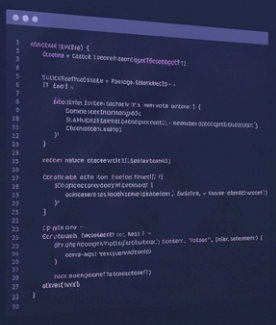
EXPANDED EDITION

ANTIGRAVITY 2.0



The Complete Field Manual for Autonomous AI Coding Agents, Local LLMs & MCP Runtimes

ANTIGRAVITY 2.0

The Complete Field Manual for AI-Powered Software Engineering, Local Multi-Agent Orchestration & MCP Extensibility

---

A Comprehensive 31-Chapter Handbook — Beginner to Advanced

Covering: Desktop App 2.0 • Standalone IDE • CLI (`agy`) • Ollama Local LLMs • Python SDK

Model Context Protocol (MCP) • Custom Skills • Lifecycle Hooks • Multi-Agent Teamwork

Audited for Technical Accuracy & System Integration Standards

Published in Toronto, Canada • 2026 Expanded Edition

Copyright & Legal Notices

Technical Auditing Standard

This handbook has been compiled and verified against the live source specifications of the Antigravity 2.0 runtime environment. All tool schemas, MCP configurations, command-line flags, YAML frontmatter syntax, lifecycle hook JSON contracts, and Python SDK classes have been validated to prevent technical hallucinations.

Disclaimer of Warranty

The instructions, code listings, and architectural patterns in this book are provided on an "as-is" basis. The authors and publishers make no warranties, express or implied, regarding the reliability or security of specific third-party integrations, local hardware setups, or model inference outputs.

Table of Contents

PART VII — OPERATIONS & REFERENCE

BONUS CHAPTERS

APPENDICES

PART I

Foundations

Understanding the architecture, philosophy, and ecosystem of Antigravity 2.0 (Chapters 1-3)

CHAPTER 1

Welcome to Agentic Software Engineering

Software engineering stands at the precipice of its most profound paradigm shift since the invention of the compiler. For over seven decades, the fundamental act of programming required humans to translate abstract mental models of logic, architecture, and business requirements into explicit, syntax-rigid lines of code. Even as high-level languages, frameworks, integrated development environments (IDEs), and continuous integration pipelines emerged to eliminate mechanical friction, the developer remained the sole engine of execution. Every file created, every test written, every bug diagnosed, and every refactor applied required direct manual keyboard input.

Google Antigravity 2.0 fundamentally breaks this constraint. It is not an incremental autocompletion widget or a conversational sidebar; it is an *AI-first agentic development platform*. In Antigravity 2.0, autonomous AI agents operate directly within your local filesystem, execution environment, and development toolchain. They read codebases, formulate multi-step execution plans, invoke compiler tools, inspect run-time logs, debug test failures, and interact with external APIs through standardized protocols—all while keeping you firmly in control as the system architect.

1.1 The Evolution: From Autocompletion to Autonomous Agents

To appreciate why agentic software engineering is fundamentally distinct from previous developer tooling, we must examine the four evolutionary waves of AI-assisted programming that led to Antigravity 2.0.

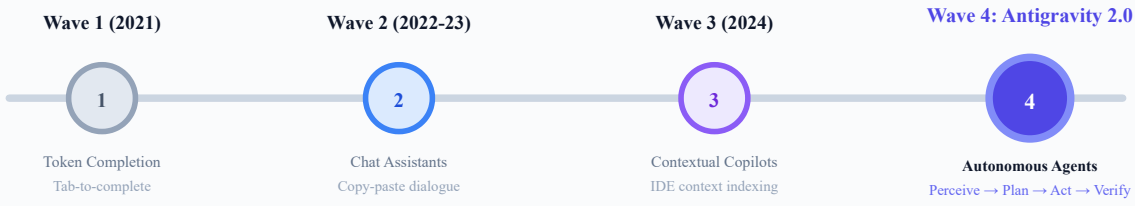


Figure 1.1: The Four Waves of AI in Software Engineering.

1. **Wave 1: Statistical Token Completion (2021–2022).** Early code models operated as advanced statistical autocomplete engines. As you typed a function signature, the model predicted the next tokens based on probability distributions learned from public repositories. While impressive for eliminating boilerplate, these models possessed zero understanding of your project architecture, could not verify syntactic correctness, and operated strictly within a microscopic context of a few lines of code.

