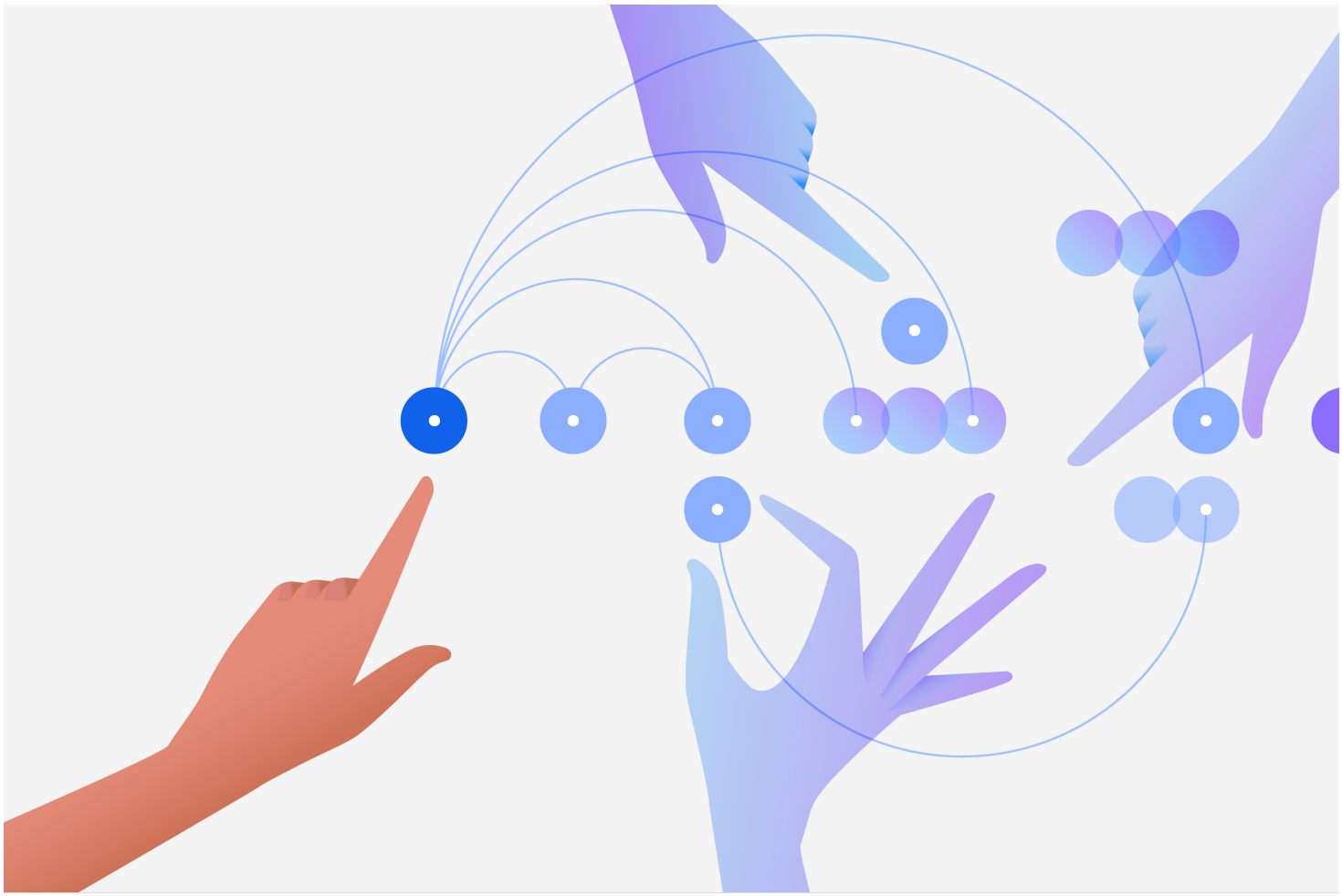
Architecting secure enterprise AI agents with MCP

Verified by **Anthropic**, October 2025



IBM

TABLE OF CONTENTS

AI agents powered by large language models (LLMs) require enhanced application development lifecycles that address the unique nature of agent development. Unlike static applications, agents are adaptive, interactive systems that must be continuously evaluated, secured, governed, and improved due to the nondeterministic and probabilistic nature of underlying LLMs. For example, a traditional software development cycle (SDLC) would deploy an agent to production after successful staging tests. However, the same agent tested with identical data can produce different results due to inherent variability. This behavior necessitates different testing and validation approaches—a key differentiator from the application development lifecycle.

This guide presents the **agent development lifecycle (ADLC)**, a structured approach to designing, deploying, and managing enterprise AI agents. At its core, it is an operational discipline based on standard DevSecOps practices that ensures agents remain safe, reliable, secure, and aligned with organizational and regulatory goals (such as compliance with AI Regulations).

WHAT ARE AI AGENTS?

AI agents are software systems that perceive context, reason over goals and constraints, and act through tools and services to complete multistep tasks. Effective agents combine capabilities such as memory, planning, tool use, and reflection. They can be made to operate within explicit authority boundaries, leverage tools to escalate when needed, and provide observable traces of their decisions and actions.

Agents are more than their external interfaces: they are decision-making and execution systems that integrate with data and applications through tools. They can break down tasks into smaller subtasks, execute and coordinate each, and deliver outcomes.

WHAT ARE ENTERPRISE AI AGENTS?

Enterprise AI agents are autonomous or semi-autonomous AI agents designed to perform tasks in a business environment. They bring agentic AI capabilities into real organizational environments with enterprise requirements for observability, security, compliance, resilience, and scale. Enterprise AI agents can run in non-hybrid as well as hybrid cloud deployments—public, private and enterprise data centers. They can also work with a mix of model types—small, large, and fine-tuned

enterprise-specific—depending on their quality, latency, and cost needs. These agents can also access governed enterprise data and execute actions through tools. Enterprise agents can be observed, secured, governed, and audited across their full lifecycle to meet business, risk, and regulatory expectations.

THE PARADIGM SHIFT

Enterprise AI agent projects succeed when organizations recognize the fundamental differences between traditional software and agentic systems:

From deterministic to probabilistic: Traditional software follows predictable execution paths, while agents make dynamic decisions that can vary even with identical inputs.

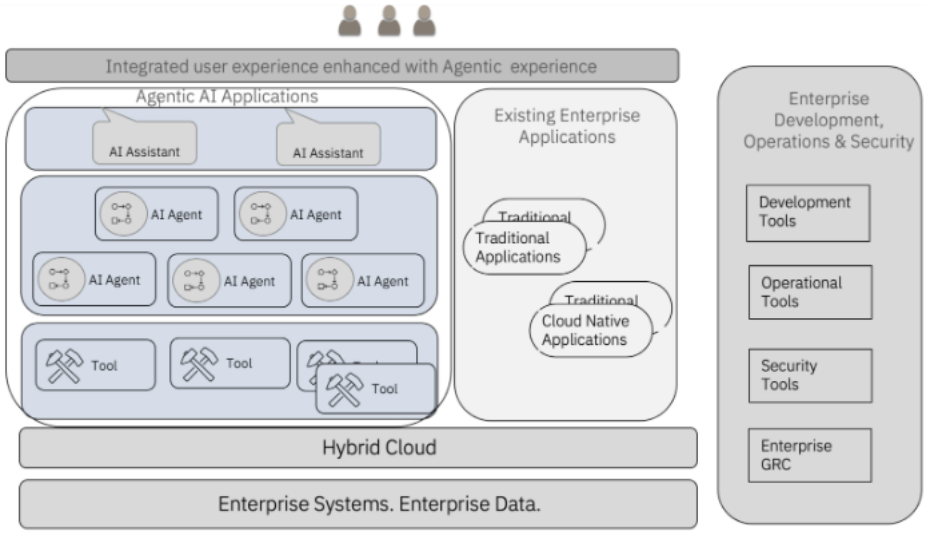
From static to adaptive: Applications typically have fixed functionality, but agents can learn and evolve their behavior based on interactions and feedback.

From code-first to evaluation-first: Traditional software metrics don't predict agent success. You can have a perfect implementation that results in poor agent behavior, or messy prompts that work great. Success depends less on implementation details and more on systematic measurement of agent behavior and business outcomes.

AGENTIC ENTERPRISE

As enterprises embrace agentic AI to drive autonomous decision-making and intelligent automation across enterprise development, operations and security, it is imperative to recognize that these agents do not operate in isolation. The introduction of agentic AI into enterprise workflows demands a rethinking of traditional IT processes to enable the transformation to an **agentic enterprise**.

The ADLC must be designed to coexist and integrate seamlessly with the broader enterprise ecosystem—spanning hybrid cloud infrastructure, enterprise data platforms, business applications, and existing management tools for development, operations, and security. These agents developed with the ADLC must be context-aware, interoperable, and governed by the same architectural principles that underpin enterprise IT.



Whether deployed in cloud-native environments or on-premises systems, tools and context that agents consume must respect enterprise controls, resilience requirements, risk-based security, data sovereignty, and regulatory constraints. When

enterprises define any additional controls around the agents themselves, the deployment should also respect those controls. This approach calls for a hybrid deployment model where agents that are embedded within existing continuous integration or continuous development (CI/CD) pipelines, observability frameworks, and security controls are *tightly constrained, permissioned, and sandboxed*—ensuring continuity, security, compliance, and operational resilience.

