Futurum®

Google Cloud

How Google is Constructing the Path for AI-Generation Developers

MARCH 2026

Introduction

AI-driven software development defines the current state of software engineering. Software is no longer built primarily through deterministic code authored by humans and later augmented with AI. It is now built by assembling AI models, tools, and agents into systems that plan, reason, execute, and iterate. This shift has reshaped how software is created, tested, and operated, and it has reset expectations for developers, platforms, and organizations.

Developers now work across AI-native development surfaces, agent frameworks, and execution environments that treat models and agents as core system components. They design agent behaviors, establish control planes that govern autonomy, and apply security and governance models suited to non-deterministic systems. Open standards such as Model Context Protocol (MCP), Universal Commerce Protocol (UCP), Agent2Agent (A2A), Agent Control Plane (ACP), and Agent Payments Protocol (AP2) form the connective layer that allows agents, tools, and platforms to interoperate at scale. These standards anchor how agentic systems are built, extended, and integrated across environments.

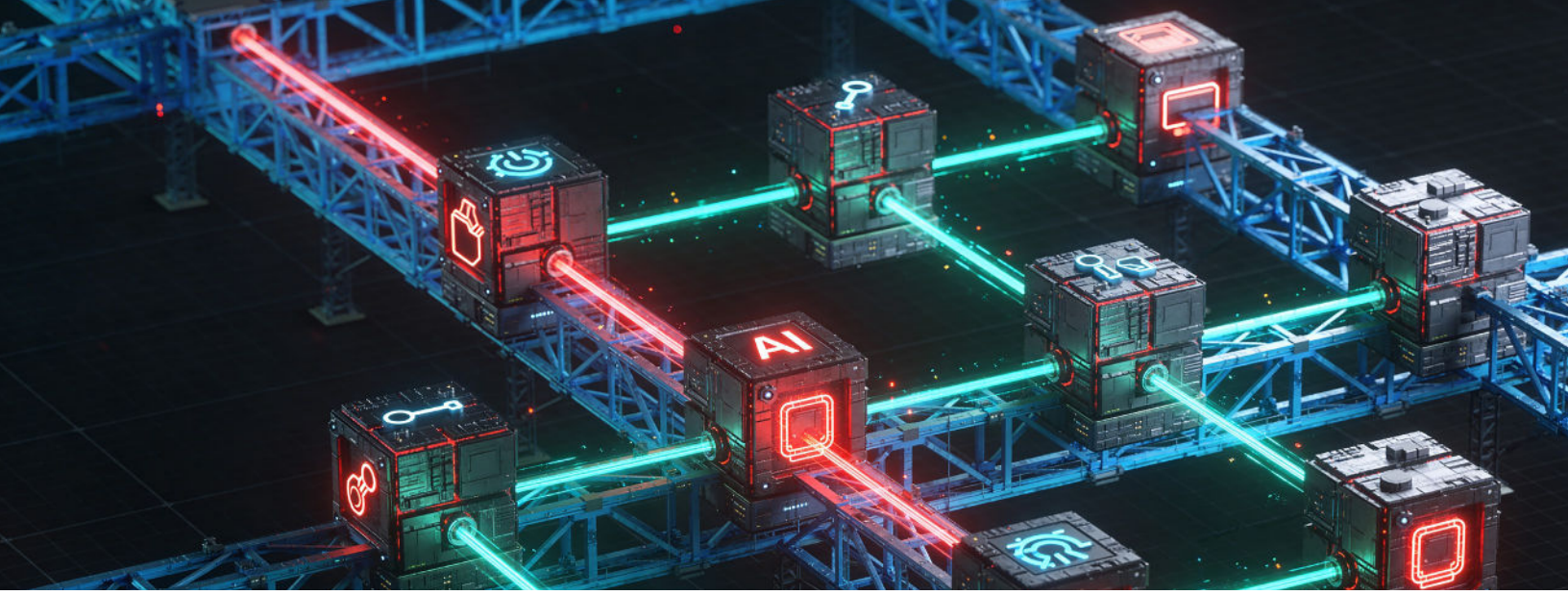
The defining challenge in this environment is execution. Developers expect rapid access to new models, agent capabilities, and development patterns, and they expect to apply them without rebuilding workflows each time platforms evolve. Platforms that collapse the distance between innovation and practical use determine how quickly teams move from experimentation to production.