2. **Wave 2: Conversational Chatbots (2022–2023).** Large language models (LLMs) gained chat interfaces. Developers could paste a stack trace into a browser prompt and receive an explanation or a refactored code snippet. However, this workflow suffered from high friction: the human served as a glorified clipboard courier, constantly copying code from the IDE to the web browser and pasting generated snippets back into local files. The chatbot had no access to the terminal, filesystem, or compiler.
3. **Wave 3: Context-Aware Copilots (2023–2024).** Tooling moved inside the IDE. By indexing repository files via embeddings and abstract syntax trees (ASTs), copilots could inject relevant snippets into the prompt. Yet, the interaction remained fundamentally reactive: the AI waited for user keystrokes to provide inline suggestions or answer isolated queries. If a suggestion broke the build, the AI could neither detect the error nor fix it autonomously.
4. **Wave 4: Autonomous Agentic Engineering (2025–Present).** Antigravity 2.0 represents the mature realization of the agentic wave. An agent is an active, goal-driven entity equipped with tools, persistent memory, and an execution loop. Given a high-level goal—such as *"Refactor the authentication module to support OAuth2 PKCE flow and fix all failing integration tests"*—the Antigravity agent inspects the file tree, drafts a systematic architectural plan, executes edits across dozens of files, runs the test runner in a sandboxed terminal, diagnoses regressions from the output, and iterates autonomously until all verification checks pass.

> **ARCHITECTURAL NOTE: THE AUTONOMY THRESHOLD**
>
> The defining leap of Antigravity 2.0 is **closed-loop execution**. In earlier tools, the feedback loop required human eyes at every iteration. In Antigravity 2.0, the agent executes actions, observes environmental feedback (compilation errors, linter output, unit test assertions), and corrects its own mistakes before presenting the verified solution to the developer.

1.2 What is an 'Agent' vs. a 'Chatbot'?

The industry frequently conflates "chatbots" with "agents." To build reliable, enterprise-grade software with Antigravity 2.0, you must develop a clear mental model of the distinction.

Consider an analogy from the physical world. A **Chatbot** is like an advisor sitting on your passenger seat. If you ask, *"How do I drive from Boston to New York?"*, the advisor reads aloud a turn-by-turn list of highways. But the advisor cannot touch the steering wheel, cannot press the brake pedal when an obstacle appears, cannot look at the fuel gauge, and cannot reroute when a sudden traffic jam closes Interstate 95. You, the human, must execute every turn, check every mirror, and handle every crisis.

An **Agent**, by contrast, is an advanced autonomous navigation system. It does not simply describe the route; it operates the vehicle controls within safety boundaries. It monitors the odometer (state awareness), checks weather sensors (environmental perception), engages the throttle and brakes (tool invocation),

observes road closures in real time (feedback verification), and recalculates routes autonomously when detours arise—all while respecting your destination constraints and safety rules.

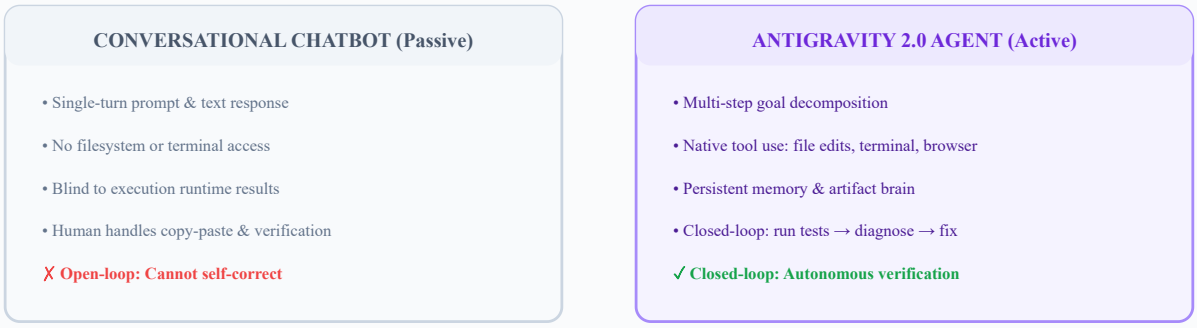


Figure 1.2: Structural Comparison between Passive Chatbots and Antigravity 2.0 Agents.

