC & egress	Approver	Primary operational traps
GKE Agent Sandbox (`gVisor` / Kata, `GA`)	Pod-level user-space kernel (`gVisor`) or micro-VM (Kata) for untrusted/multi-tenant generated code, custom sidecars, and high-scale tool execution	**GKE Workload Identity Federation** + **Secret Manager CSI Driver** (ephemeral mounts, zero JSON keys); VPC-SC + Kubernetes `NetworkPolicy` + PSC/Cloud NAT egress	Platform Engineering / Kubernetes SecOps	Blocking the Compute Engine metadata server in pod `NetworkPolicy` without configuring GKE Workload Identity Federation breaks ADC token fetches
Cloud Workstations (`GA`)	Managed VM + container for human-supervised coding agents (`agy`, Claude Code) needing **zero source code on laptops** and CMEK persistent `/home` disks	Workstation VM service account via metadata server + Secret Manager; **VPC-SC** private cluster, **PSC** client ingress, no public IP (`vmExternalIpAccess`), PGA/NAT egress	Enterprise IT / IAM & Workstation Admin	Enforcing `constraints/compute.restrictPrivateServiceConnectProducer` without allowlisting Google APIs breaks workstation PSC setup; stopping instead of `SUSPEND` drops live RAM state
Antigravity CLI Local Sandbox (`enableTerminalSandbox: true`, `GA`)	Fast, $0 cloud-cost local OS process isolation via macOS **Seatbelt** (`sbpl`) or Linux `seccomp`/namespaces	Blocks `~/.ssh` and `.env`; outbound network blocked by default except `read_url` domains; **no VPC-SC** (repo still cloned onto laptop)	Individual developer (or Enterprise profile policy admin)	Shares the host OS kernel (not a VM boundary against kernel exploits); host-daemon commands (`docker build`) fail unless allowlisted unsandboxed

Sandbox option (stage)	Isolation boundary & workload fit	Secrets, VPC-SC & egress	Approver	Primary operational traps
Agent Runtime Managed Sandboxes (Code Execution `GA` ; Computer Use `Preview`)	Zero-ops runtime tool sandboxes: **14-day stateful** Python/pandas Code Execution (`GA`) or headless browser UI Computer Use (`Preview`)	Calling agent's **Agent Identity** / attached SA + **Auth Manager** / Secret Manager outside sandbox; Code Execution supports **VPC-SC** & **CMEK** (PSC-I on Agent Runtime)	AI Platform / Application Security team	Never run the core agent orchestration loop *inside* the code sandbox; run orchestration on Agent Runtime and invoke the sandbox strictly as a tool

09. Agents CLI: Build → Evaluate → Deploy → Publish

What it is: a single, machine-readable interface that coding agents use to drive ADK projects end to end. It packages expert knowledge of ADK, evaluation and Google Cloud deployment as **skills**. Source: `github.com/google/agents-cli`.

Positioning: which tool for which job

Job to be done	Primary tool	Role, boundaries & CLI caveats
Local or Cloud Workstation interactive repo coding & refactoring	**Antigravity App / IDE** or **Antigravity CLI (`agy`)**	Interactive developer harness with `/plan` artifacts, parallel `branch` worktree subagents, and OS sandboxing; reads Agents CLI skills to build agents
Headless CI/CD PR triage, automated codegen, or batch repo migration	**Antigravity SDK (`google-antigravity`)**	Programmatic Python harness (`LocalAgentConfig` , `@hooks.*` , `policies=[deny(...), allow(...)]`) with `conversation_id` + `save_dir` state persistence
Scaffold, locally run, evaluate, and deploy an ADK agent	**Agents CLI (`agents-cli` / `google-agents-cli`)** + **ADK CLI (`adk`)**	`adk` handles inner-loop framework learning (`adk create` , `adk run`); `agents-cli` adds production scaffolding, synthetic/rubric evals, Terraform infra, and deployment to Agent Runtime, Cloud Run, or GKE
Publish a custom agent to Gemini Enterprise	**Agents CLI** (`agents-cli publish gemini`) or Cloud Console	Canonical command is **`agents-cli publish gemini`** (`--registration-type adk` or `a2a`); `publish gemini-enterprise` remains only as a hidden legacy alias
Govern fleet identity, tool access, and guardrails at scale	**Agent Gateway** + **Agent Registry** + **Model Armor**	Production runtime control plane enforcing SPIFFE Agent Identity, UAP (`roles/iap.egressor`), Semantic Governance, and inline prompt/tool sanitization
One-off Google Cloud API, IAM, or network administration	**`gcloud` CLI** / Terraform	Direct resource ops; when `agents-cli` invokes Cloud tooling, it suppresses interactive prompts (`CLOUDSDK_CORE_DISABLE_PROMPTS=1`) unless `-i` is passed
Legacy Gemini CLI (`geminicli.com`) workflows	Migrate to **Antigravity CLI (`agy`)**	Gemini CLI / Code Assist retired for individual, AI Pro, and AI Ultra tiers on `2026-06-18` (new Standard/Enterprise Code Assist sales ended `2026-10-09`); run `agy plugin import gemini` to migrate legacy extensions into Antigravity plugins

- **Rule of thumb:** use `agy` / Antigravity App to write code interactively, `google-antigravity` SDK for headless CI repo automation, `adk` to prototype framework primitives, `agents-cli` to evaluate/deploy/publish ADK agents, and Agent Gateway + Registry + Model Armor to govern deployed fleets.

