DOME SYSTEMS

A New Infrastructure Layer, Again?

Considerations for implementing platform services for AI agents in the enterprise

Executive Summary

Traditional apps follow instructions. Agentic apps pursue goals. Enterprise software is undergoing its most significant architectural shift since the move to cloud. Agents compress workflows, handle complexity that deterministic code cannot, and adapt to context at runtime. Every enterprise that deploys them gains speed and efficiency. Every enterprise that does not will fall behind.

The challenge is equally clear. Agents make decisions at runtime such that the same agent, given the same input, may take a different path each time. Existing governance tools were built for applications that follow fixed paths. Agents require governance of decisions. These are different problems, and the gap is growing with every agent deployed.

This gap is compounded by fragmentation. Agents are appearing simultaneously across cloud providers, data platforms, SaaS applications, and developer tools. Each surface has its own partial answer to governance. None provide consistency across the others. The result is an enterprise estate where agents are productive but ungoverned creating the same shadow IT pattern that followed cloud and containers, but with higher stakes because the systems in question reason for themselves.

This paper provides a framework for enterprise leaders and platform teams. It defines what an agent is, maps the surfaces where agents operate, identifies the governance concerns that cut across all of them, and proposes a blueprint for consistent operations. The goal is practical: give platform teams a clear model for enabling agent deployment at scale without sacrificing governance.

A New Era of Agentic AI Applications

The scale of the shift is difficult to overstate. Gartner forecasts 40% of enterprise applications will embed AI agents by the end of 2026, up from less than 5% at the start of 2025 (Gartner, August 2025). A category that barely existed two years ago, reaching nearly half the enterprise application estate in a single budget cycle.

The breadth is equally striking. A logistics company deploys agents that reroute shipments based on weather, traffic, and port congestion. A financial services firm runs agents that monitor transactions, flag anomalies, and execute compliance workflows without human intervention. A software team ships code with agents that write tests, review pull requests, and deploy to staging autonomously. These are production workloads, running today, across every sector.

The infrastructure tells the same story. API token consumption — a proxy for how much work models are doing — grew from 10 trillion tokens per year to over 100 trillion in twelve months on a single routing platform (a16z / OpenRouter, "State of AI," 2025). Enterprise API spending on generative AI reached $8.4 billion by mid-2025, up from $500 million two years earlier.

And this is only the beginning. As frameworks mature and inference costs fall, every workflow that involves judgment, triage, or coordination becomes a candidate. Every enterprise function is building or evaluating agents.

The question for enterprise leaders is no longer whether agents will be a significant part of the technology estate. They already are. The question is whether the governance infrastructure will be ready for what comes next.

From deterministic to probabilistic execution

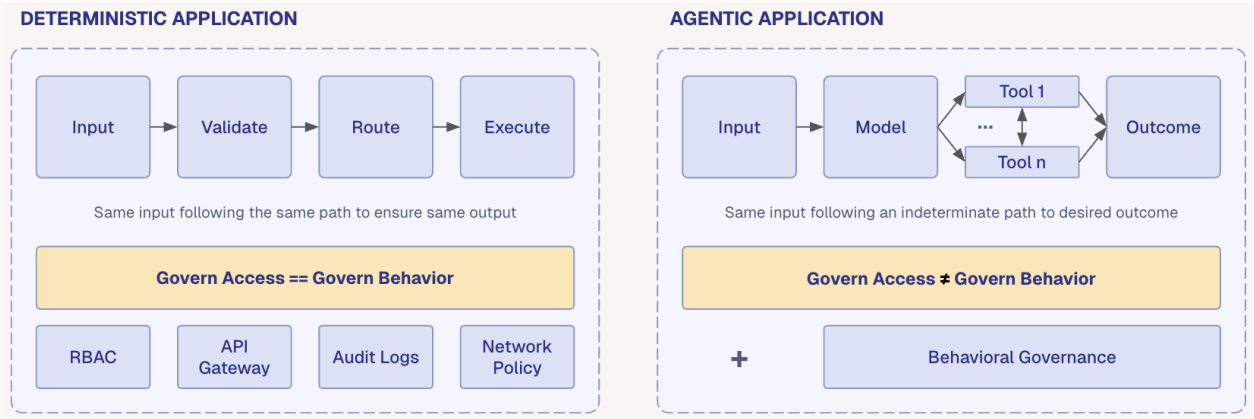
Prior to AI, enterprise software did what the developer told it to do. An expense report workflow validates the receipt, routes it to the right approver, applies the policy, and processes the payment. Same input, same output, every time.