Dimension	Conversational Chatbot	Antigravity 2.0 Autonomous Agent
Operational Model	Reactive: generates text only when queried	Proactive: plans and executes multi-step workflows to achieve a stated objective
Tool Integration	None or limited to web search APIs	Native IDE tools: read/write files, execute bash/powershell, grep, AST search, MCP extensions
State & Memory	Ephemeral context window; forgets across sessions	Persistent Brain directory, structured artifacts, conversation transcript replay
Error Handling	Outputs erroneous code; user must identify bug	Runs compilers/tests, inspects stack traces, and iterates until 100% green
Multi-Agent Team	Single persona per prompt	Subagent delegation: orchestrator spawns specialized worker agents concurrently

1.3 The Vision Behind Antigravity 2.0

Antigravity 2.0 was engineered around three core architectural tenets:

1. **Hybrid Intelligence (Local + Cloud Synergy).** Modern engineering demands both maximum intelligence and uncompromising privacy. Antigravity 2.0 seamlessly routes complex architectural reasoning to cloud-scale frontier models (such as Gemini 2.0 Pro / Flash) while routing proprietary code inspection, linting, and offline operations to local models (such as Llama 3.1 or Qwen via Ollama) running locally on your hardware.

2. **Extensibility via Open Protocols (MCP).** Rather than locking developers into a walled garden, Antigravity 2.0 adopts the open *Model Context Protocol (MCP)*. Any tool—whether a database client, a GitHub integration, a Docker daemon controller, or a proprietary internal deployment CLI—can be exposed to your agent through standardized JSON-RPC protocols.

3. **Deterministic Verification & Zero-Hallucination Philosophy.** An AI's output is only as good as its verifiability. Antigravity 2.0 treats language models as hypothesis generators and deterministic tools (compilers, linters, unit test suites, browser integration runners) as the ultimate ground truth. An agent is not permitted to declare a task complete until verification assertions are satisfied.

> **TIP: THE "ANTIGRAVITY" METAPHOR**
>
> Traditional software development often feels like dragging heavy stones uphill—fighting boilerplate, writing repetitive glue code, configuring complex build chains, and hunting missing semicolons. *Antigravity* eliminates this gravitational drag, freeing engineers to focus on system design, domain logic, and creative problem-solving.

1.4 Who Should Use This Book and How to Read It

This field manual is designed for professional software engineers, technical leads, architects, and DevOps practitioners who want to transition from basic AI prompting to building and orchestrating production-grade agentic systems.

Prerequisites

- Familiarity with general programming concepts (JavaScript/TypeScript, Python, or Go).
- Basic comfort with command-line interfaces (Bash or PowerShell) and Git version control.
- No prior machine learning or prompt engineering expertise is required—all agentic concepts are explained from first principles.

Recommended Reading Paths

Your Primary Role	Primary Focus Areas	Key Outcomes
Full-Stack / Backend Developer	Parts I, II, and III (Chapters 1–9)	Master agentic refactoring, test-driven generation, local Ollama integration, and artifact workflows.
Software Architect / Tech Lead	Parts I, IV, and V (Chapters 2, 3, 10–15)	Design multi-agent orchestration topologies, configure MCP tool ecosystems, and establish team-wide agent rules.
DevOps / Platform Engineer	Chapters 3, 8, 12, and 14	Implement CLI automation with `agy`, sandboxed CI/CD agent runners, and local air-gapped LLM deployments.

1.5 What You'll Build by the End of This Book

Throughout this book, you will not just read abstract theory; you will build concrete, real-world systems. By the time you complete this manual, you will have constructed:

- **An Autonomous Microservices Orchestrator:** A multi-agent system where a parent agent coordinates specialized subagents to generate, containerize, and test microservices concurrently.
- **A Zero-Leak Air-Gapped Coding Environment:** A complete local development setup pairing Antigravity with Ollama (Llama 3.1), allowing you to build software on planes or inside secure corporate air-gaps without sending a single byte to external clouds.
- **Custom MCP Tool Suites:** Custom Model Context Protocol servers connecting Antigravity to PostgreSQL databases, Kubernetes clusters, and Chrome browser automation engines.
- **Self-Healing CI/CD Pipelines:** Automated agent hooks that capture build breakages, reproduce failures locally, write regression tests, apply targeted fixes, and submit verified pull requests.