Install. This is the only command you run yourself:

```
uvx google-agents-cli setup  # installs agents-cli, ADK packages and skills into the coding agents on this machine
# alternatives: pipx/pip install google-agents-cli && agents-cli setup
# skills only:  npx skills add google/agents-cli
```

Prerequisites: Python 3.11+, `uv` , Node.js (for the skills), a coding agent. For deployment you also need the gcloud CLI and Terraform.

Bundled skills

Installed into your coding agent by `agents-cli setup` (refresh them with `agents-cli update`):

Skill	What the coding agent learns
`google-agents-cli-workflow`	Always active: the end-to-end flow (scaffold, build, evaluate, deploy, publish, observe), rules for preserving existing code, model choice
`google-agents-cli-scaffold`	`scaffold create` , `scaffold enhance` , `scaffold upgrade` , templates, deployment targets, prototype-first
`google-agents-cli-adk-code`	ADK API: agent types, tools, orchestration, callbacks, state
`google-agents-cli-eval`	Eval datasets, metrics, LLM-as-judge, comparing runs, failure clusters, prompt optimization
`google-agents-cli-deploy`	Agent Runtime, Cloud Run, GKE, CI/CD, secrets, service accounts, rollback
`google-agents-cli-publish`	Gemini registration: ADK vs. A2A registration types, flags, auto-detection from deployment metadata

Skill	What the coding agent learns
`google-agents-cli-observability`	Cloud Trace, prompt-response logging, BigQuery Agent Analytics, third-party tracing tools

Installing and scoping skills

Goal	Command	Effect
CLI plus skills, machine-wide	`uvx google-agents-cli setup`	Installs the tool and pushes skills to every detected coding agent (global scope is the default)
Skills for this repo only	`agents-cli setup --workspace`	Installs skills relative to the current directory, so teammates get them through Git
Target specific agents	`agents-cli setup --agent claude-code`	Repeatable flag; `--agent all` covers every supported agent
Preview first	`agents-cli setup --dry-run`	Shows what would change without writing anything
Skills only, no CLI	`npx skills add google/agents-cli`	Copies the skill folders; the coding agent installs the CLI when it needs it

- **Extensions (experimental):** `agents-cli extension add <ref>` adds third-party skills and command overrides. Project scope (default) records the pin in `agents-cli-extensions.yaml` and copies skills to `<project>/.agents/skills/`; `--global` uses `~/.agents/skills/`. When both scopes define the same command, project scope wins.
- **Trust:** any `org/repo` reference prompts before install because its commands run arbitrary code. `--yes` skips that prompt, so keep it for automation you control.
- **Claude Code caveat:** Claude Code reads `.claude/skills/`, not `.agents/skills/`, so link or copy extension skills there yourself.

Relation to Skill Registry

Aspect	Agents CLI skills	Skill Registry (Agent Platform, Preview)
Where skills live	`SKILL.md` folders on disk, inside each coding agent or repo	A private, managed Google Cloud repository
Who loads them	The developer's coding agent (Antigravity, Claude Code, Codex)	Agents that discover and load skills dynamically based on user intent
Versioning	Git, or the commit pin of an extension	API resources: a mutable **skill** plus immutable **skill revisions**
Built-in helper	None needed	The `gcp-skill-registry` skill lets an agent create, search and manage registry skills

- **Key point:** both use the `SKILL.md` folder format, so a skill written for a coding agent can later be packaged for the registry. Standalone skills can also be governed centrally in Agent Registry.

Lifecycle commands

The coding agent usually runs these for you; every one also works by hand.

Stage	Command	Purpose and notable flags
Setup	`agents-cli setup`	Install the CLI and skills (`--workspace`, `--agent`, `--skip-auth`, `--dry-run`)
Auth	`agents-cli login`	Sign in to Google Cloud or AI Studio; `--status` shows the current state
Scaffold	`agents-cli create <name>`	Alias of `scaffold create`. `-y` uses defaults (`adk_python` template, `agent_runtime` target); `-p` / `--prototype` skips CI/CD and Terraform; `-d` sets the target
Scaffold	`agents-cli scaffold enhance`	Add a target (`-d agent_runtime`, `cloud_run` or `gke`) or a `--cicd-runner` (`github_actions`, `google_cloud_build`) to an existing project; `--dry-run` previews
Scaffold	`agents-cli scaffold upgrade`	Move a project to a newer Agents CLI template version
Develop	`agents-cli install`, `agents-cli run "<prompt>"`	Install dependencies; send one prompt to the agent (non-interactive)
Develop	`agents-cli playground`	ADK web UI with hot reload, default `--port 8080`
Develop	`agents-cli lint`	Code quality checks (Ruff)
Evaluate	`agents-cli eval dataset synthesize`	Generate multi-turn eval scenarios for the local agent
Evaluate	`agents-cli eval run`	Run inference over the eval dataset and grade the traces (`eval generate` and `eval grade` split the two halves)
Evaluate	`agents-cli eval analyze`	Cluster failure modes from grade results
Evaluate	`agents-cli eval compare`	Compare two eval result files (regression check)
Evaluate	`agents-cli eval optimize`	Tune agent prompts from eval data
Deploy	`agents-cli deploy`	Deploy to Agent Runtime, Cloud Run or GKE. Useful flags: `--dry-run`, `--no-wait` then `--status`, `--update-only`, `--agent-identity` (Agent Runtime only)
Infra	`agents-cli infra single-project`	Terraform for one project (dedicated service account, secrets, IAM). Plans by default; `--apply` applies. Optional, because `deploy` works without it