An effective AI development platform provides flexibility and choice without fragmentation. Sustained developer velocity comes from delivering innovation through consistent patterns rather than isolated tools, and from aligning developer surfaces, runtimes, and platform services into a coherent system that evolves as agent-driven software becomes the default mode of development.

Management Context: AI-Driven Development at Scale

Futurum Research data indicates that AI-driven development is already embedded in day-to-day engineering work. More than 60% of software development teams now integrate AI-assisted development tools directly into daily workflows.

Source: Futurum Research, Q3 2025. AI & Developer Productivity 2025: The New Normal for Engineering Teams.



AI and Agent: How Software Development is Changing

Software development has entered an agent-centric phase. AI is no longer an assistive layer added to development tools. It is the foundation from which software is constructed. As a result, developers are becoming the engineers of agents that create software. This role shift reshapes development practices, tooling expectations, and platform requirements.

In practice, this shift is visible from the moment development begins. Developers increasingly start work by expressing intent rather than writing scaffolding code, using environments such as Google AI Studio or Google Colab to explore reasoning patterns, agent behavior, and model capabilities before committing to architecture. Gemini models serve as planning and reasoning engines in these early stages, allowing developers to test how agents interpret goals, decompose tasks, and respond to constraints. These interactions are not throwaway experiments; they establish patterns that persist as work moves into structured development and production workflows.

From Applications with AI to Applications Built from AI

Modern software systems are assembled from models, agents, tools, and workflows that operate together over time. These elements function as first-class components of the system architecture. Development no longer centers on embedding isolated model calls inside code. It centers on designing agentic systems that plan tasks, act across tools and services, validate outcomes, and maintain state across interactions.

Developers define how agents reason, how they coordinate with other agents, and how autonomy is constrained through explicit controls. Frontier models capable of multimodal reasoning and long-horizon planning make sustained agent execution practical at scale. Agents participate directly in coding, testing, deployment, and operational workflows rather than serving as passive assistants.

Management Context: Growth in AI Investments

Enterprise investment patterns show a shift from isolated tooling experiments toward platform-level adoption. Spending on AI-enabled development platforms grew more than 65% year-over-year in 2025, signaling sustained commitment to operationalizing AI across the software lifecycle rather than treating it as a discretionary add-on.

Source: Futurum Research, Q4 2025. Enterprise AI Investment Outlook 2025.

This change is reinforced by the capabilities of frontier models such as Gemini 3, which support long-horizon reasoning and multimodal understanding across development tasks. Through the Gemini API, Gemini Code Assist, and related developer surfaces, agents participate directly in refactoring, test generation, dependency analysis, and deployment validation. The model is no longer called sporadically inside an application; it is embedded into the continuous execution of the development workflow itself.

Developers as Engineers of Agentic Workflows and Systems

As developers engineer agentic systems, their focus shifts from authoring individual functions to designing systems of intent, execution, and verification. They specify how agents interact with data, infrastructure, and external services, and they define the boundaries within which agents operate and the criteria by which outcomes are evaluated.

This work increasingly involves coordinating multiple agents with distinct responsibilities. Developers design planning agents that decompose goals, execution agents that perform actions across tools and services, and validation agents that assess outcomes against defined criteria. Autonomous agents such as Jules illustrate this pattern by participating directly in development workflows, generating code, running tests, and proposing changes while remaining subject to human-defined boundaries and review. The developer's role is to shape these systems, not to relinquish control to them.

This work requires development environments that produce durable, shareable artifacts rather than ephemeral interactions. Specifications, documentation, and context must be explicit and reproducible so they can be consumed by both humans and machines. Systems must evolve as models improve, without forcing teams to re-architect applications with each platform update.

Continuous Innovation as a Defining Condition

Rapid innovation is a steady operating condition of AI-driven development. Models advance, agent frameworks expand, and development patterns mature continuously. Developers expect platforms to expose these advances quickly and coherently.

The differentiator is stability through abstraction. Platforms that provide consistent extensibility models and clear paths from experimentation to production allow developers to absorb change without sacrificing momentum. Learning, building, and operating occur as a continuous loop rather than discrete phases.