★ **KEY TAKEAWAYS FROM CHAPTER 1**

- Agentic software engineering represents Wave 4 of AI development: transitioning from reactive text generation to proactive, tool-equipped closed-loop autonomous execution.
- Chatbots advise; Antigravity agents act. Agents formulate multi-step plans, invoke filesystem and terminal tools, and verify their work against real compilers and test suites.
- Antigravity 2.0 combines hybrid cloud/local model routing, open MCP tool extensibility, and strict deterministic verification to eliminate AI hallucinations.
- Developers transition from manual typists to strategic orchestrators—directing, reviewing, and architecting systems alongside intelligent agent teams.

CHAPTER 2

Architecture Deep Dive

To harness the full potential of Google Antigravity 2.0, an engineer must understand the mechanical engine powering its autonomy. Beneath the clean user interface lies a sophisticated, deterministic control loop orchestrating model inference, structured tool calls, context window budgeting, sandboxed execution, and security policies.

This chapter dissects the internal architecture of Antigravity 2.0, exploring how the system transforms ambiguous human intent into verified, high-performance software artifacts.

2.1 The Perceive → Plan → Act → Verify (PPAV) Loop

At the heart of every Antigravity agent sits the **PPAV Cognitive Loop**. Unlike simple prompt-response cycles, the PPAV loop is a stateful, iterative state machine that executes until a solution is verified or explicit human guidance is required.

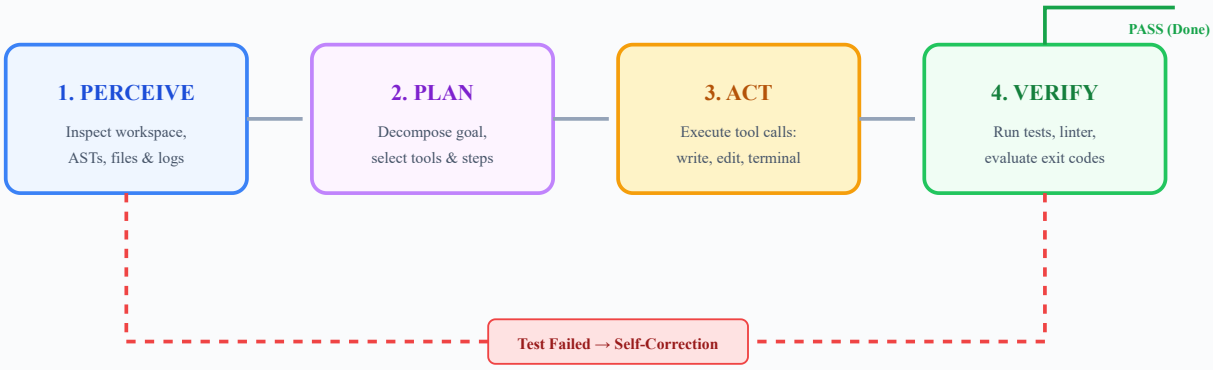


Figure 2.1: The Closed-Loop Perceive → Plan → Act → Verify (PPAV) State Machine.

Step 1: Perceive (Environmental Ingestion)

When an agent receives a prompt, it does not immediately guess a response. It first perceives its environment. This involves reading the workspace directory structure, analyzing active git branches, examining `.agentrules`, searching for relevant symbols via ripgrep/AST indexers, and inspecting error logs or open browser pages. Perception ensures the model grounds its thinking in verified project reality rather than hallucinated assumptions.

Step 2: Plan (Hierarchical Task Decomposition)

Armed with environmental context, the agent formulates a plan. For trivial tasks, this is an implicit step. For complex engineering requests (e.g., migrating a database schema across 15 models), the agent generates a persistent Markdown artifact plan. It breaks the objective into atomic, verifiable subtasks, establishing explicit dependencies and checkpoints.

Step 3: Act (Structured Tool Invocation)

The agent translates the current plan step into one or more concrete tool executions. The model emits structured JSON schema calls containing exact arguments (e.g., `replace_file_content` with precise line numbers and string matches, or `run_command` with specific CLI flags). The Antigravity tool execution engine captures these calls, enforces security checks, runs the actions, and captures stdout, stderr, and return codes.