ENTERPRISE AGENT DEVELOPMENT REQUIREMENTS

Enterprise agent development in regulated industries—such as healthcare, telecommunications, and finance—requires a fundamentally expanded approach beyond traditional software practices. These sectors demand rigorous compliance, security, and operational standards that shape a convergent set of requirements across the agent lifecycle. In the **planning phase**, organizations must generate synthetic ground truth data due to limited access to historical records. Early design of enterprise-grade access control (RBAC) and robust vendor evaluation frameworks are essential to guide stack selection and build-versus-buy decisions.

During the **code and build phase**, enterprises need advanced experiment tracking, version control for multiple agent variants, and embedded evaluation frameworks to ensure performance and reliability. Development environments must support multitool orchestration, secure integration with existing systems, and specialized vulnerability scanning that goes beyond traditional methods to detect prompt injection and adversarial threats. As agents move into testing and release, automated evaluation frameworks integrated into CI/CD pipelines become critical for monitoring behavioral drift, accuracy, and cost. The infrastructure must accommodate the stochastic nature of AI agents and enforce governance through certified catalogs and secure API execution.

In the **deployment phase**, compliance-aware infrastructure tailored to industry-specific regulations (for example, HIPAA, financial standards) is required, along with scalable, low-latency SaaS platforms and secure integration into hybrid enterprise environments. *Sandboxing* is a foundational security control for enterprise AI agents. Unlike traditional services, agents often execute dynamically generated code, interact with heterogeneous tools, and handle sensitive enterprise data. Without isolation, a compromised or misbehaving agent can access resources far beyond its intended scope. Finally, the **monitoring and operations phase** demands deep observability into agent reasoning, unified dashboards for business and technical metrics, and continuous compliance monitoring. Centralized governance registries help prevent agent sprawl and ensure secure, auditable agentic operations across the enterprise.

ENTERPRISE AGENT DESIGN PRINCIPLES

By embedding agentic AI within the fabric of enterprise architecture, organizations can unlock transformative agility, reduce cognitive load, and accelerate innovation—without compromising trust, compliance, or control.

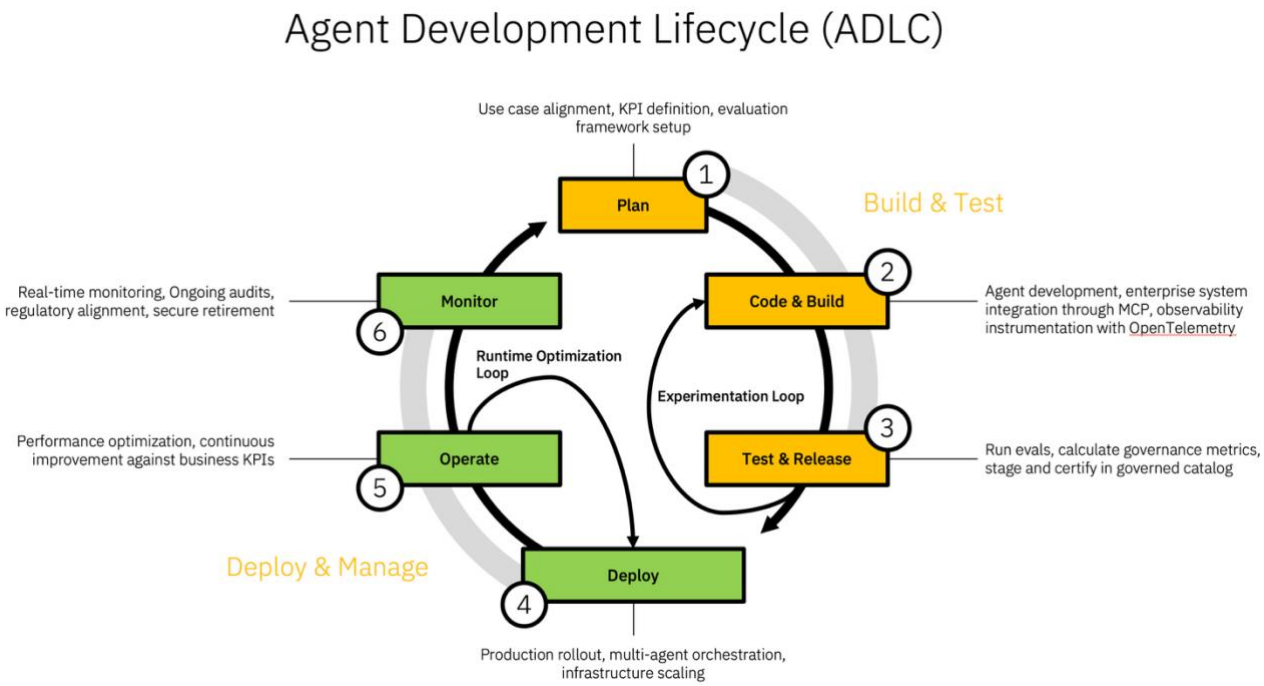
Following these design principles enables transformation to an agentic enterprise:

1. Enterprise agents are designed to have acceptable agency.
2. Enterprise agents are interoperable, secure by design, and integrate with enterprise tools.
3. Enterprise agents are evaluated to meet business objectives while mitigating risk.
4. Enterprise agents are securely deployed across hybrid cloud and AI systems.
5. Enterprise agents are continuously observed and managed to be resilient.
6. Enterprise agents are governed to meet enterprise risk and compliance.

These design principles need to be woven into the agent development lifecycle to enable collaboration across enterprise teams and seamless process integration into existing enterprises processes.

THE AGENT DEVELOPMENT LIFECYCLE: DEVSECOPS PRACTICES FOR AI AGENTS

The ADLC provides a comprehensive framework organized around six interconnected phases, building on the standard DevSecOps practices, but extending each to address the unique challenges of agentic systems:



The fundamentals of DevSecOps apply to AI agents, but with some required extensions. For example:

- **Shift-left security**: must now be applied to agentic identity management and data access through MCP.
- **Automation**: must be extended to include agentic evaluation and benchmarking.
- **Continuous monitoring**: needs to capture and analyze agentic reasoning traces and agentic tool usage.
- **Collaboration**: must include the crucial engagement of business stakeholders in the plan, build, and test phases.
- **Compliance and risk management**: must address the new risks created by agent autonomy.

AI agents inject stochastic control logic (agentic reasoning) into applications that previously relied on static control flow. This behavior requires two new inner loops in the lifecycle to test, tune, and optimize the agentic workflow. First, the Experimentation Loop between Build and Test incorporates agent evaluation frameworks and benchmarking to drive build time improvement of agentic behavior. Second, the Runtime Optimization Loop allows for inference-time scaling and continuous optimization of both agent quality and operational costs. In addition, extensions are required at each phase of the ADLC:

- Planning phase now requires agentic behavior specification and documentation of standard operating procedures in natural language.
- Code and Build phases now include prompt design and agent orchestration.
- Test and Release phases include agent evaluation.
- Deployment must deal with behavioral guardrails for agents and multi-agent scaling.
- Operations must include observability of agentic reasoning traces.
- Monitoring now includes accuracy of agent journeys, hallucination metrics, behavioral drift, and new cost and latency metrics.

PLAN

The process begins with **use case alignment** and defining the agent's purpose and key performance indicators (KPIs). Identify business outcomes (for example, customer support automation, financial compliance, or R&D copilots). While all agents have agency, enterprises should select use cases that enable productivity, while meeting risk appetite around acceptable agency. This phase ensures that agents are scoped around high-value, measurable goals rather than generic functionality.

Evaluation-first: Define business key performance indicators (KPIs) including accuracy, latency, trust scores, safety thresholds, and security requirements. Continuously measure behavior and outcomes. Ship only with evidence.

Real agency, clear limits: Give agents the capabilities to own outcomes (context, memory, planning, tools) with explicit authority bounds and human-in-the-loop paths. Capture full traces to enable reversal and improvement.

CODE AND BUILD

With requirements set, teams move into core development. Implement prompts, memory strategies, and orchestration logic. Integrate agents with enterprise systems, APIs, and external knowledge bases. Instrument observability hooks to capture agent transcripts (which include agent reasoning traces, tool calls, and outputs).

Hybrid models by design: Use a portfolio of models (frontier for hard reasoning; domain, small, on-prem for cost, latency, or locality). These models could also be consumed as a service (for example, frontier models delivered from cloud), and enterprise-specific models could be consumed from on-premises deployments.

Hybrid cloud by design: Assume hybrid cloud deployments and integrations when accessing tools and context and where the agents are deployed.

Interoperable: Prefer open standards. Use MCP for tools, resources and prompts with typed schemas; adopt consistent patterns when it comes to context storage and retrieval, tool access, and task delegation.

Tool-centric: Treat integrations to enterprise data, applications, and systems as tool integrations, enabled by MCP servers. Keep tools least-privilege, versioned, and well-documented. Use an **MCP Gateway** pattern to secure and governed connections to enterprise backend systems.

Secure-by-design: Develop agents by using secure-by-design principles to meet enterprise security requirements. Issue identities to agents so that every action taken by agents by themselves, or when acting for users, are traceable and auditable. Implement security controls that take a risk-based approach. Propagate context for attribution, log immutable decision, tool and complete trails, and preserve lineage for prompts, models, tools, and data.

