Infosys topaz

OPERATIONALIZING ENTERPRISE AI AGENTS WITH AMAZON BEDROCK AGENTCORE: HOW CITIZENS BUILT A PLATFORM FOR THE AGENT DEVELOPMENT LIFECYCLE

Krishnakumar Chellappa, Rhishikesh Deosthale | ***Citizens***

Jagadamba Krovvidi, Chetana Amancharla, Paresh Haribhai Mathukia, Virendrasinh Patil | ***Infosys***

Dhiraj Thakur, Abhishek Tiwary | ***AWS***

Infosys
Navigate your next

Introduction

Operationalizing enterprise AI agents requires more than model capability—it demands a disciplined agent development lifecycle, robust governance, and scalable runtime infrastructure. As organizations transition from prototyping to production-grade agentic systems, the challenge shifts toward managing complexity across deployment, security, and lifecycle management.

In this paper, we explore how Amazon Bedrock AgentCore powered the build and deployment of the AI Platform at Citizens, addressing enterprise governance and scalability requirements for agentic AI.

To advance this transformation, Citizens collaborated with Infosys and AWS to design an enterprise-ready approach for agentic AI. Citizens & Infosys co-created architectural blueprints and Agent Development Lifecycle (ADLC) patterns. These patterns shaped decisions across runtime design, gateway-mediated tool access, observability, and governance.

Citizens evaluated how agent-based AI systems could operate reliably at enterprise scale. This effort emphasized repeatable lifecycle controls, secure integration patterns, and governance-first design principles to support production adoption.

The platform implementation combines standardized design-time pipelines with controlled runtime orchestration. Agents are packaged as containerized artifacts, deployed through enterprise CI/CD pipelines, and executed within Amazon Bedrock AgentCore. Tool access is mediated through governed APIs, while identity propagation and policy enforcement are implemented using enterprise IAM integration and centralized audit logging.

The Challenge: Operationalizing Agentic AI in Enterprise Settings

As Citizens explored the adoption of agentic AI for enterprise workflows, several foundational challenges emerged that are common across large organizations but particularly acute for autonomous, tool invoking agents.

Enterprise integration complexity: Agents need to interact with internal APIs, applications, and knowledge systems while enforcing strict access controls.

Runtime deployment validation: Agent workloads need to run in an environment that aligns with enterprise development practices, infrastructure standards, and CI/CD workflows.

Governance and policy enforcement: Agent tool calls and system interactions need to comply with authentication, authorization, and audit requirements.

Observability and operational readiness: Enterprises need to verify trace agent reasoning, tool invocation, and runtime behavior using existing monitoring platforms.

Solution Architecture

Overview

The solution architecture defines a clear trust boundary. It separates design-time and runtime responsibilities. This separation supports governance and scalability.