Step 4: Verify (Deterministic Assertion)

After executing an action, the agent enters the verification phase. Did the file edit introduce syntax errors? Did the TypeScript compiler (`tsc`) emit warnings? Did unit tests pass? If verification succeeds, the agent advances to the next step in its plan. If verification fails (e.g., exit code 1 or failing assertions), the failure is fed directly into the next *Perceive* step as diagnostic input, triggering autonomous self-correction without bothering the human developer.

2.2 Model Routing: Cloud vs. Local (Ollama)

Antigravity 2.0 does not force a false binary choice between cloud-scale power and local privacy. Instead, it introduces an intelligent **Model Router** that dynamically routes subtasks to the optimal inference engine.

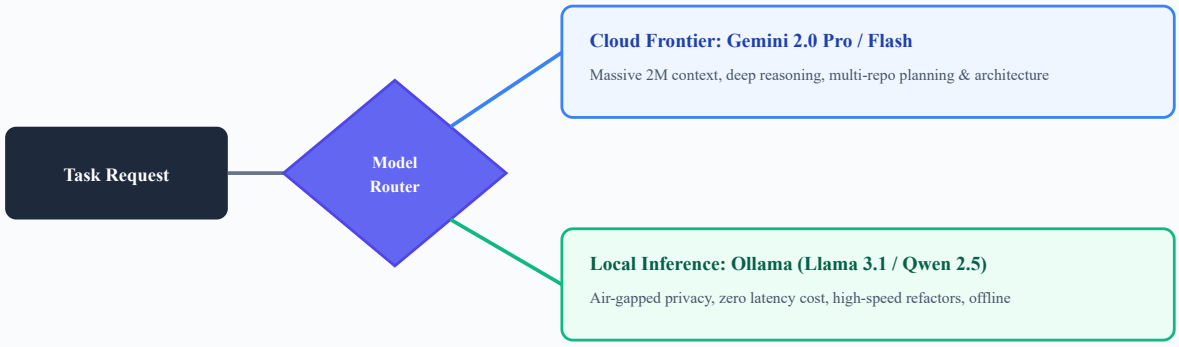


Figure 2.2: Dynamic Model Routing Architecture in Antigravity 2.0.

Model Tier	Representative Engine	Context Window	Ideal Workloads
Frontier Cloud Reasoning	Gemini 2.0 Pro	Up to 2,000,000 tokens	Whole-repository architectural planning, complex bug diagnosis across distributed systems, multi-agent orchestration.
High-Speed Cloud Execution	Gemini 2.0 Flash	Up to 1,000,000 tokens	Rapid single-file edits, routine test generation, continuous chat streaming, docstring generation.
Local / Air-Gapped Engine	Ollama (Llama 3.1 8B/70B, Qwen 2.5 Coder)	8K to 128K tokens	Offline development, strictly regulated confidential codebases, zero cloud transmission, local unit test fixes.

2.3 The Tool Execution Engine

How does an agent actually interact with your computer? In Antigravity 2.0, the LLM does not execute raw operating system syscalls directly. Instead, every action flows through the **Tool Execution Engine** using strictly typed schemas.

When an agent decides to modify a file, it produces a structured tool call. For example, using the surgical `replace_file_content` tool:

```
{
  "TargetFile": "C:/projects/auth/service.py",
  "StartLine": 42,
  "EndLine": 50,
  "TargetContent": "def verify_token(token):\n    return jwt.decode(token, SECRET, algorithms=['HS256'])",
  "ReplacementContent": "def verify_token(token):\n    # Enforce RS256 with JWKS verification\n    return jwt.de
}
```

The Tool Execution Engine performs five deterministic validation steps before writing to disk:

1. **Permission Verification:** Validates whether the target path falls within authorized workspace boundaries.
2. **Exact Target Matching:** Checks that `TargetContent` matches the disk content character-for-character, preventing accidental clobbering if the file changed concurrently.
3. **Atomic Disk Write:** Writes changes through temporary staging files to avoid partial corruption.
4. **Linter Interception:** Runs background syntax verification on the modified file immediately.
5. **Transcript Recording:** Appends the tool input, output, and execution metadata to the session transcript.