Sandbox agents and tools to constrain their execution environment and limit capabilities such as network or filesystem access to the minimum necessary. In practice, this means running agents and tools inside lightweight isolation frameworks (for example, Firecracker, gVisor, container security profiles) to enforce boundaries and prevent lateral movement. Sandboxing should be complemented by a runtime policy at the MCP Gateway layer, which can enforce rate limits, throttling, and outbound access controls centrally across agents and tools. Together, infrastructure-level isolation and gateway-level governance provide layered defense: the former ensures that compromised agents and tools remain contained, while the latter regulates interactions with external systems and shared resources.

TEST, OPTIMIZE, RELEASE

Testing agents requires more than traditional unit tests—it involves behavioral validation. Run structured evaluations against predefined benchmarks. This approach includes having test data sets that mimic the agent production data. Optionally, it also includes ground truth information on the trajectory that should be followed by the agent for answering each input. Measure governance metrics such as hallucination rate, bias, and compliance with policies. There are different strategies to compute these metrics. For example, hallucinations can be computed by comparing the LLM generated content with the context fed to the LLM. Similarly, there are guardrail models to check for bias in the LLM input and output.

Continuous evaluation and guardrails. Agents must be tested against evolving benchmarks and policy checks throughout their lifecycle, with explicit guardrails on acceptable agency. Leverage LLM-as-a-Judge (LLM-aaJ) and human-in-the-loop review to continuously measure quality, safety, and compliance to ensure that agents remain aligned to business and regulatory objectives.

Perform security testing and red teaming exercises. Evaluate risks and mitigate them before deployment. Certify agents are in a governed catalog to ensure that they meet enterprise and regulatory standards before release. This phase reduces risk by combining evaluation with governance practices.

DEPLOY

Once certified, agents are deployed into production environments.

Hybrid deployment. Plan for enterprise data and application integrations across hybrid cloud environments, hybrid data sources, and hybrid AI stacks. Adopt a gateway pattern to enable improved governance and effective enforcement.

Resilient to achieve business continuity. In this phase, agents get securely deployed into enterprise hybrid cloud environments, integrated with enterprise data and applications**. Roll out progressively to manage risk**. Design to ensure that they are resilient to outages, resilient to cyber-attacks, and resilient to agentic drift. Include a kill-switch to disable the agent under emergencies.

Support multi-agent orchestration where different agents collaborate on workflows. Scale infrastructure to handle demand while enforcing safety and cost controls. ADLC ensures runtime governance through sandboxing, versioning, rollback strategies, security enforcement and performance throttling.

Sandboxing: Sandboxing refers to running agents and their tools inside constrained execution environments that strictly limit their capabilities. These environments enforce least-privilege access to compute, storage, network, and system APIs. Sandboxing should be treated as a **baseline control**, not an optional feature. By containing execution at the infrastructure level and governing interactions at the gateway, enterprises achieve defense in depth—ensuring that even if an agent misbehaves, its blast radius remains tightly constrained.

Sandboxing should be used when deployments include untrusted or dynamic code execution, for example, code generation, data transformation, and tool orchestration across multiple trust domains (where one tool's misuse could affect another). Other examples include high-value or sensitive data handling (where accidental exfiltration must be prevented) and multitenant deployments (where different business units, customers, or projects share infrastructure).

Implementation strategies for sandboxing include:

- Lightweight virtualization
- Container security profiles (for example, apply seccomp, AppArmor, SELinux policies to restrict system calls, capabilities, and filesystem access)
- Network controls (for example, disable or tightly scope outbound and inbound connections and route all external access through an MCP Gateway for rate-limiting, throttling, and auditing)
- Filesystem policies (for example, mount ephemeral or read-only volumes and block direct access to secrets, logs, and host files)
- Gateway-level enforcement (for example, combine infrastructure sandboxing with centralized MCP Gateway policies to enforce throttling, access approvals, and compensating actions)

MONITORING AND RUNTIME OPTIMIZATION LOOP

After deployment, continuous monitoring keeps agents reliable, effective, and secure. Track real-time metrics such as accuracy, latency, cost, and user satisfaction. Detect drift in behavior or performance regressions.

Observed and managed: Combine rich traces, metrics and evaluations, SLOs and error budgets, circuit breakers, graded responses, and RCA-driven remediation across prompts, tools, and policies.

Agents have unique security threats including memory poisoning (injecting malicious data into agent memory) and tool and API misuse (manipulating agent to use a trusted tool to perform unauthorized actions). Other threats include intent breaking and goal manipulation (tweak prompts to hijack an agent's purpose) among others.

Optimize prompts, tools, and memory policies based on feedback. This phase establishes a closed feedback loop where insights from monitoring drive improvements in agent design and retraining.

OPERATE

The final stage focuses on operations, governance, and oversight.

Governed and discoverable:

Run a curated catalog of agents and tools with:

- Owners: For better accountability and escalation
- Versions: For a better change management practice
- Risk posture: To make decisions on which tools can be used with which business applications
- Environments: To have operational oversight
- Auditability: To link evidence such as evaluations, red teaming, and approvals

Conduct ongoing audits for fairness and transparency. This approach includes checking for fairness in the agent's output across different demographic groups, as well as fairness in performance and accuracy metrics across these groups. Conduct security risk and regulatory compliance tests. Align with evolving industry standards and legal requirements. Securely retire or decommission agents when they are no longer needed. This phase includes securely handling and archiving agent input and output data that contains sensitive or confidential content as well as storing the agent code for future reference. The ADLC ensures that security and compliance are not a one-off step but a continuous discipline throughout the lifecycle.

Each phase should produce deliverables and have a gate:

- **Use case**: charter, KPIs, risk appetite (Gate: leadership approval).
- **Development**: prompts-as-code, tools, memory policy (Gate: design review passed).
- **Testing**: evaluation suite results, red team report, compliance checklist (Gate: quality and risk thresholds met).
- **Deployment**: rollout plan, feature flags, rollback or kill-switch (Gate: change advisory approval).
- **Monitoring**: dashboards customized to the agent use case, SLOs and SLA-based alert thresholds, alert runbooks (Gate: operational readiness confirmed).
- **Review:** metrics computed by using audit logs, data lineage, decommission plan (Gate: compliance review closed).

The ADLC moves agents from idea to production with DevSecOps at the core. It emphasizes continuous evaluation, monitoring, security, and governance. This strategy enables organizations to scale agents safely and confidently by keeping them trustworthy, auditable, and aligned to business value.

ENTERPRISE CONSIDERATIONS BUILDING AI AGENTS

WHEN TO BUILD AGENTS

We recommend finding the simplest solution that addresses your specific business need. This might mean not building agentic systems at all—many problems can be solved more effectively with traditional automation, well-designed prompts, or retrieval systems. For example, a system that classifies customer emails into categories can be built with a well-designed prompt and doesn't require an AI agent. However, a system that automatically answers customer emails would benefit from an agentic retrieval-augmented generation (RAG) approach.

Agentic systems often trade latency and cost for improved task performance. Consider whether this tradeoff makes sense for your specific use case. The most successful implementations begin with these characteristics:

Well-defined problem scope: Successful implementations focus on specific, measurable business problems rather than broad automation objectives. Problems requiring contextual judgment, complex decision-making, or multistep reasoning are good candidates for agentic approaches. For example, an agent can conduct comprehensive research for a business strategy report. This example involves multistep reasoning: gathering data from internal and external sources, analyzing it, drafting the report, and refining the final output.

Clear success metrics: Teams that succeed establish quantitative success measures before any development work begins. These metrics should reflect business value—customer satisfaction improvements, processing time reductions, or error rate decreases—rather than purely technical measures.

Manageable complexity: Start with single-agent solutions that handle defined tasks. Multi-agent orchestration introduces operational complexity that often outweighs theoretical benefits until you've established competence with simpler systems.

Enterprise integration requirements: Plan for observability, security, resilience, and compliance requirements during initial design. These constraints fundamentally shape agent architecture and shouldn't be treated as implementation details.

PROVEN APPLICATION AREAS

Enterprise deployments show three patterns with consistent value and manageable risk:

Customer support and service: Mature use case that combines conversational interfaces with clear metrics. Agents resolve routine inquiries while escalating complex cases. Factors critical for success: having strong evaluation criteria, knowledge base leverage, and clear human-handoff rules.

Document-heavy processes: High value in contracts, compliance, and research workflows. Agents route, validate, and synthesize across multiple sources. Factors critical for success: having structured inputs, verifiable outputs, encoded business rules, and measurable accuracy gains against existing approaches.

Knowledge work and development augmentation: Agents assist with research, analysis, documentation, and testing. Success factors: quality feedback loops, defined task standards, and measurable productivity improvements.

STRATEGIC IMPLEMENTATION CONSIDERATIONS

SECURITY AND RISK MANAGEMENT

Enterprise agentic systems introduce unique security challenges that traditional application security doesn't address:

Autonomous decision-making risks: Agents make independent decisions that can impact business operations, customer relationships, and regulatory compliance. Unlike traditional applications with predictable execution paths, agents require security frameworks that can govern dynamic, context-dependent behavior.