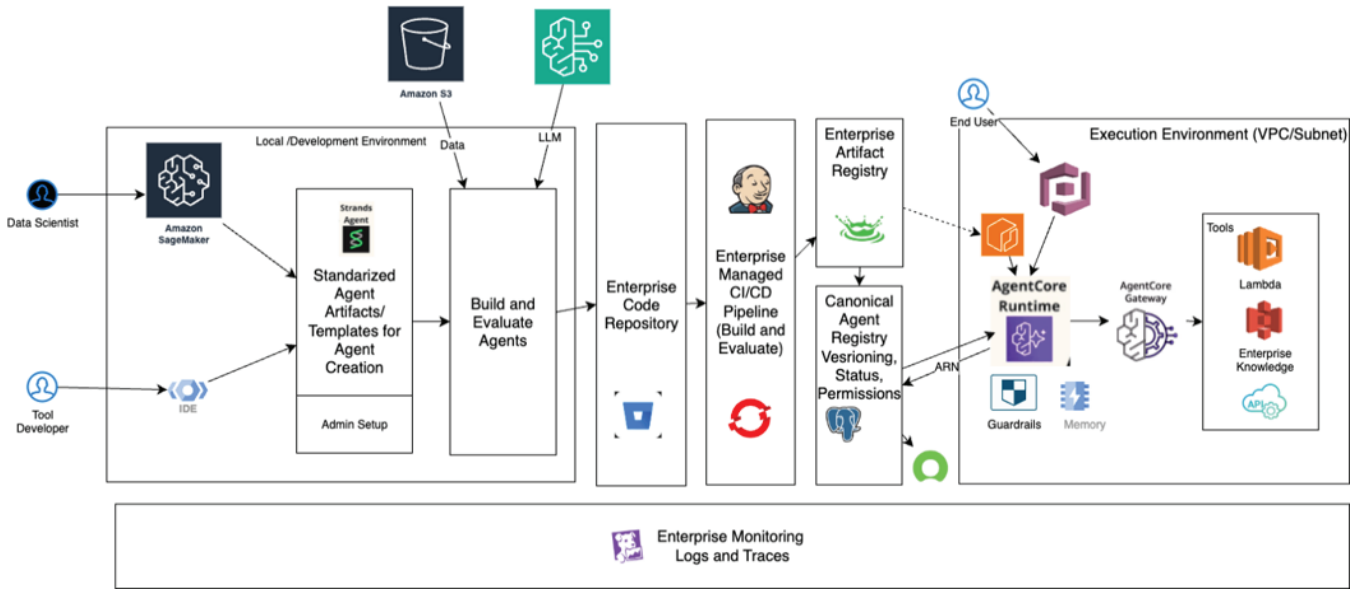


Figure 1: Enterprise Agent Deployment, Registration, and Governance Flow

At a high level, the architecture consists of:

- Design-time architecture, which standardizes how agents are created, validated, deployed, registered, and governed.
- Run-time architecture, which governs how agents execute, reason, invoke tools, and interact with enterprise systems under enforced security controls.

This separation supports enterprises to scale agent development and innovation while maintaining consistent security, governance, and operational oversight.

Design-Time Architecture

The Design-Time architecture spans across four key capabilities.

1. **Standardized Agent Creation:** Agent development follows prescribed templates for agent instructions, tool bindings, and configuration, with explicit declaration of required tools and separation of reasoning logic from environment-specific configuration. This supports predictable agent behavior.

2. **Enterprise CI/CD Integration:** Agent artifacts are fully integrated into existing enterprise CI/CD pipelines, treating agents as first-class software assets. All agent changes are version-controlled, automated validation and security scanning are enforced, and promotion across environments follows formal approval gates.

3. **Canonical Metadata Registration:** Once deployed, each agent registers a canonical metadata record capturing agent identity, ownership, supported capabilities, approved tool access, version, and lifecycle state. This separates the authoritative enterprise definition of the agent from runtime execution details.

4. **Agent Registry and Governance:** Registered agents are published to a centralized enterprise Agent Registry, enabling controlled onboarding, lifecycle state management (active, restricted, deprecated), and visibility for platform administrators and risk teams.

Design-Time Flow: Agent Development to Production Readiness

This section describes the end-to-end design-time flow, showing how an agent progresses from initial development to production-ready state under enterprise controls.

The design-time flow guarantees only validated, governed, and approved agents are eligible for runtime execution, progressing through seven stages.

1. **Agent Authoring** using standardized patterns defining instructions, tools, configuration, and dependencies.

2. **Source Control and Version Management** establishing a single source of truth and full traceability for every change.

3. **CI/CD-Driven Validation** triggering automated configuration, policy, dependency, and security scanning on every commit.

4. **Agent Packaging** into immutable runtime artifacts such as container images, ensuring reproducibility and version isolation.

5. **Environment Promotion and Deployment Approval** across development, testing, staging, and production with policy-driven, auditable gates.

6. **Canonical Metadata Registration** formally establishing what the agent is authorized to do from a governance perspective.

7. **Agent Registry and Governance:** Publishing the agent to the enterprise Agent Registry and Agent Hub for controlled discovery and reuse.

The following table outlines the lifecycle stages, environments, entry criteria, quality gates, and accountability required to ensure that agents progress to production in a controlled and auditable manner—without interacting with real enterprise systems until their behavior and autonomy are explicitly validated.