Stage	Command	Purpose and notable flags
Infra	`agents-cli infra cicd`	CI/CD pipeline plus staging and prod infrastructure
Publish	`agents-cli publish gemini`	Register the deployed agent with a Gemini Enterprise app (`--gemini-app-id` , `--agent-runtime-id` , `--registration-type adk` or `a2a`). `publish gemini-enterprise` is the older name, kept as a hidden alias
Inspect	`agents-cli info`	Project config, paths, CLI version and active extensions; `--json` for machine output

> **NOTE**
> Command names and flags come from the `google/agents-cli` README, docs and CLI source at writing time. The tool ships often: confirm with `agents-cli <command> --help` before relying on a flag.

```
agents-cli create support-agent --prototype -y  # scaffold with defaults, no CI/CD
agents-cli run "Where is my order 1234?"          # smoke test, one prompt
agents-cli eval run                               # release gate
agents-cli scaffold enhance -d cloud_run          # add a deploy target later
agents-cli deploy --dry-run  # preview, then drop --dry-run
agents-cli publish gemini --gemini-app-id "$GEMINI_APP_ID"
```

Project layout: `app/agent.py` , `app/fast_api_app.py` , `tests/{eval,integration,unit}` , `agents-cli-manifest.yaml` , `Dockerfile` , `GEMINI.md` (guidance for coding agents) and `.env` .

Eval files: the default dataset is `tests/eval/datasets/basic-dataset.json` and metrics live in `tests/eval/eval_config.yaml` . Older projects with `tests/eval/evalsets/*.evalset.json` must migrate to the new dataset format. Pass `--dataset` and `--metrics` to `eval run` to target other files; `eval compare baseline.json candidate.json` diffs two result files.

Auth: a Gemini API key in `.env` (comment out the `GOOGLE_CLOUD_*` lines), **or** Application Default Credentials (`gcloud auth application-default login`) with the `GOOGLE_CLOUD_*` lines set.

Observability: Cloud Trace is on by default. Asking the agent to "set up observability" provisions a service account, a Cloud Storage bucket and a BigQuery dataset.

Templates: `--agent adk_python` (alias `adk` , the `-y` default) or `--agent adk_go` ; `--agent` also accepts a template repository (`org/repo` , optionally with `--ref`) or `local@<path>` ; confirm the exact syntax with `agents-cli create --help` . RAG is no longer a built-in template (it is now a clone-and-study recipe). For a single-file learning agent, use `adk create` instead.

Agent vs. Human mode

Agents CLI serves two callers with the same commands and syntax. *There is no documented `--mode agent` or `--mode human` flag.* The difference is how prompts and output are handled.

Dimension	Agent mode (coding agent or CI)	Human mode (developer at a terminal)
Caller	Antigravity or another coding agent following the `google-agents-cli-*` skills; CI runners	You, typing commands directly
Prompts	`-y` / `--yes` (alias `--auto-approve`) skips prompts and fills missing values with defaults; gcloud prompts stay suppressed	`-i` / `--interactive` shows menus and prompts (for example `create -i` , or `publish gemini -i` , which lists your Gemini apps)
Inputs	Explicit flags or env vars (`publish gemini` reads `GEMINI_APP_ID` and `AGENT_RUNTIME_ID`)	Interactive answers, plus the `playground` browser UI
Output	Single-shot commands (`run "<prompt>"`) and `info --json` for parsing	Console output and local URLs meant for reading
Best for	Build-eval-deploy loops, CI/CD, repeatable releases	First login, exploration, debugging in `playground`

- **Trap:** `agents-cli run --mode` exists, but it selects the protocol used to talk to the agent (`adk` , `a2a` , `adk_live`). It has nothing to do with agent vs. human mode.
- **Anti-pattern (interactive flags in agent loops):** passing `-i` from a coding agent or CI job leaves the command waiting on a prompt nobody answers.

Example workflows

Build and ship a new agent with Antigravity

1. **Brief:** ask Antigravity to "use agents-cli to build a support agent for order lookups". The `workflow` and `scaffold` skills activate, and Antigravity asks about tools, inputs, outputs and success criteria.
2. **Scaffold:** Antigravity runs `agents-cli create support-agent --prototype -y` , then writes `app/agent.py` with the `adk-code` skill.
3. **Check locally:** `agents-cli run "<prompt>"` and `agents-cli lint` ; you open `agents-cli playground` to try it yourself.
4. **Gate:** `agents-cli eval run` must pass before any cloud step.
5. **Ship:** `agents-cli deploy` to Agent Runtime, then `agents-cli publish gemini` to register the agent in your Gemini Enterprise app.

Fix a quality regression on a deployed agent

1. **Widen coverage:** `agents-cli eval dataset synthesize` adds scenarios around the reported failure.
2. **Measure:** `agents-cli eval run` , then `agents-cli eval analyze` to cluster the failures by cause.
3. **Repair:** `agents-cli eval optimize` , or patch the instructions by hand.
4. **Prove it:** re-run `eval run` and use `agents-cli eval compare` on the old and new result files to confirm nothing regressed.

5. **Roll out:** `agents-cli deploy --update-only` updates the existing Agent Runtime deployment instead of creating a duplicate.