This is deterministic execution, and enterprise governance was built around it. Control who can access the application, configure what it can do, and you control the outcome. RBAC, API gateways, audit logs, network policies: the entire operational toolkit assumes software follows a fixed path.

Agentic applications break that assumption.

An expense agent reads the receipt, decides the category, flags anomalies it was not programmed to look for, and routes based on judgment. The same receipt, submitted twice, may take a different path each time. The agent reasons about context and adjusts its behavior accordingly.

This is probabilistic execution. The developer provides capabilities and constraints. The agent decides how to use them.



This shift matters because the control point has moved. In a deterministic system, governing access is governing behavior. In an agentic system, governing access is necessary but no longer sufficient. What the agent decides to do with its permissions depends on reasoning at runtime, compounded by the risk of agents calling each other under assumed privileges.

The shift is not constrained to one platform. Agents are appearing across cloud providers, data platforms, SaaS, and developer tools. Each is building its own answer to governance. None are solving it consistently across the others.

Insufficient tooling and inconsistent operations is the enterprise problem. Governance is fragmented across every surface where agents run, and the existing toolkit was designed for a world where applications did not make their own decisions.

This pattern is not new. Every major shift — containers, cloud, microservices — followed a similar arc: a new pattern emerges, development teams adopt it faster than the governance around it catches up, and eventually a platform layer addresses the gap. The agent era follows the same arc, but faster. Teams are shipping agents at a pace that outstrips the operational practices around them. There is no established pattern for how an agent should be built, deployed, or governed.

For platform teams, this creates a familiar but intensified challenge: how do you provide a productive, governed environment when the thing being built reasons for itself?

The answer starts with defining what an agent actually is.

Components of an Agent

Strip away the marketing and the abstraction, and an agent is code. It is software. It executes in a runtime, it has dependencies, it produces output. In that sense, it is not fundamentally different from any other application a platform team has to support.

What makes it different is what the code does. An agent talks to a model and makes tool calls. That is the entire distinction. The code sends context to a large language model, the model reasons about what to do, and the agent acts on that reasoning by calling tools: APIs, databases, messaging systems, anything it has access to.

Every agent then, regardless of how it is built or where it runs, is composed of three things:

Component	What	Example Variations
Code	Runtime and orchestration. Receives input, manages state, invokes model, calls tools.	**Language:** Python, TypeScript, Go **Framework:** LangChain, Vercel AI, Crew AI **Runtime:** Serverless, k8s
Model	Reasoning engine. Receives context, decides what to do next.	**Frontier models**: Anthropic, OpenAI, Google **App-specific models:** SaaS **Open weight custom models:** Internal
Tool	Anything the agent calls to interact with the external world. APIs, databases, messaging.	**Protocol:** MCP, Rest **API:** Any

These three components compose every agent. A simple chatbot has minimal code, a single model, and no tools. A complex workflow agent has sophisticated orchestration code, may use multiple models, and has access to dozens of tools. The components are the same but the complexity varies.

The important thing for platform teams to understand is that each component introduces its own operational dimension. The code needs to be deployed, versioned, and monitored like any other software. The model needs to be configured, constrained, and may change between runs. The tools

need to be discovered, authorized, and audited. An operational model that only addresses one of these three is incomplete.

Categorizing the Agentic Estate for an Enterprise

The three components of code, model, and tool appear in every agent. But agents themselves appear across at least five distinct surfaces in the enterprise, each with its own runtime, its own governance model, and its own blind spots.

Understanding these surfaces is essential for platform teams, because a governance strategy that addresses only one or two of them leaves the rest ungoverned.