2.4 Context Window Management and Progressive Disclosure

Even with multi-million token context windows, flooding an LLM with an entire repository degrades reasoning performance, increases latency, and escalates costs. Antigravity 2.0 solves this with **Progressive Disclosure**.

> **CORE PRINCIPLE: PROGRESSIVE DISCLOSURE**
>
> An agent should only load the minimum information necessary to take the next correct step. Antigravity never dumps whole codebases into context blindly; it uses hierarchical pointers, file outlines, AST summaries, and artifact references.

When exploring an unfamiliar project, an agent follows an information hierarchy:

- **Level 1: Directory Tree & Workspace Summary:** High-level folder hierarchy and `README.md`.
- **Level 2: Symbol Signatures & Interfaces:** Class outlines, exported function signatures, and type definitions extracted via AST analysis.
- **Level 3: Exact Implementation Slices:** Reading only lines 40–90 of the specific file under active refactoring.

2.5 The Conversation Transcript System

Every interaction in Antigravity 2.0 is backed by an append-only, idempotent **Transcript Store**. The transcript captures the exact timeline of user prompts, agent thoughts, structured tool calls, execution outputs, and user approval events.

This architecture enables several critical capabilities:

- **Crash Resilience:** If a local machine reboots or network drops during a 20-minute refactor, Antigravity reconstructs the agent's exact cognitive state upon restart.
- **Branch Forking & Time-Travel:** Developers can fork a conversation transcript at step 5 to explore an alternative architectural hypothesis without losing the original branch.
- **Audit Trails:** Enterprise compliance teams can review complete records of every command executed and every file edited by the AI.

2.6 Security Layers: Sandboxing, Permissions, and Allowlists

Granting an AI system access to run terminal commands and edit files requires strict security boundaries. Antigravity 2.0 implements defense-in-depth security:

Security Layer	Mechanism	Protection Provided
Workspace Sandboxing	Virtual filesystem jails & path boundary enforcement	Prevents agents from reading or modifying files outside designated project directories (e.g., blocks access to `~/.ssh` or system files).
Command Allowlists & Denylists	Pattern-matching engine with risk classification	Safe commands (`git status`, `npm test`) run automatically; destructive commands (`rm -rf`, `git push --force`, `DROP DATABASE`) require explicit human approval.
Human-in-the-Loop Gating	Interactive approval prompts in Desktop & IDE surfaces	Ensures critical external operations (e.g., deploying to production, running database migrations) cannot occur without human sign-off.
Local Redaction Filters	Regex-based PII and secret scanner	Automatically strips API keys, OAuth tokens, and sensitive credentials before transmitting prompt data to cloud models.

★ **KEY TAKEAWAYS FROM CHAPTER 2**

- The PPAV loop (Perceive → Plan → Act → Verify) provides closed-loop autonomy, allowing agents to diagnose compiler/test errors and self-correct automatically.
- Antigravity's Model Router flexibly blends massive-context cloud reasoning (Gemini 2.0) with private, zero-latency local inference (Ollama Llama 3.1).
- Tool calls execute through a strictly typed engine with atomic disk writes, target-matching verification, and sandbox boundaries.
- Progressive disclosure protects the context window, feeding models hierarchical summaries rather than raw code dumps.
- Multi-layered security—including workspace sandboxing, command allowlists, and human-in-the-loop gating—guarantees safe, audit-compliant execution.

CHAPTER 3

The Antigravity Ecosystem

A world-class agentic platform must meet developers where they work. Antigravity 2.0 is not a single rigid application; it is a unified ecosystem accessible across three distinct user surfaces, all powered by a shared agent runtime core and customized through a modular configuration stack.

Whether you prefer a standalone AI-first IDE, a dedicated Desktop mission-control application, or high-speed terminal commands inside a headless CI/CD pipeline, the underlying intelligence, tools, and memory remain 100% consistent.