Harden a prototype for production

1. **Add a target and pipeline:** `agents-cli scaffold enhance -d cloud_run --cicd-runner github_actions` (preview with `--dry-run`).
2. **Provision:** `agents-cli infra single-project` to plan, then rerun with `--apply`; or `agents-cli infra cicd` for a staging and prod split.
3. **Observe:** ask Antigravity to "set up observability"; the `observability` skill covers Cloud Trace, prompt-response logging and BigQuery Agent Analytics.
4. **Verify:** `agents-cli info` confirms the project config and CLI version before the first pipeline run.

10. Dev-to-Prod Lifecycle & Claude Code Comparison

End-to-end flow

1. Plan in Antigravity (`/plan`) → design spec.
2. Agents CLI skills scaffold the ADK project → `agents-cli run` / `playground`.
3. Write eval datasets → `agents-cli eval run` → iterate until they pass.
4. `agents-cli deploy` → Agent Runtime, Cloud Run or GKE.
5. Publish to Gemini Enterprise. Gemini Enterprise passes the user's email to the agent. The Agent Runtime ↔ Gemini Enterprise connection is VPC-SC compliant.
6. Observe with Cloud Trace. Govern with the Agent Platform services (see the Agent Platform guides).

Vendor-neutral harness comparison (Antigravity vs. Claude Code)

Concept	Antigravity	Claude Code
Always-on project memory	`AGENTS.md` / `GEMINI.md`	`CLAUDE.md`
Skills	`.agents/skills/<name>/SKILL.md`	`.claude/skills/<name>/SKILL.md`
Subagents	`.agents/agents/<name>.md`	`.claude/agents/<name>.md`
MCP config	`mcp_config.json`	`.mcp.json`
Hooks	`hooks.json` (`PreToolUse`, `PostToolUse`, `Stop`, ...)	`settings.json` hooks (`PreToolUse`, ...)
Permission precedence	Deny > Ask > Allow	deny → ask → allow
Agent-building helper	Agents CLI works with **both**	Agents CLI works with **both**
Surfaces	App, CLI (`agy`), IDE and Python SDK	Terminal CLI plus IDE integrations
Models	Gemini by default; the model selector also lists some Claude and GPT-OSS models (availability depends on your plan)	Claude models only; on Google Cloud they are served as Agent Platform partner models
Google Cloud auth and billing	Google sign-in; Google Cloud credentials when it drives `gcloud` or `agents-cli`	`CLAUDE_CODE_USE_VERTEX=1` plus ADC; usage is billed to your Google Cloud project, Provisioned Throughput optional
Built-in isolation	Terminal sandbox, permission rules, Git worktrees for subagents	Permission rules and Claude Code's own sandboxing settings
Hardened environment	Cloud Workstations, Agent Sandbox on GKE	Same options: Cloud Workstations, GKE Sandbox (gVisor)
Best fit	Teams that want parallel subagents, artifacts review and Gemini-first workflows	Teams standardized on Claude that still need Google Cloud IAM, residency and billing

Key point: MCP servers and SKILL.md skills carry over between harnesses. Agents CLI installs the same skills into Antigravity, Claude Code and Codex.

Claude Code on Agent Platform: model access

Claude Code on Google Cloud solves for: teams that want Anthropic's coding agent but need the traffic, identity and bill to stay inside their Google Cloud project instead of a separate Anthropic API account.

```
Laptop / Cloud Workstation / GKE pod
  Claude Code —ADC (user or SA)→ Agent Platform Claude endpoint
       │                          global | multi-region (us, eu) | region
       └─ MCP → Google remote MCP (Model Armor floor setting)
            └→ Custom MCP on Cloud Run / GKE (Agent Gateway + Model Armor)
```

1. **Enable the API:** turn on `aiplatform.googleapis.com` in the project that will carry the usage (a dedicated project keeps cost tracking and IAM simple).
2. **Enable the model:** in Model Garden, open the Claude model card and click **Enable** to accept the partner terms. This requires `roles/consumerprocurement.entitlementManager`, and the org policy must allow `cloudcommerceconsumerprocurement.googleapis.com`.

3. **Grant invoke rights:** give developers or the runtime service account `roles/aiplatform.user` (it carries `aiplatform.endpoints.predict`), or a custom role with just that permission.
4. **Point Claude Code at the project:** set the env vars below, or run `claude`, pick the third-party platform login and let the wizard write them into `~/.claude/settings.json` (`/setup-vertex` reopens it).
5. **Verify:** `/status` inside Claude Code must show Google Vertex AI as the API provider, with your project, region and model.

- **Reseller caveat:** if a prohibited reseller manages your billing account, you cannot accept the Claude terms at all; the fix is on the reseller side.