Surface	Examples	Flexibility of control		
		Code	Tool	Model
AI Platforms (AISP)	Anthropic, OpenAI			
Cloud Platforms (CSP)	AWS, Azure, GCP			
Data Platforms (DSP)	Salesforce, ServiceNow, Databricks, Snowflake			
SaaS	Writer, Harvey, Cognition			
Individual	Claude, ChatGPT, OpenClaw, Cursor			

The governance gap is visible through each component of the agent. It is the breadth of agents in the estate that underscores the enterprise governance challenge: how to impose consistent governance when applications are running on different platforms, each with their own approach?

The path then is to understand the control points: code, model, tool.

> **Code.** On a cloud platform, the platform team controls the runtime: they choose the framework, manage deployment, and own the lifecycle. On a data platform, the code runs

inside the vendor's environment with limited visibility. Inside an ISV application, the code is the vendor's — the enterprise has no control at all. The further down the table, the less the enterprise can govern how agents are built and deployed.

Model. On a cloud platform, the team selects the model, sets guardrails, and tracks configuration. On a data platform, the model may be embedded in the vendor's pipeline with limited configurability. Inside an ISV or SaaS application, model selection is the vendor's decision entirely. The enterprise cannot enforce consistency across models it does not choose.

Tool. On a cloud platform, tool access is governed by IAM and network policy — familiar but incomplete for agents that reason about which tools to call. On a data platform, tool access is scoped to the data boundary but uncontrolled beyond it. Inside an ISV, the agent accesses the vendor's internal systems and potentially the enterprise's systems through integrations the enterprise does not govern. At the Apps layer, a developer using Claude with MCP tools has unrestricted access to whatever the tool server exposes.

The agentic estate for an enterprise, then, consists of five surfaces of agent deployments, each with three components.

No existing tool provides consistent governance across all of them. That is the gap platform teams need to close, and from there build ever increasing value through the opportunities of agentic software development.

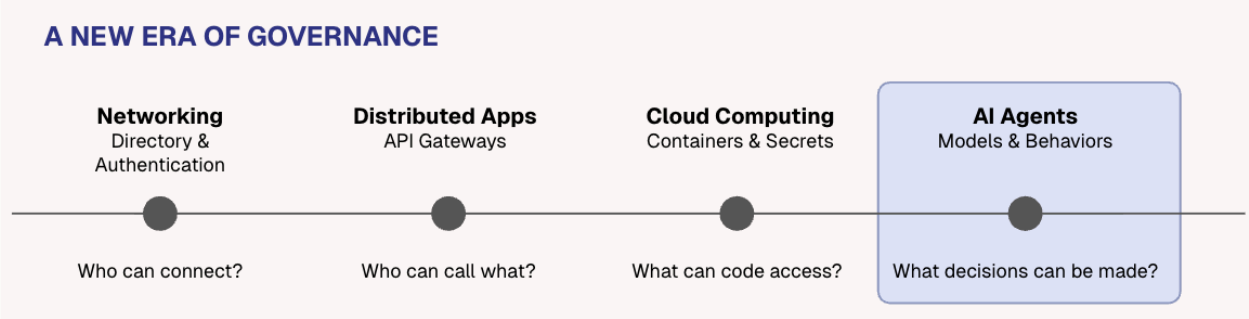
Why existing tools are insufficient

Platform teams already have operational tooling: RBAC, audit logs, secrets management, network controls, API gateways. These tools were designed for deterministic systems. They work well for controlling who can invoke a service, what resources a service can access, and what network paths are available.

They do not work well for agents, because agents reason about their own behavior. RBAC controls who can invoke an agent, but not what the agent will decide to do. An audit log records that an agent called an API, but not the reasoning that led to the decision to make that particular call which is ultimately what we will require to satisfy regulators. A secrets vault restricts database access, but does not prevent an agent from using the database within authorized parameters to do something the organization does not want done.

The existing tools control access. Agents require control of decisions. These are different problems, and they require different infrastructure.

Every era of computing that introduced new patterns of access and action eventually produced a governance infrastructure layer built for those patterns.



The agent era has introduced a new control problem: the decisions that reasoning systems make. The infrastructure that addresses it operates at the tool call boundary where reasoning becomes action. Platform teams that build this layer now will be ahead of the curve. Those that wait will face the same catch-up they faced with containers and cloud, but with higher stakes, because the systems they are governing can reason for themselves.