3.1 Three Surfaces Compared

Antigravity 2.0 provides three primary interfaces tailored for different engineering workflows:

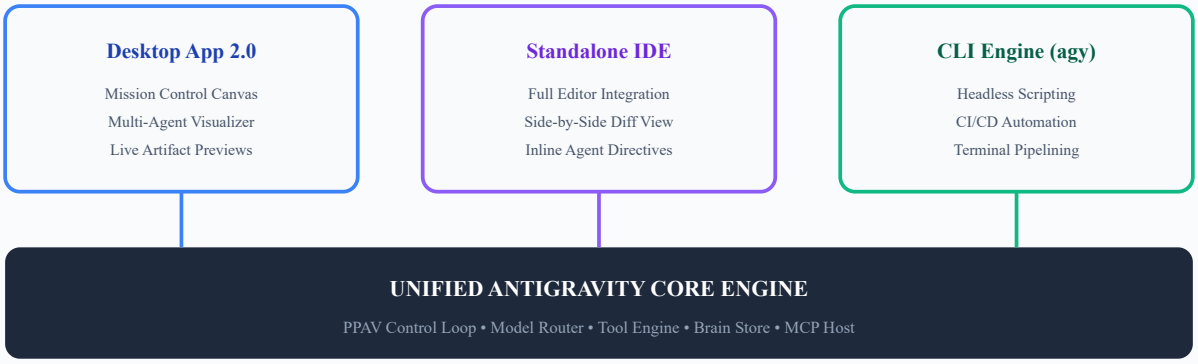


Figure 3.1: Unified Antigravity Core Shared Across All Three Surfaces.

1. Antigravity Desktop App 2.0

The Desktop App serves as your agentic mission control center. It provides rich visual canvas interfaces, real-time multi-agent hierarchy graphs, interactive artifact renderers (displaying markdown reports, Mermaid architecture diagrams, and HTML mockups), and comprehensive permission-approval consoles. It is ideal for project planning, large-scale refactoring overviews, and multi-subagent coordination.

2. Standalone IDE

Built on modern editor foundations, the Standalone IDE embeds Antigravity agents directly into your code editing flow. You get split-pane diff editors, inline ghost text suggestions, breakpoint-aware agent

debugging, and hotkey triggers. When you want to stay in flow while writing code with an agent pair-programmer, the IDE surface is unmatched.

3. The CLI (`agy`)

The `agy` command-line utility brings full agentic power to your terminal and CI/CD pipelines. It supports headless execution, unix piping (`cat error.log | agy fix`), automated script generation, and remote container execution.

3.2 Surface Decision Matrix

Engineering Scenario	Desktop App 2.0	Standalone IDE	CLI (`agy`)
Architectural Design & Planning	**Recommended** (visual artifacts & canvas)	Secondary	Unsupported
Active Daily Coding & Debugging	Secondary	**Recommended** (inline diffs & editor flow)	Possible
Multi-Subagent Team Orchestration	**Recommended** (live hierarchy tree)	Supported (tabbed)	Scriptable
CI/CD Pipelines & GitHub Actions	Unsupported	Unsupported	**Recommended** (headless non-interactive)
Remote Headless Servers / SSH	Unsupported	Supported (Remote SSH)	**Recommended** (native terminal)

3.3 The Customization Stack

Antigravity 2.0 adapts to your specific team conventions, style guides, and domain tooling through a layered **Customization Stack**. Each layer addresses a specific scope of control:

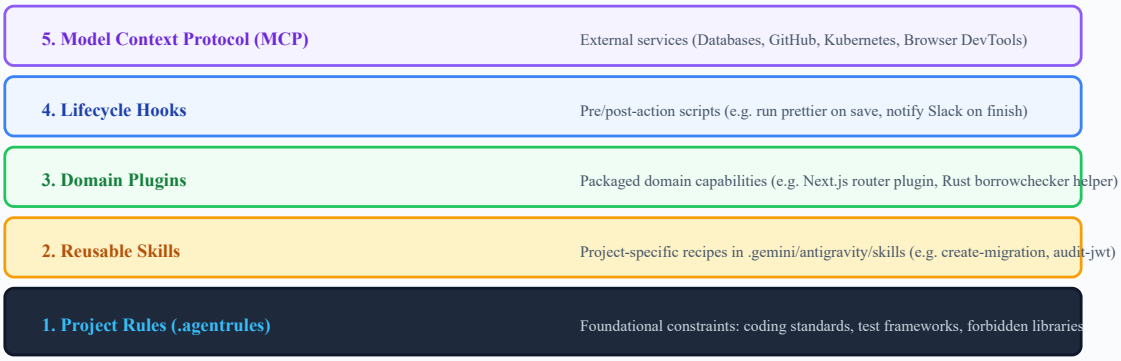


Figure 3.2: The Five-Tier Antigravity Customization Stack.