Choosing the endpoint (`CLOUD_ML_REGION`)

Endpoint	Value	Data residency	Prompt caching	Provisioned Throughput	Pick it when
Global	`global`	No: requests are served from any supported region	Yes	No	You want the best availability and fewest capacity errors, with no residency rule
Multi-region	`us`, `eu`	Yes, within the jurisdiction	Yes	No	You need residency but still want pooled capacity
Regional	for example `us-east5`	Yes, in that region	Yes	Yes, where offered	You need a single region or reserved capacity

- **CMEK:** Cloud KMS keys cannot live in the `global` location, so any customer-managed key requirement already rules out the global endpoint. Check the Agent Platform CMEK page for partner-model coverage before promising key control over prompts.
- **Per-model overrides:** with `CLOUD_ML_REGION=global`, a model that lacks global support can be routed elsewhere through its `VERTEX_REGION_CLAUDE_*` variable. A malformed region value is ignored and Claude Code falls back to `us-east5`.
- **Pin models:** set `ANTHROPIC_DEFAULT_OPUS_MODEL`, `ANTHROPIC_DEFAULT_SONNET_MODEL` and `ANTHROPIC_DEFAULT_HAIKU_MODEL` to IDs enabled in your project. Unpinned aliases follow Claude Code's built-in defaults, which may not be enabled for you and default to Opus pricing.

```
gcloud auth application-default login  # laptop only; skip on a VM or pod with an attached SA
export CLAUDE_CODE_USE_VERTEX=1  # route Claude Code to Agent Platform
export ANTHROPIC_VERTEX_PROJECT_ID=my-proj  # wins over GOOGLE_CLOUD_PROJECT and the ADC file
export CLOUD_ML_REGION=global                 # or us / eu / us-east5
export VERTEX_REGION_CLAUDE_HAIKU_4_5=us-east5 # per-model region override
# export ENABLE_PROMPT_CACHING_1H=1  # 1-hour cache TTL instead of 5 minutes
# export DISABLE_PROMPT_CACHING=1  # turn caching off (on by default)
claude  # then /status to confirm the provider
```

Claude Code on Agent Platform: cost and capacity

- **Prompt caching** is on by default in Claude Code. A coding session resends the same system prompt, tool schemas and repo context on every turn, so cached prefixes cut both latency and input cost. Caches are scoped to one Google Cloud project.
- **Default TTL:** 5 minutes. **Extended TTL:** 1 hour, requested with `ENABLE_PROMPT_CACHING_1H=1`.
- **Billing shape:** writing to the cache costs more than a normal input token (more again for the 1-hour TTL); reading from it costs far less.
- **Provisioned Throughput** reserves capacity for some partner models for a fixed fee, in the regions you choose, and is bought through sales. Reserved requests are served ahead of pay-as-you-go traffic when the system is busy. It works only on regional endpoints, not global or multi-region.
- **Quota errors (429):** on regional endpoints, check that both the primary and the background model are offered in that region, or move to `global`. Rate-limit increases go through Google Cloud quotas, not Anthropic.

> **NOTE**
> The cache multipliers on the Agent Platform pricing page at writing time: 5-minute writes cost 1.25x base input, 1-hour writes 2x, and cache reads 0.1x. Treat these as directional and recheck the pricing page before quoting them.

Claude Code on Agent Platform: keyless auth and isolation

Where Claude Code runs	Identity	Least-privilege roles	Isolation
Laptop	User ADC from `gcloud auth application-default login`	`roles/aiplatform.user` on the model project	Claude Code permission rules only; the shell sees your whole home folder
Cloud Workstations	Service account set on the workstation configuration, read through the metadata server	`roles/aiplatform.user`, plus `roles/mcp.toolUser` if it calls Google remote MCP servers	Private cluster, VPC Service Controls, no public IP (section 08)

Where Claude Code runs	Identity	Least-privilege roles	Isolation
GKE pod	Workload Identity Federation for GKE	Same roles, bound to the pod's principal	GKE Sandbox (gVisor) for agent-run builds and tests

- **No API keys:** none of these paths needs `ANTHROPIC_API_KEY`. Avoid service account key files too; use ADC from a user login or an attached identity.
- **Workstation tip:** Cloud Workstations ships keep-alive samples for Claude Code so a suspended workstation keeps the agent's session (see section 08).
- **Expiring user credentials:** Claude Code's `gcpAuthRefresh` setting runs a command (typically the ADC login) when it detects expired credentials, instead of failing mid-session.
- **Anti-pattern (keys in the repo):** a service account key in `.env` or `GOOGLE_APPLICATION_CREDENTIALS` on a laptop is a long-lived secret the agent can read and leak.

Governing MCP traffic

- **Google remote MCP servers:** enable a Model Armor floor setting with MCP sanitization (integrated service `GOOGLE_MCP_SERVER`) in the project. Every tool call and response to a supported Google MCP server is screened for prompt injection and sensitive data, whatever harness made the call.
- **Custom MCP servers:** host them on Cloud Run or GKE behind Agent Gateway, which applies access policies and Model Armor templates to agent egress traffic.
- **Residency:** some Google MCP servers route Model Armor calls across jurisdictions, so check the supported-products list if residency matters.

Claude Code on Agent Platform: organization policy guardrails

Constraint	What it governs	What to set for Claude Code
`constraints/vertexai.allowedModels`	Which Model Garden models principals can use, per action, written as `publishers/PUBLISHER/models/MODEL:ACTION` (`predict`, `deploy`, `tune`)	Allow `:predict` on every Claude model the team pins (primary and background, e.g., `publishers/anthropic/models/claude-...:predict`), plus `publishers/anthropic/models/count-tokens:predict`, because Claude token counting has its own publisher model ID
`constraints/vertexai.allowedPartnerModelFeatures`	Advanced partner-model features (`:web_search`, `:structured_outputs`). In organization projects both are blocked by default and return **`FAILED_PRECONDITION`** until explicitly allowlisted	Allow at publisher (`publishers/anthropic`), model, or `model:feature` granularity (`:structured_outputs`, `:web_search`) only for features sessions need
`constraints/gcp.resourceLocations`	Physical locations where regional/multi-region Google Cloud resources and endpoints can be created	Restrict to approved jurisdictions (`in:eu-locations`, `in:us-locations`, or specific regions) and pair with blocking the `global` endpoint when residency applies
Custom Org Policy on `aiplatform.googleapis.com/Endpoint`	Partner telemetry / prompt-sharing opt-in flags on publisher model endpoints	Enforce `action_type: DENY` when `has(resource.publisherModelConfig.dataSharingEnabledProvider)` is true to block opt-in partner data sharing
Restrict service usage	Which Google APIs can be enabled	Keep `cloudcommerceconsumerprocurement.googleapis.com` allowed so an entitlement manager can accept Claude terms
`constraints/iam.disableServiceAccountKeyCreation`	Creation of new external service account keys	Enforce it so the keyless pattern above cannot be bypassed with a JSON key