Expanded attack surfaces: Conversational interfaces, prompt injection vulnerabilities, and autonomous tool usage create attack vectors that traditional security tools don't address. Enterprise implementations must include agent-specific security controls alongside conventional protections.

Compliance complexity: Regulatory frameworks often struggle with autonomous decision-making systems. Financial services must audit agent decisions, healthcare systems must protect patient data in conversational contexts, and public sector applications must ensure transparent automated decisions.

BUSINESS VALUE REALIZATION

Successful enterprise agent implementations focus on measurable outcomes rather than technological sophistication:

Process enhancement vs. replacement: The highest return on investment (ROI) comes from agents that enhance existing business processes rather than attempting to replace them entirely. This approach reduces change management complexity while providing clear value measurement opportunities.

Quality and consistency gains: Agents excel at applying complex rules consistently across large volumes of work to reduce human error in areas like compliance checking, data validation, and process adherence.

Operational scalability: Unlike human workers, agents provide immediate scalability for demand spikes, continuous availability, and consistent performance standards—creating value particularly in customer service, processing pipelines, and monitoring applications.

AGENT OBSERVABILITY AND OPERATIONS

Agent observability and agent operations form a tightly interwoven framework for building, deploying, and managing agentic AI:

- **Agent observability** provides critical visibility into agent behavior through rich telemetry, enabling teams to understand how agents perform in real-world conditions.
- **Agent operations** uses that telemetry to manage, adapt, and ensure resilience of agents in production, and feeds learnings back into ADLC.

WHY OBSERVABILITY CHANGES

Unlike deterministic software, agents interpret unstructured inputs and rely on large language models (LLMs) that can produce non-deterministic outputs, even from identical prompts, making reproducibility and traceability challenging. Their reasoning often involves emergent behaviors and tool usage that must be captured beyond just final outputs.

Additionally, agentic systems operate across multiple turns, modalities, and agents, which requires observability to track evolving context and interactions. Crucially, the paradigm shifts from “is it up?” to “is it right?”, where incorrect, biased, or hallucinated outputs pose operational and security risks even when systems are technically performant. This new reality demands a rethinking of observability frameworks.

- **Non-determinism**: identical inputs can produce different outputs; reproducibility needs rich traces.
- **Multiturn, multi-agent**: behavior emerges across steps, tools, and agents; context evolves.
- **Tooling side effects**: correctness depends on tool calls, parameters, retries, and errors—not just final text.

BUILDING BLOCKS

To achieve enterprise agentic AI observability and operations, there are key building blocks that enterprises would need to put in place:

- **Telemetry coverage**: traces, logs, events; inputs/outputs; token and cost accounting; tool calls; safety flags.
- **Holistic MELT**: agent-specific metrics in context of platform metrics (metrics, events, logs, traces end-to-end).
- **Analytics platform**: pluggable analytics for advanced metrics, investigations, recommendations, optimizations, and the resulting evaluations, insights, recommendations, and automated fixes that work across frameworks.
- **Open telemetry ingestion**: support telemetry from heterogeneous apps and platforms to enable broad adoption.

EVALS IN PRODUCTION AND DEVELOPMENT

Evaluating how agents perform is an important element of agent observability for enterprise teams to measure and monitor. These include eval types and metrics:

- **Offline evals**: during build and CI to benchmark behavior and regressions.
- **Online evals**: in production to continuously measure quality, safety, and business outcomes.
- **In-the-loop evals:** invoked at runtime to guide agent decisions or route flows. For example, in an agentic RAG application, compute the context relevance of the fetched context to decide if that should be used to generate an answer.

Metrics and evaluation framework should comprise of:

- **Quality metrics**: for example, task success percentage, groundedness, and tool-call success rate.
- **Safety metrics**: for example, jailbreak rate, sensitive data leakage rate, and policy violations. Identifying such safety issues is challenging but there has been increased focus in this field to detect such kind of issues by using LLMs.
- **Operations metrics:** for example, latency, token or cost per task, cache hit rate, and error classes.
- **Business metrics:** for example, satisfaction scores, cost per outcome, and value delivered; optional composite trust and alignment scores derived from multiple signals.

INSIGHTS, ROOT-CAUSE ANALYSIS, AND REMEDIATION

An agentic flow is the sequence of reasoning steps, actions, and interactions an agent uses to accomplish a task. Enterprise insights into agent performance should be derived from analyzing evaluation outputs and computed metrics at the agentic flow level. Agentic insights could span:

- **Multi-flow insights**: aggregate signals across flows to surface top problem themes. For example, in the past one week, the agents answer quality dropped on last Thursday. This insight has been derived across all the flows of the agent over the past week.
- **Single-flow insights**: analyze a specific conversation or trace for issues and opportunities. For example, the user inquired about health plans for veterans, but the agent failed to retrieve veteran-specific information from the vector database. This forced the user to ask clarifying questions, leading to frustration. This insight comes from a multiturn conversation between the user and agent.
- **Message-level insights**: deep-dive on a single turn and its contributing factors. For example, the user gave a thumbs down due to the agent's slow response time. This insight is based on a single user message.
- **Task and trajectory views**: purpose-built views to inspect a unit of work and its end-to-end path across tools and decisions.

Such insights can then be applied to determine root cause when things go wrong, or to take remedial actions. Root cause analysis would include correlating failures to tools, prompts, models, or dependencies. Based on identified root case, remediation could include recommendations and optionally automate fixes with approvals.

AGENT OPERATIONS AND OBSERVABILITY AT BUILD TIME

Enterprises should plan for observability and agent operations right at build (agent development) time. This strategy includes various dimensions:

- **Evaluation framework**: make behavior measurable by default; capture traces in dev and CI.
- **Prompt, tool, and model optimization**: compare variants to find better accuracy, latency, and cost trade-offs.
- **Experiment tracking**: compare versions; store decisions and rationale on why a specific version was selected, record dataset used to test each version, record prompt versions, model version, model params, toolset versions, environment and configuration, code commit, eval suite version; compare variants with lineage retained.
- **Champion–challenger**: validate candidate improvements against current production baselines across metrics.
- **Lifecycle management**: catalog versions, ownership, risk posture, and why a version was promoted.

ROLES AND RESPONSIBILITIES

Agentic systems disrupt boundaries between development, testing, deployment, and operations. Their dynamic, unpredictable nature requires all roles to adapt:

- **Developers**: instrument traces; design prompts and policies; iterate with evaluation data instead of intuition alone.
- **Testers**: validate intermediate states and routing decisions, not just final outputs; extend testing post-deployment.
- **Site reliability**: monitor trends, investigate drift and anomalies, drive RCA and mitigations with guardrails and runbooks.
- **Business owners**: track business metrics and trade-offs (cost, latency, quality); run A/B and what-if analyses.

AGENT SECURITY

Agentic AI architectures expand the attack surface (conversational interfaces, prompts, new protocols like MCP) and introduce unpredictable agent behavior. Because they tightly couple perception, reasoning, memory, planning, and action with external tools in continuous loops, compromising any component (protocol, prompt, API, or tool) can cascade across the system.

These architectures create dynamic, system-wide technical risks where one breach can trigger broad failure. Such technical risks create four critical business-level challenges:

- **Uncontrolled agent access and privilege escalation:** Agents autonomously escalate privileges to bypass approval and tracking, which raises accountability gaps.
- **Agent-enabled data leakage and prompt exploitation:** Agents might share sensitive data through malicious prompts, through nondeterministic responses that leak information during interactions.
- **Autonomous attack amplification:** Malicious agents outpace human defenses, coordinate rapid distributed attacks, and compress timelines for compromise.
- **Agentic drift and noncompliance:** Autonomous agents adapt and drift from policies undetected; gradual algorithmic shifts evade monitoring, which undermines compliance. While undetected drift could be considered a monitoring concern that could escalate into a security issue through one of the above three items, continuous compliance monitoring is also becoming a key enterprise focus area.

SECURITY SOLUTION FRAMEWORK

Enterprise agentic AI deployments require security approaches tailored for autonomous, unpredictable environments and expanding attack surfaces based on agentic interactions. Traditional tools built for static and predictable systems are necessary but not sufficient. Enterprises should strategize and plan for four targeted solutions to add to enterprise security solutions, each mapped to a core business risk:

- **Agent identity and access**: Assign unique credentials, enforce just-in-time access, factor in context-aware access controls, and maintain continuous audit trails for accountability.
- **Agent and data protection**: Adopt gateway patterns (for example, MCP gateways) to filter prompts for injections, monitor information flow, and track abnormal data sharing. Create secure deployment environments to isolate deployments and harden agent deployments.
- **Autonomous agent defense**: Continuously detect agent threats by proactive threat hunting. Leverage AI-based approaches to determine what protections should be applied to mitigate those attacks and achieve rapid containment.
- **Agent security risk and compliance**: Assess security risks to agentic systems and factor those into enterprise risk decisions throughout ADLC and enterprise process. Continuously monitor for configuration drifts and access patterns for security compliance.

RISK MANAGEMENT AND COMPLIANCE

Enterprises must expand their enterprise and third-party risk assessment policies and processes to include the agentic AI system supply chain, which can include third-party agents, tools (including MCP servers), and AI models:

- SBOMs for tools, prompts, agent code, and components.
- Artifact signing and verification before deployment; trusted registries only.
- Dependency scanning for servers, plugins, integrations; pin versions.
- Scoped tool permissions and approvals; enforce least privilege by default.
- Continuous audits for fairness, transparency, and security.