Enabling Consistent Operations for Enterprise AI Agents

With a clear definition and taxonomy of agents, the practical question follows: how to enable consumers and development teams to adopt this new model, while maintaining acceptable compliance and governance?

Three governance surfaces

Each component of an agent — code, model, tool — presents a distinct governance surface. Together they define where a platform team must operate.

Each governance surface presents a different type of control point.

Component	Control point	What it governs	Limitation
Code	Registration	Makes agents known and manageable: identity, capabilities, lifecycle, status	Necessary but does not control what the agent decides to do
Model	Configuration	Constrains reasoning: system prompt, model selection, guardrail settings	Constrains but does not deterministically prevent undesired behavior. Model gateways add economic and routing governance
Tool	Authorization	Evaluates what is about to happen and decides whether to allow it	Key point of governance that applies consistently across all agents and surfaces

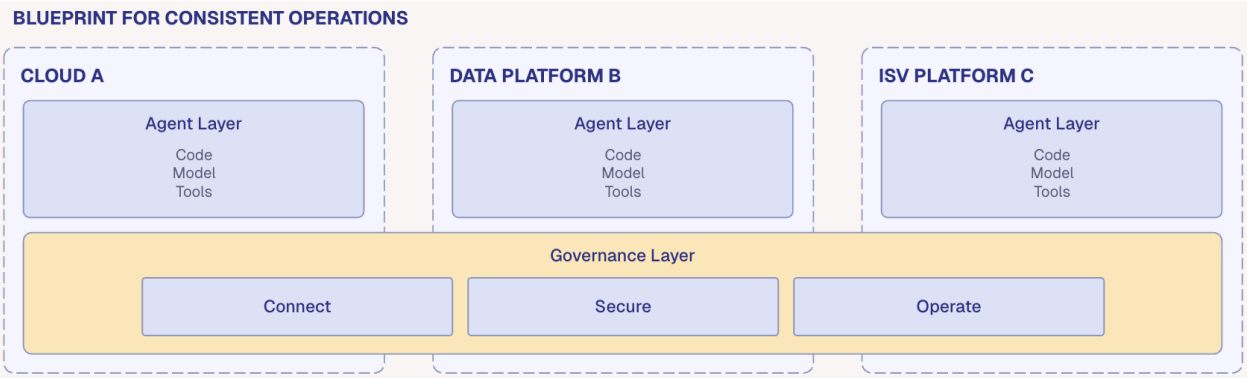
These control points remain generally true regardless of where the agent is deployed. An agent on AWS Lambda, an agent embedded in Salesforce, and an agent on a developer's laptop all have code, a model, and tools. All three control points are available in principle. The challenge is applying them consistently across all five surfaces.

A blueprint for platform teams

Platform teams have solved this class of problem before. When containers created deployment complexity, they built standardized deployment pipelines and orchestration platforms. When microservices created observability challenges, they built service meshes and centralized logging. When cloud adoption created identity sprawl, they implemented federated identity.

The agent era requires the same approach: a platform layer that provides consistent operations across all agents, regardless of where they run.

The blueprint is organized as a workflow with three operational domains. Every agent, on every surface, passes through the same sequence: **Connect → Secure → Operate.** Within each domain, specific actions deliver governance outcomes.



Domain	Outcome
Connect	Every agent is known, and every tool is accessible through a governed path.
Secure	Every action is evaluated against policy before it executes.
Run	Every decision is recorded, and the platform is observable through existing tools.