- **Allowlist semantics:** a custom allow list implicitly denies every model not listed; with no policy set or inherited, every model is allowed.
- **Hierarchy:** an explicit deny at the folder wins over an allow at the project, unless the project policy is set to override its parents.
- **Limits:** list each model individually (there is no "all third-party models" value), and stay under 500 allowed plus denied values per policy.
- **Scope:** the policy applies to every principal in the resource, not per user, and only to Model Garden models (not Model Registry).
- **Anti-pattern (allowlist without count-tokens):** the Claude model is allowed, but token-count requests are still blocked because their publisher model ID is missing.

Claude Code on Agent Platform: Developer Knowledge MCP server

The Developer Knowledge MCP server (`https://developerknowledge.googleapis.com/mcp`, remote Streamable HTTP transport) lets coding agents ground answers in current **public English Google developer documentation** through three tools: `search_documents`, `get_documents` and `answer_query`.

Auth path	How Claude Code sends it	Watch out for
API key (the path Google documents for Claude Code)	`X-Goog-Api-Key` header	Restrict the key to the Developer Knowledge API; keep it out of committed files

Auth path	How Claude Code sends it	Watch out for
ADC bearer token	`Authorization: Bearer` plus `X-Goog-User-Project`	Tokens from `print-access-token` last one hour; the quota project header needs `serviceusage.services.use` (for example `roles/serviceusage.serviceUsageConsumer`)

```
gcloud services enable developerknowledge.googleapis.com
claude mcp add google-developer-knowledge --transport http \
  https://developerknowledge.googleapis.com/mcp --header "X-Goog-Api-Key: $DK_KEY"
```

That command writes to local scope. For a team-shared, keyless setup, commit a project-scoped `.mcp.json` whose `headersHelper` script prints a fresh ADC token on every connection:

```
{
  "mcpServers": {
    "google-developer-knowledge": {
      "type": "http",
      "url": "https://developerknowledge.googleapis.com/mcp",
      "headersHelper": "/usr/local/bin/dk-headers"
    }
  }
}
```

- **Helper contract:** the command must print a JSON object of string headers (here `Authorization` and `X-Goog-User-Project`); Claude Code reruns it on reconnect, so expiring tokens do not break the session.
- **Approval:** project-scoped servers stay pending until each developer trusts the workspace and approves them; a cloned repo cannot pre-approve its own servers.
- **VPC-SC routing constraint:** `developerknowledge.googleapis.com/mcp` indexes public English docs only and does **not** support the `restricted.googleapis.com` VIP inside a VPC Service Controls perimeter; route perimeter egress via `private.googleapis.com` with Private Google Access.
- **Errors:** a 403 asking for a quota project means the `X-Goog-User-Project` header is missing; a 404 or 403 on `/mcp` usually means the API is not enabled.

Claude Code on Cloud Workstations

Layer	Setting	Why it matters for a coding agent
Cluster	Private cluster, reached through a Private Service Connect endpoint and a private DNS zone for the cluster hostname	No one can open the workstation from the public internet
Perimeter	VPC Service Controls; restrict the Compute Engine API whenever the Cloud Workstations API is restricted, and keep Cloud Storage and Artifact Registry VPC-accessible for image pulls	Source code and model traffic stay inside the perimeter
VM network	`--disable-public-ip-addresses` on the configuration, plus Private Google Access or Cloud NAT (one is required)	Private Google Access is how Claude Code still reaches the Agent Platform and MCP endpoints on `googleapis.com`
Image	Custom container image with Claude Code, gcloud, linters, managed settings and the MCP header helper baked in	Every developer starts from the same reviewed toolchain
Service account	Set on the configuration; needs Artifact Registry Reader for a private image and log and metric write permissions	It is the identity ADC resolves to inside the workstation (see the table above)
Audit	`--enable-audit-agent` turns on Linux `auditd` logging (the service account needs `logging.buckets.write`)	Records what agent-run commands did on the VM

```
gcloud workstations start-tcp-tunnel --project=PROJECT --
region=REGION \
  --cluster=CLUSTER --config=CONFIG --local-host-port=:2222
WORKSTATION 22
# then point VS Code Remote-SSH (or any SSH client) at
localhost:2222
```

- **Local IDE access:** the tunnel forwards a local port to the workstation's SSH port; it authenticates with your gcloud credentials and is authorized by the workstation's IAM policy, so nothing listens publicly. `gcloud workstations ssh` does the same in one command.
- **Private cluster caveat:** the laptop must sit on a network that reaches the PSC endpoint (for example over VPN or Interconnect), or the tunnel cannot connect.
- **Keep TCP on:** `--disable-tcp-connections` blocks plain TCP connections, so leave it unset when developers rely on the SSH tunnel.
- **Org policy interplay:** if `constraints/compute.trustedImageProjects` is enforced, allow `projects/cos-cloud`, because workstation VMs boot from Container-Optimized OS images.