GOVERNANCE: TEST, CERTIFY & CATALOG

Governance provides the structures, controls, and evidence needed to safely scale agentic systems. It operationalizes certification, cataloging, lifecycle decisions, and runtime policy enforcement across the ADLC.

GOVERNED CATALOG

Registration: Record agent purpose, owners, environments, and data and classification boundaries.

Capabilities: Enumerate tools, resources, prompts, and external dependencies with versions.

Risk posture: Document threat model, risk appetite, and mitigations per release.

Policies:

- **Authority boundaries**: What the agent can and cannot do autonomously; approvals for high-impact actions.
- **Data handling**: Classification, minimization, masking and redaction, retention, and consent.
- **Auditability:** Immutable decision and audit trails including tool calls and side effects.

Evidence: Link eval results, red team reports, approvals, and audit artifacts.

CERTIFICATION WORKFLOW

Prerelease checks: Quality, safety, and compliance thresholds met; security testing passed; approvals captured.

Promotion gates: Feature flags, rollout plan, rollback and kill-switch, change tickets.

Runtime attestations: Artifact signing and verification, SBOMs for tools, prompts, code.

EXPERIMENTATION TRACKING AND LINEAGE

Goal: Capture metadata about the experiments that led to the creation of the new version of an agent. Several experiments are run before a version is selected or identified.

Run metadata: Capture dataset, version or hash, prompt version or hash, model version or hash, decoding parameters, toolset versions, environment or configuration, code commit, and eval suite version.

Lineage graph: Link experiments to candidates to releases; retain decisions and rationale for promotion or rollback.

Replayability: Store seeds, prompts, and inputs for deterministic replays where possible; include trace IDs for selective replay.

Governance link: Attach top candidate runs (and their artifacts) to the catalog entry when proposing a release; include thresholds, deltas versus champion, and risk notes.

Reproducible manifest: Signed manifest with fixed versions and identifiers for agent, tools, prompts (hash), model (ID or hash), datasets or fixtures, and dependencies.

VERSIONING AND LIFECYCLE

Goal: Capture and track the agent versions. This step only tracks the output of the experimentation.

Version policy: Semantic versions for agent, tools, and prompts; additive changes favored. Pin model IDs (and commit or hash where available) and record model parameters.

Provenance and SBOMs: Generate software and prompt SBOMs that enumerate agent code commit, tool versions, prompt hashes, model identifiers, datasets or fixtures, and dependency trees. Sign and store alongside releases. Signing provides a trusted record of the artefacts that went into a specific agent code commit.

Release notes and impact levels: Classify changes as breaking, behavior-shifting, or non-functional; require corresponding validation depth and communication.

Deprecation policy: Announce deprecations with timelines; support dual-run where feasible to smooth migrations.

Comparison: Champion–challenger evaluation before promotion; shadow or canary when needed. Champion-challenger results have more weight than offline evaluations. They provide insights into how the new version will work on actual production data.

Retirement: Decommission plan, data retention, and evidence preservation; successor mapping.

MCP SERVERS LIFECYCLE: ENTERPRISE GUIDE & BEST PRACTICES

AI tools turn LLM “thoughts” into “actions”. Model Context Protocol (MCP) standardizes the interface AI agents use to interact with tools. Tools allow AI agents to integrate into enterprise and external systems and data to either retrieve information or perform an action. For example, create a GitHub Issue, open a ServiceNow ticket, read requirements from Confluence, and more.

MCP servers are the enterprise-grade integration surface for agentic systems. They expose tools, resources, and prompts in a standardized way that allows agents to act within well-defined, auditable boundaries. This page provides a high-level, principles-first guide for building secure, scalable AI tools with the MCP in enterprise environments.

WHAT IS MCP (AND WHY IT MATTERS)

The Model Context Protocol is an open standard that connects AI applications (clients) to the systems where context and actions live (servers). Instead of bespoke, per-app integrations, MCP defines a consistent way to discover capabilities and invoke them safely:

- Servers expose tools, resources, and prompts with typed schemas and descriptions.
- Clients connect over STDIO (local) or streaming HTTP (remote) to discover and call capabilities.
- The protocol enables secure, auditable action execution with clear contracts, which makes it the preferred standard for building AI tools at enterprise scale.

In this guide, “AI tools” are implemented as MCP servers. Treat MCP as the default interface for agent-accessible capabilities.

MCP CORE CONCEPTS

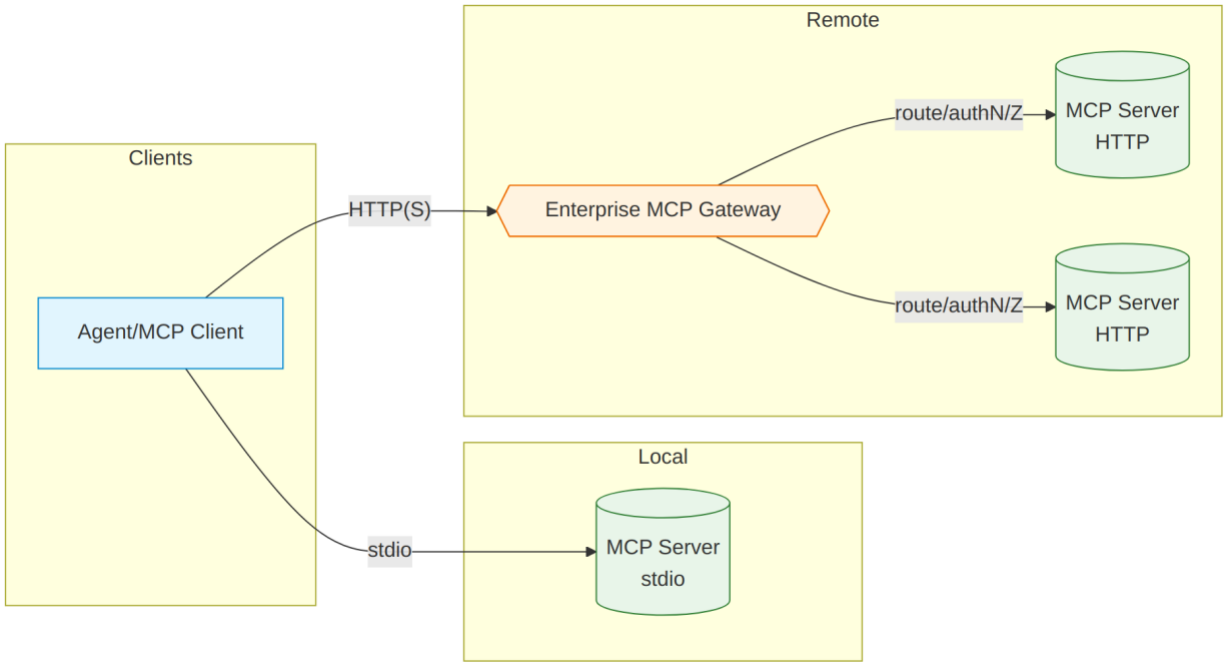
- **Tools**: Executable actions with explicit input/output schemas, constraints, and side-effect disclosures.
- **Resources**: Readable data sources (files, records, documents) the client can fetch for model context.
- **Prompts**: Predefined interaction templates that standardize tasks and reduce prompt drift.
- **Discovery**: Clients enumerate available tools, resources, prompts and obtain schemas at connect time.
- **Authorization**: Servers enforce scopes and roles per tool, with optional approvals for high-risk actions.
- **Transports**: STDIO for local processes; streaming HTTP for remote services and scale-out.

Choose the language that best matches your operational model and integration profile. Optimize for maintainability, observability, and SLOs—not theoretical speed.

WHERE MCP SERVERS FIT

MCP servers operationalize your enterprise integrations and actions. They live alongside your existing systems, enforce security and governance controls, and provide reliable, explainable capabilities to agents across use cases. They follow the same design principles as agents, including:

- **Evaluation-driven development**: Treat MCP behavior as a product; define success metrics, test with multiple models, and measure tool reliability.
- **Security by design**: Integrate identity, authorization, and audit into the server—not around it.
- **Controlled autonomy**: Offer precise, least-privilege tools with approval paths for high-impact actions.
- **Continuous governance**: Observe, version, and evolve servers without breaking dependent applications.