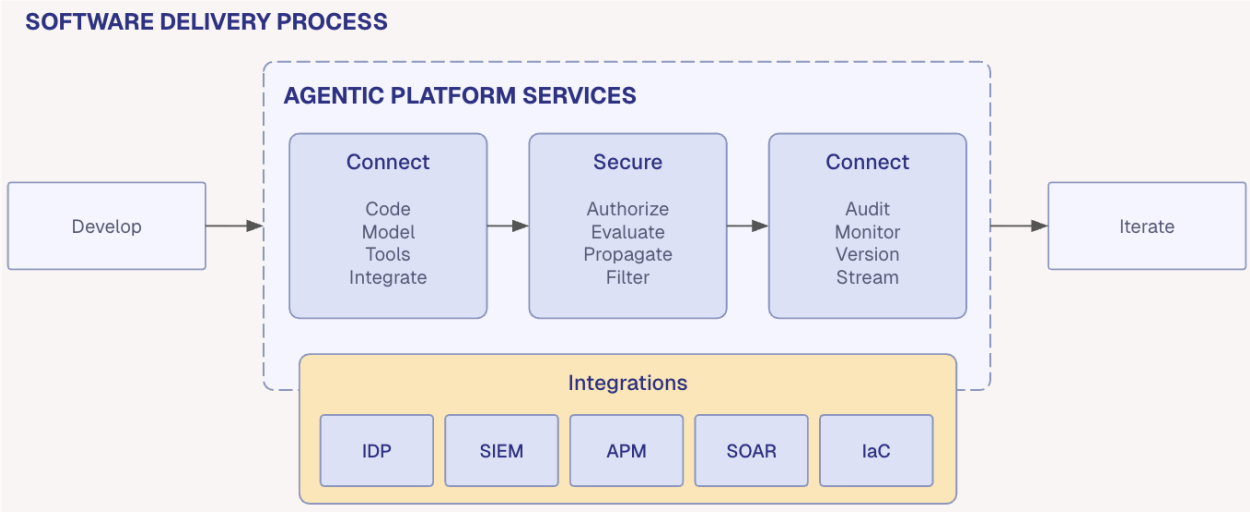
Each domain maps to the governance surfaces presented by code, model, and tool, and each applies across all five surfaces where agents operate. Enterprise platform teams will be challenged to deliver on some version of this blueprint.

Dome. The AI Infrastructure Management Platform.

The infrastructure this era requires is agent-native: built for reasoning systems from the ground up, not adapted from tooling designed for deterministic ones. And delivered in a manner that enables consistent control but without introducing developer and user friction.

Dome is the Agentic Infrastructure Platform: a system of control that provides consistent governance across every agent, every runtime, every cloud. Connect. Secure. Operate.

Every agent that makes a tool call passes through Dome. Dome observes the call, evaluates it against policy, and decides whether the call is allowed to execute. This is true whether the agent is a Lambda function in AWS, a query on a data warehouse, a workflow in a SaaS application, or a script on a developer's laptop.



Domain	Capability	Action
Connect	**Register**	Assigns identity to the agent and declares its capabilities. Makes the agent known and governable.
	Discover	Surfaces the tools the agent is permitted to use, filtered by policy. Replaces hardcoded endpoints and shared credentials.
	Provision	Issues credentials and configures access. Agents authenticate to Dome and never hold tool credentials directly.
	Integrate	Connects the agent to the governed path via SDK, sidecar, or gateway regardless of runtime or surface.
Secure	**Authorize**	Authorizes each agent using identity to prepare for onward execution.
	Evaluate	Applies full policy context: conditions, parameters, user context. Rapid deterministic evaluation.
	Propagate	Extends identity and authorization context across agent hierarchies. Prevents privilege escalation when agents invoke other agents.
	Filter	Controls what comes back. Permits the record, redacts the field. Governs outputs, not just inputs.
Operate	**Audit**	Records every governed action with full context: who, what, under which rule, with what result.
	Monitor	Tracks agent health, status, and behavior patterns. Surfaces anomalies before they become incidents.
	Version	Manages policy as code. Rules are versioned, testable in staging, and deployable through CI/CD.
	Stream	Pushes events to existing infrastructure: SIEM, APM, SOAR. Adds to the stack rather than replacing it.

Integrations. Dome connects downstream to the enterprise stack the platform team already operates: identity providers, SIEM, APM, SOAR, and infrastructure-as-code tooling. Governance decisions flow into existing workflows rather than creating new ones.

Governance capabilities for each agentic control point

Dome's capabilities map directly to the three agent components. For each — code, model, tool — the platform provides capabilities across the three operational domains: Connect, Secure, and Operate.