Google's platform approach reflects this reality by preserving interaction models even as underlying capabilities evolve. Developers encounter consistent abstractions as they move between experimentation, agent-first development environments, and production runtimes, reducing the need to relearn workflows as models and tools advance. This philosophy aligns with Google's long-standing stewardship of languages such as Go, where stability at the abstraction layer enables sustained innovation underneath.

Integration and Learning as Execution Advantages

Integration and learning have always been part of software engineering. In the agent era, the advantage lies in how efficiently innovation becomes executable. Developers value platforms that surface new capabilities through familiar patterns, shared abstractions, and opinionated guidance that reduces decision overhead.

Treating agents as applications becomes concrete when execution environments are designed around that assumption. Managed runtimes such as Cloud Run and Vertex AI, including Vertex AI Agent Engine, allow agent-based systems to run as long-lived services with defined lifecycles, shared state, and observable behavior. Evaluation, tracing, and operational visibility

are applied continuously, enabling developers to iterate on agent logic with the same discipline used for traditional production software.

Moving from prototype to production depends on treating agents as applications. Managed runtimes, shared memory and context models, evaluation tooling, and operational visibility must be native platform capabilities. When these are built in, developers spend their time engineering agent behavior and system outcomes rather than assembling infrastructure.

Google's Position in the Current Development Landscape

Google approaches AI-driven development as a platform problem rather than a tooling problem. The strategic focus is cohesion: aligning how developers enter the ecosystem, how they build agentic systems, and how those systems are deployed and operated.

Developers expect choice in languages, frameworks, and models, but they also expect clear paths that allow work to progress across levels of abstraction without fragmentation. Google's platform direction reflects this expectation by connecting developer surfaces and platform services into a continuous experience that supports rapid experimentation, structured development, and production-grade operation of agentic software.

These capabilities reflect a deliberate effort to align entry paths, agent-first development surfaces, execution environments, and governance into a single development continuum. This alignment sets the stage for understanding how individual tools connect into a cohesive developer experience rather than operating as isolated points of innovation.

From Individual Tools to a Cohesive Developer Experience

Developers do not experience platforms as portfolios. They experience them through entry points, workflows, and how easily work moves from experimentation into production. What matters is whether tools connect into a coherent experience as requirements grow.

Google AI Studio provides a fast, self-service entry point for working with Gemini models, exploring agent behavior, and prototyping ideas without configuring cloud environments or enterprise billing.

Parallel environments such as Google Colab serve a similar role for exploratory and data-driven workflows, allowing developers to experiment with models, agents, and logic in a notebook-based context. These entry points are intentionally low-commitment, enabling learning and exploration without forcing early decisions about infrastructure, deployment, or governance.

That work transitions naturally into Antigravity, where developers move from exploration to engineering agentic workflows that plan, execute, and validate work across repositories, tools, and services. The surface changes, but the development model remains consistent.

Agentic work progresses into production without replatforming. Developers building agents through Antigravity or the Agent Development Kit deploy them using managed runtimes such as Vertex AI Agent Engine or Cloud Run, where agents operate as long-running services with defined execution boundaries and scaling behavior. Trust, evaluation, and governance are integrated into the same platform surfaces developers already use.

As agent-driven systems scale, Kubernetes-based execution becomes increasingly important. Google Kubernetes Engine provides a consistent foundation for running inference-heavy and agent-oriented workloads across environments, supporting portability and operational consistency. This allows teams to scale agentic systems without fragmenting execution models or introducing environment-specific behavior.

These experiences show how developer entry paths, agent-first development surfaces, and platform services align into a cohesive whole. Developers retain flexibility in languages, models, and deployment patterns while benefiting from consistent abstractions as work evolves from experimentation into production.

Application and platform teams extend these workflows through services such as Firebase and Firebase Studio, where agent-driven logic connects directly to end-user applications that support consistent application delivery across platforms. Platform-oriented services including Application Design Center and App Hub provide additional structure for designing, governing, and operating applications built on agentic foundations, reinforcing cohesion across development and operations as systems move into sustained production use.