MCP Topology

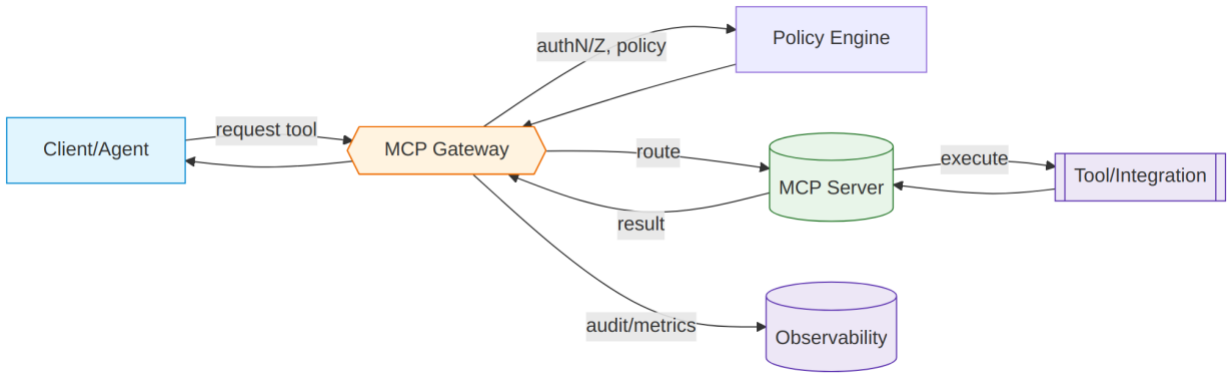
MCP GATEWAY PATTERN

Use an enterprise **MCP Gateway** when you need centralized security, control, and scale across many servers and tenants. The gateway becomes the single, policy-enforced ingress for agent access to organizational capabilities.

- **Centralized control**: One place for authentication, authorization, routing, rate limiting, quotas, and service discovery or catalog.
- **Security boundary**: Terminate TLS, enforce mTLS to backends, broker OAuth tokens or scopes, enforce permissions per tool and prompt security.
- **Policy and guardrails**: Apply policy-as-code (for example, OPA) for tool allow-or-deny, environment gating, approval requirements, and sensitive-data handling.
- **Multitenancy**: Per-tenant isolation for configurations, keys, logs, metrics, limits, and catalogs; distinct dev, stage, prod routes and policies.
- **Governance and audit**: Standardized logging, request correlation, audit trails for who, what, when, and why across all servers.
- **Reliability and scale:** HA, autoscaling, circuit breaking, retries with idempotency, backpressure, and traffic shaping (blue or green, canary, shadowing).
- **Compatibility layer**: Feature detection, server capability negotiation, schema normalization, version pinning, and kill switches for faulty tools.
- **Plugins**: Extensions with pre and post hooks to trigger observability, PII filtering, XSS or profanity filters, security, authentication, authorization, and more.

Minimum gateway responsibilities: identity and scope brokering, catalog or registry, routing and health checks, rate limits and quotas, policy enforcement, audit and metrics, and emergency kill switches. Example implementation: **mcp-context-forge**.

REQUEST FLOW WITH GATEWAY AND APPROVALS



MULTI-TENANCY AND ISOLATION

Single tenant by default: Simplifies auditing, secrets, and blast radius. Prefer per-tenant deployments for high-risk domains.

Explicit tenancy boundaries: Separate data paths, keys, and logs by tenant; apply per-tenant rate limits and quotas.

Workload isolation: Use containers with non-root users, read-only filesystems where possible, and minimal base images.

SECURITY FOUNDATIONS

Identity and access: Use OAuth per MCP specification with proper authorization flows and token refresh. Apply least privilege by default (read-only), with fine-grained, per-tool and per-parameter authorization. Gate high-risk or write operations behind explicit roles, policies, and approvals.

Input and output safety: Validate inputs with strict schemas, types, and ranges; reject invalid requests immediately. Sanitize all outputs to prevent injection into downstream systems and clearly label side effects. Apply guardrails and policy checks based on environment, resource class, or sensitivity, and require user approval where appropriate.

Secrets and transport: Store credentials only in managed secret managers—never inline. Enforce TLS for all transport, sign and verify containers and artifacts, and rely on trusted registries. Restrict network access with explicit ingress and egress controls, service-to-service authentication, and allowlists.

Sandboxing: External plugins should be deployed in sandboxed and properly permissioned environments (for example, lightweight virtualization, container security profiles) to enforce least-privilege access to infrastructure resources (network, filesystem, secrets, and more).

TOOLING DISCIPLINE

Design principles: Tools should have clear, actionable descriptions (purpose, constraints, side effects, usage guidance) and stable, versioned interfaces. Keep capabilities bounded—favor small, focused tools over "kitchen-sink" ones.

Management and governance: Support read-only deployments, dynamic enablement by tenant, role, environment, and production-safe filtering. For example, exclude destructive operations or gate sensitive tools with approvals.

State handling: Default to stateless execution for scalability and resilience. When state is required, externalize it to managed stores with explicit TTLs, clear PII handling, and no hidden in-memory dependencies.

Operations and safety: For long-running or side-effecting actions, use asynchronous patterns (handles, status tools, callbacks or webhooks), enforce idempotency with client keys and provide compensating or rollback actions where possible. Concurrency should be supported to handle parallel requests safely.

SCALABILITY AND RESILIENCY

Horizontal scale: Design for concurrent, short-lived requests; prefer idempotent operations and safe retries.

Rate limiting and backpressure: Apply per-tenant, per-tool limits; surface “try later” semantics; protect upstream systems.

Health, readiness, and circuit breakers: Publish health endpoints; trip on dependency failures; shed load gracefully.

Caching and batching: Cache read-heavy operations with TTL; batch compatible requests to reduce API usage and cost.

Compatibility and versioning: Version the server, tool schemas, and side-effect contracts; provide feature detection and safe fallbacks.

Connection management: Reuse HTTP clients, enable keep-alive, and set timeouts and retry policies with jitter.

GOVERNANCE, COMPLIANCE, RISK AND OBSERVABILITY

Observability and auditability: Maintain structured audit trails (who, what, when, why, with redaction) capturing arguments, decisions, and outcomes. Collect meaningful metrics (success rates, latency, errors, approvals, denials) to track reliability and policy impact.

Governance and policy: Centralize guardrails as code (for example, OPA) and enforce consistently across environments. Maintain curated catalogs of approved servers and tools with ownership, versions, capabilities, security review dates, risk levels, and compliance tags.

Compliance and risk management: Classify and control data handling (locality, retention, minimization, redaction and tokenization). Embed compliance into release processes with SBOMs, vulnerability scans, container signing, and dependency policies. Enforce separation of duties for dev, ops, and approvals, with break-glass procedures for production.

Lifecycle and change management: Pin versions with clear compatibility notes and migration guides. Enforce deprecation policies with timelines, dual-run windows, and automated detection of deprecated usage. Coordinate releases with dependency owners, align with change windows, and provide rollbacks.

Resiliency and operations: Define SLOs for critical tools (success rate, p95 latency, error budgets) and maintain runbooks for common failures, outages, and recovery steps. Ensure escalation paths, on-call routing, and tested rollback processes.

TESTING AND CERTIFICATION

Cross-model evaluation: Validate tool discovery and execution across hosted and local models; monitor behavioral drift.

Security tests: Negative testing, authorization bypass attempts, input fuzzing, and rate-limit validation.

Load and chaos: Establish SLOs; test degradations, dependency outages, and retry and idempotency behavior.

Prerelease gates: Block releases on failing quality, security, or performance thresholds.

Capability detection: Implement feature detection and fallbacks for evolving protocol features.

Contract tests: Maintain tool schema contract tests and golden responses for backward compatibility.

PACKAGING AND DISTRIBUTION

Project and build standards: Use a clear structure (src, tests, docs) with Makefiles for local dev, testing, container builds, and scans. Containerize with minimal, reproducible images that run as non-root, pin dependencies, and publish signed artifacts.

Containerization practices: Based on minimal images (Distroless, UBI), strip compilers and shells at runtime, and enforce read-only filesystems where possible. Provide health endpoints (/health, /ready) and wire liveness or readiness probes. Maintain dependency hygiene with pinned versions, lockfiles, and automated vulnerability patching.

Orchestration and networking: In Kubernetes, use HPAs for CPU and latency, PDBs for availability and resource requests and limits for predictability. Secure traffic through TLS termination at ingress and enforce mTLS and service identity for east-west communication (service mesh preferred). Apply strict network policies (egress allowlists, namespace isolation, per-workload secrets).

Secrets management: Source all credentials from external secret managers with regular rotation; never mount broad or shared credentials into pods.

Supply chain integrity: Generate and store SBOMs per build, fail builds on critical vulnerabilities, and enforce signing of containers and manifests. Verify signatures at deploy and enforce cluster-level signature policies. Ensure reproducibility with deterministic builds, pinned bases, cached layers, and recorded provenance.

Configuration and capability declaration: Drive configuration from environment variables and document required capabilities in the project README (for example, needs_filesystem_access, needs_api_key_user).

QUICK BUILD CHECKLIST

Purpose and scope: Single, clearly defined server role and bounded toolset.

SDK and spec: Official SDK where possible; document SDK and spec versions.

Security: OAuth scopes, least-privilege tools, approvals for high-risk actions, secrets in a manager.

Validation: Strong input schemas, output sanitization, error taxonomy, and retries with idempotency.

Operations: Health and readiness, rate limits, backpressure, circuit breakers, and basic SLOs.

Observability: Structured audit logs, metrics (success, latency, errors), tracing, and correlation IDs.

Compatibility: Versioned tool schemas, deprecation policy, feature detection, and contract tests.

Packaging: Minimal signed container, non-root runtime, and reproducible builds.

Docs: README with capabilities and tags, environment variables, runbooks, and changelog.

ENTERPRISE ADOPTION PATTERNS

Host responsibilities: Client apps must show full tool descriptions, request approvals for risky actions, and prevent tool shadowing.