Claude Code on Agent Platform: setup pitfalls

Pitfall	Symptom	Fix
Model not enabled	"Model not found" (404) even though the caller has `roles/aiplatform.user`	Enable the model in Model Garden; an entitlement manager must accept the partner terms

Pitfall	Symptom	Fix
Org policy blocks procurement	Nobody can enable the Claude model, even with the entitlement manager role	Allow `cloudcommerceconsumerprocurement.googleapis.com` in the restrict-service-usage policy
Unsupported region	404 for one model, or 429 on a regional endpoint	Pick a region listed for that model, use `global` or `us` / `eu`, or set the model's `VERTEX_REGION_CLAUDE_*` override
Wrong project billed	Calls land in an unexpected project	`ANTHROPIC_VERTEX_PROJECT_ID` takes priority over `GOOGLE_CLOUD_PROJECT` and the ADC file; set it explicitly
Expired ADC	"Could not load the default credentials" mid-session	Re-run the ADC login, set `gcpAuthRefresh`, or move to a workstation with an attached service account
Global plus residency or CMEK	Compliance review fails	Switch to a regional or `us` / `eu` endpoint and enforce `constraints/gcp.resourceLocations`; global has no residency guarantee
Expecting Provisioned Throughput on global	Reserved capacity is never used	Provisioned Throughput applies to regional endpoints only
Unrestricted shell on a laptop	The agent reads `~/.ssh`, kubeconfigs or cloud credentials, or pushes outside the perimeter	Move sessions to a VPC-SC Cloud Workstation, keep deny rules, and never run `--dangerously-skip-permissions` outside a sandbox
Model missing from the org allowlist	The model is enabled in Model Garden, yet calls to it are refused	Add `publishers/anthropic/models/MODEL:predict` (and `count-tokens:predict`) to `vertexai.allowedModels`, or check for a deny inherited from a folder
Partner feature blocked	`FAILED_PRECONDITION` when requests use `:web_search` or `:structured_outputs` in an organization project	Allow the feature in `vertexai.allowedPartnerModelFeatures` at the narrowest granularity that works
Static ADC token in MCP headers	The Developer Knowledge server starts returning auth errors about an hour into the session	Replace the pasted token with a `headersHelper` script, or use an API key
Tunnel from outside the private network	`start-tcp-tunnel` cannot reach a private-cluster workstation	Connect from a network that reaches the PSC endpoint (VPN or Interconnect); private clusters accept no public clients, browser or tunnel

11. Exam Traps & Failure Modes

Symptom	Root cause	Fix
A rule in `.agents/rules/` never applies	Missing or camelCase `trigger`, or file in a nested subfolder	Use snake_case `trigger`, keep the file flat, or register it in `rules.json`
A skill never activates	Vague `description`	Say **what** the skill does and **when** to use it
A huge always-on rule gets ignored	20k-token rules budget exceeded → reduced to a pointer	Switch to `model_decision` or a skill
A remote MCP server fails	Legacy SSE endpoint	Point to the Streamable HTTP `/mcp` endpoint, or use a stdio bridge
The agent wipes files on the host	`always-proceed` / `--dangerously-skip-permissions` without a sandbox	Sandbox + `proceed-in-sandbox` + deny rules
Headless SDK pipeline loses state across CI stages or runs unconstrained commands	`LocalAgentConfig` omitted `conversation_id` + `save_dir`, or enabled write capabilities without `policies=[deny(...), allow(...)]`	Pass `conversation_id`, `save_dir`, and explicit `policies` in `LocalAgentConfig`
Team plans a new workflow around Gemini CLI (`geminicli.com`)	Gemini CLI / Code Assist retired for individual, AI Pro, and AI Ultra tiers on `2026-06-18` (new Standard/Enterprise Code Assist sales ended `2026-10-09`)	Standardize on `agy` and migrate legacy extensions via `agy plugin import gemini`
Parallel subagents overwrite each other	Shared workspace, overlapping scope	`branch` (Git worktree) and non-overlapping scopes
The agent slows to a crawl	Slow synchronous hook	Keep hooks fast. Move heavy work out of the loop
A workstation loses agent state overnight	Idle timeout stops the VM	`idleAction=SUSPEND` + keep-alive hooks
Private Cloud Workstation fails PSC setup	`constraints/compute.restrictPrivateServiceConnectProducer` blocks Google API producer endpoints	Allowlist required Google APIs in the PSC producer org policy

Symptom	Root cause	Fix
GKE Agent Sandbox pods fail ADC token fetches	Pod `NetworkPolicy` blocked Compute Engine metadata server without workload federation	Configure **GKE Workload Identity Federation** (and Secret Manager CSI Driver for secrets)
Core ADK orchestration loop placed inside Agent Runtime Code Execution sandbox	Confusing runtime orchestration with the isolated code tool	Run the ADK orchestrator on Agent Runtime and invoke the 14-day stateful Code Execution sandbox strictly as a tool
Developer Knowledge MCP fails inside VPC-SC	Traffic routed to `restricted.googleapis.com` VIP (unsupported by `developerknowledge.googleapis.com`)	Route `developerknowledge.googleapis.com` through `private.googleapis.com` with Private Google Access
Data exfiltration risk from dev VMs	Public IPs, public cluster	Private cluster, VPC-SC, `vmExternalIpAccess` constraint
The agent shipped without evals	Deploy before evaluating	`agents-cli eval run` with an eval dataset is a release gate
A coding agent or CI job hangs on an `agents-cli` command	`-i` / `--interactive` passed in an unattended run	Use `-y` / `--yes` plus explicit flags or env vars
`agents-cli publish gemini-enterprise` missing from `--help`	Renamed to `publish gemini`; the old name is a hidden alias	Use `agents-cli publish gemini`
Treating Antigravity as the agent runtime	Mixing up the dev harness and production	Deploy the ADK agent to Agent Runtime, Cloud Run or GKE