1. **Rules (`.agentrules`):** Foundational, persistent guardrails. These markdown/plaintext files live at your repository root and define mandatory coding standards, naming conventions, required testing frameworks, and architectural constraints.
2. **Skills (`skills/`):** Reusable procedural recipes. A skill provides explicit step-by-step instructions for repetitive engineering workflows (e.g., adding a new GraphQL mutation or generating a Prisma database migration).
3. **Plugins:** Distributable packages that bundle custom tools, rule sets, and skills for specific frameworks (e.g., an AWS CDK plugin or a Flutter mobile development pack).
4. **Lifecycle Hooks:** Event-driven triggers executed before or after specific agent actions (e.g., auto-formatting modified files with Prettier or checking security vulnerabilities before commit).
5. **Model Context Protocol (MCP):** The universal standard for exposing external tools and APIs to the agent runtime.

3.4 The Brain Directory and Artifact System

Where does an agent store its long-term memories, plans, research notes, and intermediate drafts? In the project's **Brain Directory** (located in your application data directory under `brain/<conversation-id>`).

The Brain system separates ephemeral conversational chatter from persistent, structured engineering deliverables called **Artifacts**.

Brain Component	Filesystem Location	Function & Purpose
Structured Artifacts	`brain/<session>/*.md`	Persistent, user-facing documents: architectural plans, audit reports, database entity schemas, migration checklists.
Artifact Metadata	`brain/<session>/*.metadata.json`	Tracks user feedback state, preview requirements, and version lineage for each artifact.
Scratch Space	`brain/<session>/scratch/`	Temporary execution workspace for one-off diagnostic scripts, temporary test builds, and intermediate data files.
Subagent Stores	`brain/<session>/.agents/`	Isolated transcript logs and memory state for concurrent child subagents.

BEST PRACTICE: WORKING WITH ARTIFACTS

Treat artifacts as living project documents. When embarking on a major feature, instruct the agent to generate an `architecture_plan.md` artifact first. Review and edit the artifact in the Desktop or IDE visual preview before clicking **Proceed** to initiate automated code generation.

KEY TAKEAWAYS FROM CHAPTER 3

- Antigravity 2.0 delivers three tailored surfaces—Desktop App 2.0 for mission control, Standalone IDE for inline code authoring, and `agy` CLI for automation—all sharing the same unified core engine.
- The Customization Stack (Rules → Skills → Plugins → Hooks → MCP) enables granular, layer-by-layer customization of agent behavior and environment access.
- The Brain directory and Artifact system provide persistent, structured memory and live markdown previews, bridging human architectural oversight with autonomous agent execution.

End of Free Sample

You have reached the end of the free sample edition of **Antigravity 2.0: The Complete Field Manual**. The foundations covered in these first three chapters establish the core paradigm of agentic software engineering and local offline AI execution.

The full publication features **31 comprehensive chapters across 7 distinct parts**, including deep-dives into hooks JSON schemas, Model Context Protocol configurations, custom workspace rules, sandboxed execution runtimes, and complex multi-agent teamwork blue prints.

What You'll Unlock in the Full Version:

- Setting up local private LLMs (Llama 3.1, Qwen 2.5, DeepSeek) with Ollama
- Writing Custom Workspace Rules (GEMINI.md) & modular Skill definitions
- Custom JSON configurations for the event-driven Lifecycle Hooks engine
- Model Context Protocol (MCP) integrations using Stdio & SSE transports
- Multi-agent task distribution strategies and python SDK orchestration

About This Book

Antigravity 2.0: The Complete Field Manual is the definitive operational handbook and deployment guide for Google's AI-first agentic development platform. Spanning 31 chapters across 7 detailed parts, this expanded volume covers everything from initial workstation configuration to production-grade multi-agent orchestrations.

WHAT YOU'LL MASTER:

- Local private AI deployment via Ollama (Llama 3.1, Qwen 2.5, DeepSeek)
- Workspace Rules (GEMINI.md), custom Skills, and Plugins
- Event-driven Lifecycle Hooks JSON configuration & stdin/stdout API
- Model Context Protocol (MCP) transport pipelines (Stdio & SSE)
- Multi-agent task distribution and planning mode artifacts

Published in Toronto, Canada • 2026 Edition