Environment segmentation: Distinct dev, stage, prod; different toolsets and policies per environment.

Change management: Deprecation policies, version pinning, rollback strategies, and migration guidance.

PACKAGING AND INFRASTRUCTURE CHECKLIST

Container hardening: Minimal, non-root, read-only FS, health probes, and resource limits.

Network and identity: TLS and mTLS, egress policy, service accounts, and scoped OAuth tokens.

Observability: Logs, metrics, and traces; correlation IDs; dashboards and alerts.

Supply chain: SBOM, signatures, vulnerability scans, and provenance attestations.

Operations: HPA and PDB, rollouts (blue, green, canary), and backups of stateful dependencies.

CI/CD: Build, test (unit, integration, perf), scan, sign, and promote across dev, stage, prod with gates.

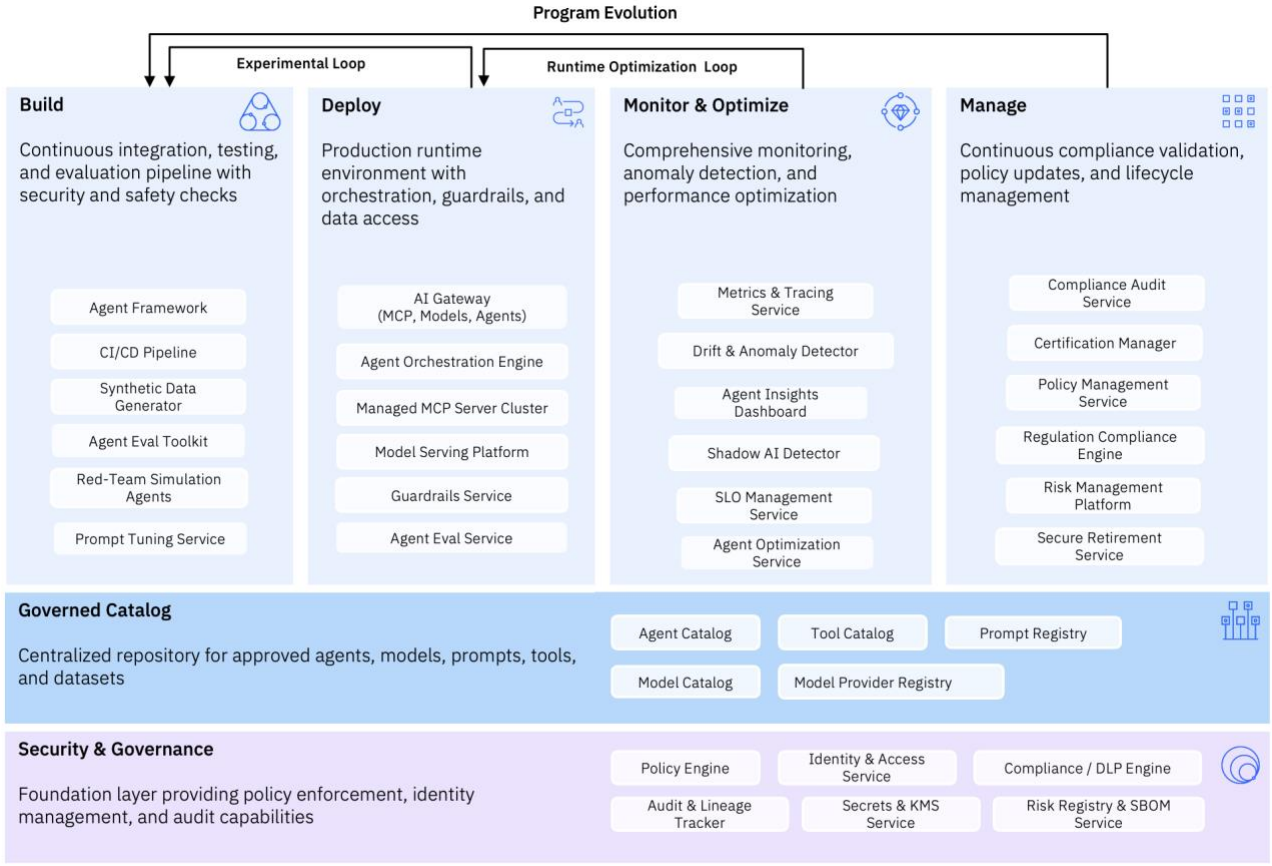
PRODUCTION READINESS CHECKLIST

- Identity and authorization implemented with least privilege and approvals for high-risk tools.
- Input validation, output sanitization, and policy guardrails in place.
- Audit logging, metrics, and alerts wired into enterprise observability.
- Rate limits, backpressure, health checks, and circuit breakers configured.
- Secrets in a managed store; containers minimal, signed, and non-root.
- Versioned APIs and tools with clear deprecation paths and compatibility tests.
- Documented SLOs, runbooks, incident response, and rollback procedures.

MCP servers succeed in the enterprise when they are treated as durable products: narrowly scoped, strongly governed, observable, and easy to evolve. Favor clarity, safety, and operability over breadth—then scale capabilities through catalogs and consistent patterns rather than bespoke implementations.

REFERENCE ARCHITECTURE & ENTERPRISE REQUIREMENTS FOR AN AGENTIC AI PLATFORM

In summary, the platform that is underneath enterprise agents determines operability, security, and scale. The following nonfunctional and functional requirements align with this guide's focus areas.



This framework outlines the reference architecture for implementing ADLC, structured into four core phases: **build, deploy, monitor and optimize, and manage**. The **build** phase focuses on continuous integration, testing, and evaluation with strong security and safety checks, leveraging tools like CI/CD pipelines, synthetic data generators, and red team simulation agents. **Deploy** transitions models into a production runtime environment with orchestration, guardrails, and data access, supported by components such as AI gateways, orchestration engines, and model serving platforms.
Once deployed, the **monitor and optimize** phase ensures system reliability through comprehensive monitoring, anomaly

detection, and performance tuning. Services like metrics tracing, drift detection, and shadow AI detection play a key role here. Finally, the **manage** phase addresses compliance validation, policy updates, and lifecycle management, incorporating tools for audits, certification, and risk management.
Supporting these phases are two foundational layers: a **governed catalog** for centralized management of approved agents, models, prompts, and datasets, and a **security and governance** layer that enforces policies, identity management, and audit capabilities. The framework also emphasizes two feedback loops—**experimental** and **runtime optimization**—to enable continuous improvement and adaptation.

AGENTIC AI PLATFORM REQUIREMENTS

NON-FUNCTIONAL REQUIREMENTS (NFR)

Category	Requirements
Architecture, Integration and Scaling	Agent and tool catalogs; MCP Gateway for capability routing and policy (REST↔MCP conversion, virtual MCP servers, protocol bridging); Model Gateway for unified API access (OpenAI-compatible) and portfolio management; horizontal and federated scaling to support thousands of tools and agents across deployments and platforms.
Build-time security	Builder identity and access (RBAC), workspaces, data security, access logging, and enterprise SaaS/SOC controls; observability of the builder environment (operational logs, availability and performance monitoring, metering); governance and compliance of the builder environment (standards, AI governance, supply-chain risk management for agents and tools).
Run-time security	Agent identity, authentication and OAuth, and delegation with granular authorization; data encryption (BYOK) and strong tenant isolation; AI application protection (prompt security, artifact signing and attestation, hardened deployments, memory protection); security monitoring (audit logs, threat detection, incident-response integration); enterprise security integration (identity systems, SOC, red-teaming) with a clear shared-responsibility model.
Observability and monitoring	Full telemetry coverage (metrics, events, logs, traces), distributed tracing and conversational logging; quality monitoring through evaluation frameworks (accuracy, bias, safety); integration with enterprise observability stacks and cost and token accounting.
Governance and compliance	Compliance with enterprise and industry standards (ISO or SOC, GDPR, HIPAA as applicable); safety guardrails (hallucination, bias, relevance), drift detection; governed catalogs for models, agents, tools and supply-chain risk management; enterprise GRC integration and evidence capture.
Resilience, cost and ethics	Self-healing, failover, and graceful degradation; cost controls and optimization levers; bias detection and fairness metrics; optional sustainability metrics such as carbon-footprint tracking.
Deployment and future-ready	Support from air-gapped to hyperscaler deployments; bring-your-own authentication and secrets; portability across environments; versioning and compatibility strategies for models, tools, and prompts.

FUNCTIONAL REQUIREMENTS (CAPABILITIES)

Category	Requirements
Memory and state management	Short- and long-term memory with configurable retention; session persistence and context maintenance across interactions; integration with vector and or graph databases (CRUD), caching, and PII handling policies.
Planning and execution	Task decomposition and execution planning; optional human-in-the-loop plan review and approvals for high-impact actions; safe tool orchestration and secure code execution; async and parallel processing patterns with idempotency and retries.
Interoperability and developer experience	Support MCP for tools, resources, prompts, agent-to-agent (A2A) patterns, OpenAI-compatible model APIs, and OpenAPI-described integrations; plugin architecture and curated marketplace or catalog; open-source SDKs with a fast start (“5 minutes to fun”); bring-your-own model or agent.