12. Glossary

- **Artifact:** a structured deliverable (plan, diff, walkthrough, recording) that you review in place of individual tool calls.
- **Planning vs. Fast mode:** plan-first execution vs. direct execution.
- **Progressive disclosure:** only the name and description of skills (and `model_decision` rules) go into the context until they are needed.
- **Plugin:** a namespaced bundle of skills, rules, hooks, MCP servers and agents.
- **Terminal sandbox:** an OS-level boundary (macOS Seatbelt `sbpl`, Linux `seccomp` /namespaces) for agent shell commands.
- **`LocalAgentConfig`:** the Antigravity SDK config object (`vertex=True`, `conversation_id`, `save_dir`, `capabilities`, `policies`) used to persist and constrain headless CI/CD runs.
- **Agents CLI:** a skills and command toolkit that lets any coding agent build, evaluate, deploy and publish ADK agents.
- **ADK CLI (`adk`):** the framework-level CLI bundled with ADK (`adk create`, `adk run`), used for learning and inner-loop work.
- **Agents CLI extension:** an experimental, commit-pinned add-on that contributes skills and command overrides at project or `--global` scope.
- **Skill Registry:** a private Agent Platform repository (Preview) from which agents discover and load versioned skills.
- **Skill revision:** an immutable snapshot of one version of a Skill Registry skill.
- **Non-interactive mode:** running `agents-cli` with `-y` / `--yes` so missing values fall back to defaults and no prompt blocks a coding agent or CI job.
- **Agent Sandbox on GKE:** a `gVisor` /Kata-isolated Kubernetes runtime (`SandboxTemplate`, `SandboxWarmPool`) for untrusted, LLM-generated code.
- **Agent Runtime Code Execution sandbox:** a managed, VPC-SC and CMEK-compatible tool sandbox (GA) for generated Python/pandas execution that retains state for up to 14 days.
- **Computer Use sandbox:** a managed Agent Platform sandbox (Preview) for isolated headless browser and UI automation.
- **Workstation suspension:** `idleAction=SUSPEND` keeps VM memory and agent state through idle periods.
- **Partner model (Claude on Agent Platform):** an Anthropic model served by Agent Platform after you enable it in Model Garden; billing, IAM and quotas are Google Cloud's.
- **Global endpoint:** the `global` location for partner models; best availability, separate quotas, no data residency and no Provisioned Throughput.
- **Multi-region endpoint:** the `us` or `eu` location; keeps processing inside the jurisdiction while pooling capacity.
- **Prompt caching:** reuse of a repeated prompt prefix for 5 minutes by default or 1 hour on request; cache reads are billed far below normal input.
- **Provisioned Throughput:** reserved, fixed-fee model capacity in chosen regions that takes priority over pay-as-you-go traffic.
- **`CLAUDE_CODE_USE_VERTEX`:** the Claude Code env var that switches it from the Anthropic API to Agent Platform, used with `ANTHROPIC_VERTEX_PROJECT_ID` and `CLOUD_ML_REGION`.
- **Model Armor floor setting:** project-wide minimum filters; with the `GOOGLE_MCP_SERVER` integration it screens calls to Google remote MCP servers.
- **`vertexai.allowedModels`:** the list constraint that allows or denies Model Garden models per action (`predict`, `deploy`, `tune`); a custom allow list implicitly denies everything else.
- **`vertexai.allowedPartnerModelFeatures`:** the list constraint that unlocks partner-model features such as `:web_search` and `:structured_outputs` (otherwise rejected with `FAILED_PRECONDITION` in organization projects).
- **`dataSharingEnabledProvider`:** the `publisherModelConfig` attribute checked by a custom Org Policy on `aiplatform.googleapis.com/Endpoint` to block opt-in partner data sharing.

- **Developer Knowledge MCP server:** Google's remote MCP endpoint (`developerknowledge.googleapis.com/mcp`) for public English developer docs (`search_documents` , `get_documents` , `answer_query`); requires `private.googleapis.com` inside VPC-SC.
- **`headersHelper` :** a Claude Code MCP setting that runs a command at connection time and sends the JSON headers it prints, used for short-lived tokens.
- **Workstation TCP tunnel:** `gcloud workstations start-tcp-tunnel` , an IAM-authorized forward from a local port to a workstation port, used for SSH from a local IDE.
- **Characterization test:** a test that records what existing code does today, so a refactor can prove behavior did not change.
- **Revision traffic split:** an Agent Runtime setting (Preview) that sends set percentages of traffic to active revisions; the default sends all traffic to the latest revision.
- **CodeMender:** Google's code security agent on Agent Platform (Preview) that verifies a vulnerability with an exploit and proposes a validated patch (`cm fix`).