Introducing the AI Development Platform Framework

AI-driven development operates at a scale and level of autonomy where tools alone no longer carry the system. Tools accelerate individual tasks, but agentic software introduces lifecycle, governance, and operational requirements that require platform-level support.

An AI development platform absorbs complexity so developers can focus on engineering outcomes. It supports learning, experimentation, development, deployment, and operation as a continuous loop. New models, agent capabilities, and development patterns are incorporated without forcing developers to rebuild workflows or integration scaffolding.

A framework provides a way to reason about platform cohesion without prescribing a single workflow. It distinguishes tools from the capabilities they enable and provides a lens for evaluating platform direction over time. Each component addresses a distinct source of friction in agent-driven development. Weakness in any component limits the effectiveness of the platform as a whole.

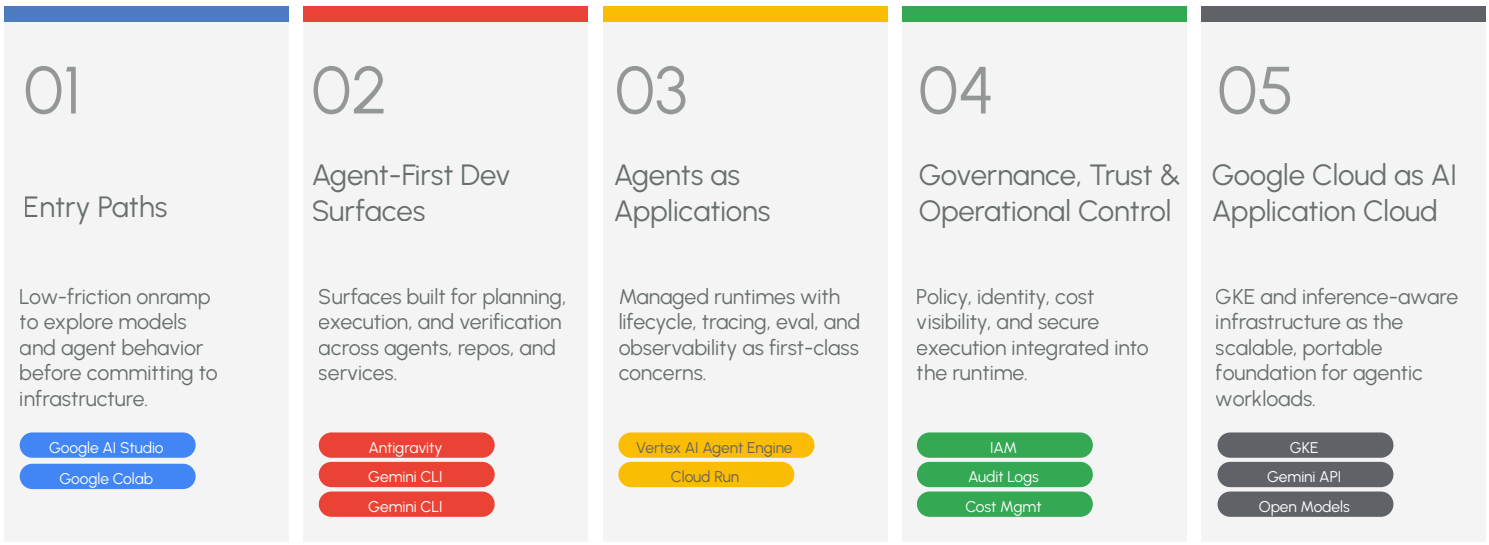
Core Components of an AI Development Platform

Clear and Progressive Entry Paths

Entry paths shape how developers first experience a platform and whether they stay with it as work evolves. Clarity at the starting point matters because developers often begin by experimenting rather than committing to infrastructure decisions.

Google AI Studio provides low-friction entry for learning and prototyping while preserving continuity with more advanced development surfaces. As requirements grow, developers advance naturally toward structured development and production workflows without abandoning their mental model or retooling from scratch.

AI Development *Platform* Framework



Source: Futurum Research

Agent-First Development Surfaces

Agentic software requires development surfaces designed for planning, execution, and verification over time. Agent-first surfaces orchestrate work across agents, tools, repositories, and systems rather than focusing solely on code editing.