	Connect	Secure	Operate
Code	Registers each agent with identity, capabilities, and tier. Provisions credentials in a single step. Integrates via SDK, sidecar, or gateway.	Authorizes agent-to-platform communication. Isolates credentials — agents authenticate to Dome and never hold tool credentials directly.	Monitors agent health and lifecycle. Audits registration and configuration changes. Suspends or revokes from anywhere.
Model	Registers configuration per agent — system prompt, model selection, guardrail settings.	Evaluates policy independently of model reasoning — Cedar rules enforce boundaries regardless of what the model decides.	Audits decision context — not just what happened, but under which agent configuration.
Tool	Discovers tools through the MCP Gateway — a single governed endpoint across all connected servers. Filters discovery by permissions. Provisions credentials on egress.	Authorizes each tool call against per-agent, per-tool, per-action Cedar policy. Evaluates in sub-5ms, fail-closed. Filters responses — permits the record, redacts the field.	Audits every governed action with a full chain of evidence: who, what, under which rule, with what result. Streams events to existing infrastructure. Versions policy as code.

A path to implementation

For any platform team, standardizing processes while supporting innovation can be hard. For that reason, we'd propose a simple maturity approach to applying control across the estate. Each phase tightens governance progressively starting with non-invasive visibility and moving toward full operational control. Platform teams get value at every step without needing to commit to the full lifecycle upfront.

Phase	Actions	Outcome
1. Visibility Know what exists	Register agents. Catalog tools. Establish identity. Deploy audit trail. Non-invasive. No changes to existing agents required.	A complete, queryable picture of the agent estate: what agents exist, what they access, and what they do.
2. Policy Control what happens	Define authorization rules in Cedar. Enforce per-agent, per-tool policies. Version and test rules in staging. Deploy through CI/CD.	Every tool call is evaluated against policy before execution. Fail-closed by default. Deterministic, auditable decisions.
3. Scale Govern everywhere	Extend the same primitives across clouds, data platforms, SaaS, and developer tools. Federate identity. Stream events to existing infrastructure.	Consistent governance across all surfaces. One policy model, one audit trail, regardless of where agents run.

Benefits for Enterprise Teams

Different roles within the enterprise arrive at the agent governance problem from different directions. Dome is designed to serve all of them through the same platform.

Role	What they need	What Dome provides
AI Developers	Speed. Get agents connected to tools without waiting for manual provisioning. Iterate without bypassing governance.	A complete, queryable picture of the agent estate: what agents exist, what they access, and what they do.
Platform Engineers (Operators)	Operational consistency. A governed paved road for agent deployment that fits existing infrastructure (Kubernetes, serverless, CI/CD).	Registry, lifecycle management, and deployment flexibility (SDK, sidecar, gateway). One operational model across all surfaces.
Platform Engineers (Security)	Identity-based controls, least-privilege policy, and auditable decisions for agent-to-tool communication.	Cedar policy engine. Per-agent, per-tool authorization. Fail-closed. Deterministic evaluation. Complete audit trail.
CISOs	Compliance evidence without manual audit work. Confidence that agent governance meets the same rigor as service-to-service governance.	Exportable record of every governed action: who, what, when, under which policy, with what result. Queryable and auditable.

Conclusion

Every era of enterprise computing has produced a moment where the pace of adoption outran the infrastructure to govern it. Networks produced authentication and directory services. Distributed applications produced API gateways. Cloud computing produced container orchestration and secrets management. In each case, the enterprises that built the platform layer early gained a structural advantage through a combination of the speed at which they could adopt what came next, and consistent methods to operate in those eras.

The agent era is now.

The enterprises that establish consistent governance for agents of all kinds will be the ones that enable their teams to deploy agents fastest and most confidently. They will say yes to business units requesting agent capabilities, because the infrastructure to govern those capabilities already exists.

The enterprises that wait will face the same catch-up they faced with containers and cloud, but with higher stakes. The systems they need to govern are reasoning systems. The gap between deployment velocity and governance capability widens every week. And the cost of closing it later is always higher than the cost of building it now.

The path forward is clear. Define the agent. Map the surfaces. Establish the control points. Build the platform layer. The blueprint in this paper provides the framework. The choice is when to start.

https://domesystems.ai