Stages / Setup	Use Case Identification	Sandbox Development (Stubbed Tools)	QA / Pre Production (Governed Real Tool Access)	Production Deployment & Operation
Environment	Business / Product workspace	Isolated Sandbox (no real tools)	QA / Pre Prod with enterprise trust boundary & tool governance	Production
Key Activities	Identify candidate use cases; define users, autonomy, impacted systems; classify risk; define success metrics & failure boundaries	Build agent logic; stub/mock/simulate all tools; implement reasoning, tool selection, memory, error handling; telemetry hooks	Agent Serving (Real Time, Batch, Event Based), Connect to real tools via governed access; enforce AuthZ,, rate limits, quotas, audit; use masked production like data; E2E workflow tests	Deploy agent; support monitoring, tracing, emergency stop switches, rollback; finalize runbooks; set SLAs/SLOs; ownership transition
Entry Criteria	Portfolio intake open; business problem defined	**G1** approval; sandbox env provisioned; tool contracts stubbed	**G2** approval; connections allowlisted; quotas & audit configured	**G3** approval; production change window
Quality Gate	**G1:** Business Sponsor signoff, Architecture, security, platform review; validate tool scope, data sensitivity, autonomy level	**G2:** Functional verification; adversarial tests (hallucinated actions, policy violations, boundary breaches); trace/log validation	**G3:** Functional, security, compliance sign-off; confirm no policy bypass; validate performance & cost	Monitor drift, failures, policy violations; safe iterations on prompts/tools; re-certify on scope/autonomy changes; decommission at end-of-life
Exit / Gate	Use case proposal documented	Code complete in sandbox; test plan authored	E2E tests complete; non functional tests executed	Production launch completed; handover to Ops
Owners (RACI)	**R:** Product Owner, Use Case Sponsor **A:** Business Owner **C:** Arch/Sec/Platform **I:** Ops, Data Privacy	**R:** Agent Builder **A:** Tech Lead **C:** Platform, Security **I:** Product	**R:** AI Engineer **A:** Tech Lead **C:** Security, Data Privacy, Ops, Agent Builder, Platform **I:** Product	**R:** AI SRE **A:** Service Owner **C:** Engineering, Security, Product, Platform **I:** Stakeholders
Primary Artifacts	Use case brief, value hypothesis, initial risk class, success KPIs, failure/rollback boundaries	Agent code, prompt/orchestration files, stubbed tool contracts, test plan, telemetry schema	Agent Serving Code, Agent Metadata, E2E test scripts & results, performance test, cost profile, access policies, runbooks (draft)	Deployed artifacts, infra as code, dashboards & alerts, runbook, incident playbooks
Quality / Readiness Metrics	Problem solution fit stated; KPIs & risk class recorded; initial privacy review noted	Unit & component pass ≥95%; negative tests implemented; no real system calls present	E2E pass; p95 latency & error budgets met; cost within budget; no unauthorized access	Health SLOs met; error budget tracked; MTTR targets defined; emergency stop switch validated

Table 1: Gated Lifecycle for Enterprise Agent Development

Run-Time Architecture

The run-time architecture spans across four structural components.

1. **AgentCore Runtime:** Agents execute within the Amazon Bedrock AgentCore serverless runtime, which provides isolated execution of agent sessions, context and memory handling, integration with foundation models via Amazon Bedrock, and enforcement of agent-aware policies. Isolation securely separates state, memory, and reasoning across users and invocations.

2. **Identity Enforcement:** Each deployed agent operates with a unique, verifiable identity established at runtime using federated identity mechanisms. This provides least-privilege access to tools and services, full traceability of agent actions, and alignment with enterprise IAM and audit requirements.

3. **Trust Boundary Enforcement:** A clear trust boundary separates agent reasoning from enterprise system access. At this boundary, requests are authenticated and authorized, traffic is inspected and rate-limited, and only policy-compliant requests proceed. Agents are configured not to directly invoke internal services or APIs, ensuring autonomy never bypasses enterprise security controls.

4. **Tool Invocation Governance:** Beyond the trust boundary, agent-aware governance enforces discovery of only explicitly approved tools, action-level policies constraining how tools may be used, and centralized logging of all tool interactions.

Run-Time Flow: Agent Invocation and Operations

During live operation, the end-to-end runtime flow proceeds through following steps.

1. **Agent Invocation** by a business application or workflow, with lifecycle state validated and invocation bound to the canonical agent definition.

2. **Agent Execution** within the AgentCore serverless runtime, interpreting requests using the selected foundation model within secure, ephemeral isolation.

3. **Policy-Aware Reasoning and Tool Selection**, where only approved and registered tools are discoverable and action-level constraints apply.

4. **Enterprise Trust Boundary Enforcement**, applying authentication, authorization, traffic inspection, rate limiting, and centralized audit logging.

5. **Agent-Aware Tool Governance**, enforcing tool contracts (MCP or REST), input/output schemas, action-level policies, and identity propagation.

6. **Tool Execution** against governed enterprise services including internal APIs, data platforms, workflow systems, and approved SaaS integrations—all fully traceable.

7. **Response Synthesis**, where tool outputs are combined with model reasoning and output controls such as formatting, validation, or redaction are applied.

8. **Response Delivery** to the invoking application or user interface, with all internal governance remaining transparent.

9. **Observability and Operational Feedback** through continuous telemetry including reasoning traces, tool invocation logs, and performance metrics, integrated with Amazon CloudWatch and enterprise observability platforms.

Key Outcomes for Citizens

Citizens achieved several tangible operational improvements through the implementation of the AI platform using AWS Bedrock core infrastructure.

Standardized agent development lifecycle: Agent creation now follows consistent templates, design patterns, and CI/CD workflows. This makes agents easier to build, review, and maintain across teams.

Improved maintainability and predictability: By enforcing structured design-time and runtime patterns, agents exhibit more consistent behavior. This simplifies updates, reduces variability, and supports long-term maintainability.

Built-in governance compliance: Data scientists and AI engineering teams can now develop agents that align with enterprise governance requirements by design. Policy enforcement, access control, and audit requirements are integrated into the development lifecycle rather than applied post-development.

End-to-end traceability and observability: All agent actions, including reasoning steps and tool interactions, are logged and traceable. This makes it significantly easier to diagnose issues, debug failures, and validate agent behavior in production environments.