Antigravity prioritizes agent management and workflow orchestration, producing durable, reviewable outputs with shared context. Local tooling such as the Antigravity and Gemini CLI extends this model into the developer environment, enabling agents to propose actions, navigate filesystems, and automate tasks as part of a coherent workflow.

Platform Services That Treat Agents as Applications

Agents scale only when treated as applications. Managed runtimes such as Vertex AI Agent Engine and Cloud Run support deployment, execution, evaluation, and operation as first-class concerns. Developers move from local development to production without redesigning systems.

Native evaluation, tracing, and observability services provide visibility into agent behavior and outcomes, enabling continuous improvement rather than one-time deployment.

Governance, Trust, and Operational Control

As agents gain autonomy, trust and control become platform responsibilities. Secure execution environments, policy enforcement, identity and access controls, and cost visibility are integrated into execution rather than imposed externally.

Governance enables velocity by allowing developers to build powerful agentic systems with accountability and safety built in.

Google Cloud as the AI Application Cloud

Google Cloud centers on running and operating intelligent systems rather than provisioning infrastructure. The application cloud aligns execution, scale, and control around agent-driven workloads.

Managed runtimes treat agents as production software, while Kubernetes provides a consistent execution model across environments. Inference-aware routing, scheduling, and resource allocation support predictable performance without manual tuning.

Choice remains central. Developers can access Gemini models through APIs, deploy open or proprietary models in managed environments, or mix approaches based on cost, performance, and compliance needs. Developer surfaces, runtimes, and governance controls align around a common execution model, allowing agents to move into production without losing context or visibility.

Analyst Perspective and Outlook

Platform coherence now outweighs feature velocity. Developers evaluate whether platforms sustain momentum as workflows, models, and agent capabilities evolve. Integrated systems of work outperform fragmented toolchains.

Tolerance for stitching together disconnected tools continues to decline as agentic systems move into production. Openness and interoperability are baseline expectations. Open standards provide stability by allowing adoption of new capabilities without brittle dependencies.

Google is well-positioned due to its breadth of capability and increasing alignment across developer surfaces, runtimes, and platform services. Execution risk lies in simplification and guidance rather than technical depth. Maintaining clear paths and coherent experiences will matter as much as introducing new capabilities.

Conclusion

The trajectory of software development toward agent-driven systems is clear. Developers are the engineers of agents that create software, and platforms are becoming the operating systems for that work.

The platforms that succeed will be those that let developers move from intent to execution without friction, evolve as agent-driven systems mature, and preserve human judgment, accountability, and creativity as first-class elements of the software lifecycle.

Important Information About This Report

AUTHORS

Mitch Ashley
Vice President & Practice Lead Software Lifecycle Engineering | The Futurum Group

PUBLISHER

Futurum Research

INQUIRIES

Contact us if you would like to discuss this report and The Futurum Group will respond promptly.

CITATIONS

This paper can be cited by accredited press and analysts, but must be cited in context, displaying author's name, author's title, and "The Futurum Group." Non-press and non-analysts must receive prior written permission by The Futurum Group for any citations.

LICENSING

DISCLOSURES

The Futurum Group provides research, analysis, advising, and consulting to many high-tech companies, including those mentioned in this paper. No employees at the firm hold any equity positions with any companies cited in this document.

Google Cloud

ABOUT GOOGLE CLOUD

Google Cloud provides a suite of tools designed to support the software development lifecycle, offering various entry points for AI and application development. The Google Cloud developer tools ecosystem includes Antigravity, Gemini CLI, Google AI Studio, and Jules. For infrastructure and deployment, the platform utilizes Google Kubernetes Engine (GKE), Vertex AI and Cloud Run. These tools allow developers to transition from initial prototyping to production-scale deployment while maintaining enterprise-grade security protocols. Together, these capabilities and tools position Google Cloud as a leader in AI-driven software development space.

Futurum®

ABOUT THE FUTURUM GROUP

The Futurum Group is an independent research, analysis, and advisory firm, focused on digital innovation and market-disrupting technologies and trends. Every day our analysts, researchers, and advisors help business leaders from around the world anticipate tectonic shifts in their industries and leverage disruptive innovation to either gain or maintain a competitive advantage in their markets.