Category	Requirements
Knowledge management	Retrieval-augmented generation (RAG) and knowledge integration; artifact generation and storage (docs, reports, visualizations) with lineage; scalable processing patterns such as map-reduce for large inputs.
Human-agent collaboration	Human-in-the-loop approvals and escalations; transparent explainability (“debug mode”) and trace inspection; trust-building via understandable decisions and selective reproducibility.
Performance and evaluation	Behavior logging, scoring, and self-evaluation; human review mechanisms and red team workflows; champion-challenger comparisons and promotion gates integrated with CI/CD.
Future patterns and autonomy	Inter-agent collaboration and swarms where justified; adaptive learning and self-evaluation loops; proactive intelligence and event-driven actions with clear authority boundaries and kill-switches.

APPENDIX: VOICE OF THE CUSTOMER: CLIENT EXAMPLES

HEALTHCARE CLIENT EXAMPLE

A large U.S. healthcare payer set out to improve its member support experience by introducing agentic AI while maintaining HIPAA compliance. The project illustrated how the agent development lifecycle guides secure and reliable deployment of enterprise agents.

Use case design and evaluation. The client aimed to automate common support requests and reduce call center load. Early work was slowed by limited access to historical call data and a legacy chatbot that was difficult to untangle. To move forward, the team defined clear KPIs such as resolution rate, containment rate, latency, and safety thresholds. They also developed methods to create or synthesize ground truth data in a compliant way to ensure that agent behavior could be measured from the start.

Development and integration. Existing test processes were manual and subjective. The team adopted an evaluation framework that benchmarked agent behavior against business metrics. Prompts, memory policies, and integration logic were versioned and managed like code. Tools were implemented as governed services with clear boundaries and least-privilege access, which made them easier to monitor and secure.

Testing and certification. Without a single “correct” answer for each member interaction, testing focused on structured evaluations such as task success rate, groundedness, and compliance checks. Security testing and red teaming were also introduced. Agents were released only after passing through a governed catalog process that enforced evaluation thresholds, HIPAA compliance, and rollback planning.

Deployment and scaling. Standard SaaS deployments did not meet HIPAA requirements, so the client deployed a fully managed, compliant agent stack. Infrastructure was optimized for high-throughput, low-latency inference and designed with resilience in mind, including circuit breakers, canary rollouts, and failover mechanisms to prevent drift or outages from impacting member service.

Monitoring and optimization. Monitoring had previously been fragmented across Grafana dashboards, cloud logs, and business reports. The client replaced these systems with unified observability that tracked both technical metrics (latency, error rate, throughput) and business outcomes (containment, resolution, satisfaction). Continuous evaluation and monitoring created a feedback loop to improve prompts, tools, and models over time.

Review and compliance. Compliance became a continuous process rather than a one-time check. Agents and tools were cataloged with full version history, ownership, and audit trails. Policies enforced strict authority boundaries and data handling rules. Agents could be retired or decommissioned in a controlled manner, with all evidence and lineage preserved.

Outcome. By following the ADLC with embedded DevSecOps practices, the healthcare payer achieved measurable improvements in member support while maintaining HIPAA compliance. Resolution and containment rates increased, call escalations decreased, and the organization established a repeatable framework for deploying future healthcare agents.

APPENDIX: ENTERPRISE USE CASES

We describe our experiences with customer engagements in the healthcare, telecommunication and financial industry.

LARGE HEALTHCARE COMPANY

This healthcare organization implemented agentic chatbot solutions for member services and voice assistance, operating within a HIPAA-compliant, cloud-based SaaS environment. The deployment required sub-10 second response times for all agentic interactions while maintaining strict healthcare data privacy standards. The organization faced significant challenges in the planning phase due to restricted access to historical call center data, which prevented the upfront construction of ground truth datasets necessary for agent training and evaluation. Additionally, their existing chatbot business logic proved complex and difficult to reverse-engineer, creating substantial technical debt that impacted development velocity. Key requirements identified include methodology and tools for synthetic ground truth generation for agentic trajectories. Other key requirements include automated agent build and test generation using standard operating procedures as input, and AI-driven discovery capabilities for reverse-engineering business logic from legacy systems.

The testing and deployment phases revealed critical infrastructure gaps requiring comprehensive solutions. Their manual testing processes dominated evaluation approaches, relying on human testers rather than automated, metrics-based assessment frameworks, which highlighted the need for common agent eval frameworks and benchmarking harnesses for best-in-class agentic benchmarks. The deployment phase emphasized requirements for standard SaaS-based deployment of HIPAA-compliant agentic stacks with high-throughput, low-latency LLM inference infrastructure and hardware acceleration as a service. Their monitoring infrastructure consisted of custom Grafana dashboards tracking latency and error rates but lacked unified metrics integration. This revealed requirements for built-in agent observability tooling, unified metrics dashboards, and comprehensive AgentOps tooling that captures and analyzes agentic trajectories against both business and technical metrics, including continuous improvement capabilities.

The healthcare use case demonstrated the critical need for fully managed, integrated agentic stacks that can operate within strict compliance frameworks while delivering performance characteristics demanded by customer-facing applications. Their specific requirements centered on containment rate optimization, error rate minimization, response time optimization, and the ability to automatically generate recommendations with intent identification mapped to domain ontologies that require sophisticated business logic integration.

LARGE TELECOMMUNICATION COMPANY

This telecommunications provider undertook a significant initiative to enhance their internal Knowledge as a Service (KaaS) platform, by targeting 95% accuracy for production-ready applications and 80% for proof-of-concept deployments. Their current development cycle spans 9–12 months per application to achieve production-ready accuracy standards, with a substantial backlog of gen AI applications that require rapid iteration and deployment. The organization's architecture leverages Langflow, NVIDIA infrastructure, and third-party evaluation tools, which reflects the complex vendor ecosystem integration requirements. Key planning phase requirements include integrated stacks for agent building combined with AgentOps to accelerate adoption, enterprise-level access control support in development environments, and packaged AgentOps observability tools integrated with agent builders.

The development and testing phases exposed critical gaps around experiment tracking frameworks, agentic evaluation capabilities, and version management systems. The organization struggled with tracking tens of agent variants and lacked integrated evaluation frameworks, which ultimately required external partnerships for evaluation and experimentation capabilities. Their requirements emphasized common agent eval frameworks and services embedded directly into development stacks, benchmarking harnesses for best-in-class agentic benchmarks compatible with any agentic framework, and comprehensive experiment tracking systems. Testing phase requirements included enterprise-grade role-based access

control for agent execution through APIs, robust data lifecycle management for knowledge bases, and comprehensive version control systems for agent code.

The telecommunications case highlighted enterprise-scale operational requirements, including end-to-end lifecycle governance with integrated RBAC capabilities that span development to production environments. Their technical requirements centered on achieving and maintaining high accuracy thresholds while managing complex multitool orchestration, preventing agent sprawl through centralized governance, and eliminating shadow AI usage through approved catalogs of agents, models, prompts, and tools. The organization's scale demands reflected the operational complexity of telecommunications infrastructure where comprehensive governance frameworks and centralized registries become critical for managing enterprise-wide agent deployments.

LARGE FINANCIAL COMPANY

This major bank represents an industry-first collaboration focused on defining and building enterprise agent development lifecycle (ADLC) and AgentOps frameworks specifically tailored for highly regulated financial services environments. The financial institution's requirements fundamentally extend beyond traditional DevSecOps practices to recognize that while traditional approaches secure source code, AI agents require comprehensive security for data access, embeddings, prompts, and RAG pipelines. Their security infrastructure must implement specialized vulnerability scanning that tests for prompt injection, jailbreaks, adversarial examples, and model poisoning rather than relying solely on conventional code scanning. The banking environment demands evolved security models that account for agent autonomy and tool use, where agents can call external APIs or trigger workflows beyond their core code. This scenario requires security coverage across the entire orchestration layer including tools, connectors, and multi-agent collaboration scenarios.

The financial services context introduces comprehensive evaluation and governance requirements that address the stochastic nature of AI agents compared to traditional deterministic applications. Their framework requires continuous evaluation that is embedded within CI/CD pipelines to monitor not only uptime, but also accuracy, behavioral drift, context relevancy, and cost trends in real-time. Critical requirements include observability systems that capture reasoning traces and decisions for regulatory audit purposes, expanding traditional monitoring to encompass LLM reasoning chains, vector store queries, tool calls, and orchestration workflows. The institution requires centralized governance registries that maintain catalogs of approved agents, models, prompts, and tools to prevent agent sprawl and shadow AI usage. This governance process is similar to how traditional DevSecOps tracks services and dependencies but is adapted for the complexities of agentic systems.

This collaboration is anticipated to establish new industry standards for agent governance in regulated financial environments, where the monetary consequences of errors demand extreme precision and comprehensive audit capabilities. The banking use case emphasizes the need for explainable AI systems that can demonstrate regulatory compliance across multiple frameworks, including SOX compliance, PCI DSS standards, and regional banking regulations. Their requirements are expected to drive innovations in model risk management, algorithmic bias detection and mitigation, continuous compliance monitoring, and risk-aware agent orchestration that will likely influence agent development practices across other highly regulated industries. These changes will establish patterns for enterprise-scale agent governance that balances innovation with strict regulatory adherence.

Produced in the
United States of America
October, 2025

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

THE INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT.

IBM products are warranted according to the terms and conditions of the agreements under which they are provided.