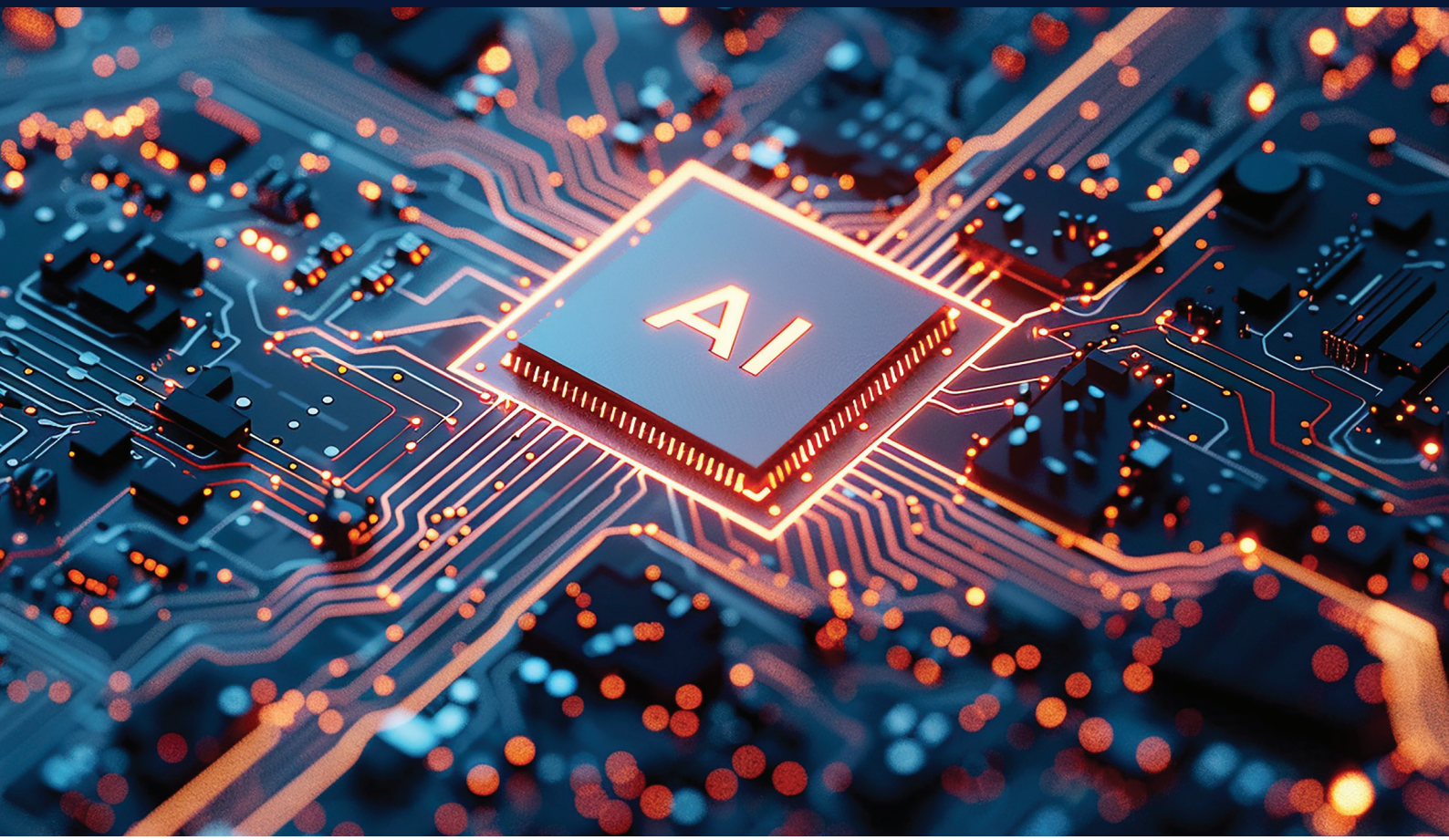
Reduced operational ambiguity: Centralized agent registration, metadata management, and lifecycle controls provide clear visibility into agent ownership, capabilities, and lifecycle state.

Foundation for scalable agent adoption: The platform establishes repeatable patterns that enable multiple teams to build and deploy agents consistently, accelerating enterprise-wide adoption of agentic AI.

Conclusion

As enterprises move beyond AI experimentation toward production-grade agentic systems, having a structured approach to agent development, deployment, and governance becomes essential.

The partnership between Citizens, Infosys, and AWS demonstrates how Amazon Strands, Amazon Bedrock AgentCore, combined with a well-defined Agent Development Lifecycle, can provide a repeatable, secure, and scalable foundation for enterprise agentic AI.



For more information, contact askus@infosys.com

Infosys
Navigate your next