Agentic AI frameworks, platforms, protocols, and tools on AWS

AWS Prescriptive Guidance



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Prescriptive Guidance: Agentic AI frameworks, platforms, protocols, and tools on AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction **1**
Intended audience 1
Objectives 2
About this content series 2
Frameworks **3**
Strands Agents 4
Key features of Strands Agents 4
When to use Strands Agents 5
Implementation approach for Strands Agents 5
Real-world example of Strands Agents 6
LangChain and LangGraph 6
Key features of LangChain and LangGraph 6
When to use LangChain and LangGraph 7
Implementation approach for LangChain and LangGraph 8
Real-world example of LangChain and LangGraph 8
CrewAI 8
Key features of CrewAI 9
When to use CrewAI 9
Implementation approach for CrewAI 10
Real-world example of CrewAI 10
AutoGen 11
Key features of AutoGen 11
When to use AutoGen 12
Implementation approach for AutoGen 12
Real-world example of AutoGen 12
LLamaIndex 13
Key features of LlamaIndex 13
When to use LlamaIndex 14
Implementation approach for LlamaIndex 14
Real-world example of LlamaIndex 15
Comparing agentic AI frameworks 15
Considerations in choosing an agentic AI framework 16
Platforms **18**
Why platforms matter 18

Types of agentic AI platforms ... 19
Platform selection considerations ... 19
Amazon Bedrock Agents ... 19
Key features of Amazon Bedrock Agents ... 20
When to use Amazon Bedrock Agents ... 20
Implementation approach for Amazon Bedrock Agents ... 21
Real-world example of Amazon Bedrock Agents ... 21
Amazon Bedrock AgentCore ... 22
Key features of AgentCore ... 23
When to use AgentCore ... 24
Implementation approach for AgentCore ... 24
Real-world example of AgentCore ... 25
Protocols ... 26
Why protocol selection matters ... 26
Advantages of open protocols ... 27
Agent-to-agent protocols ... 27
Deciding among protocol options ... 28
Selecting agentic protocols ... 29
Agentic protocol selection considerations ... 29
Implementation strategy for agentic protocols ... 29
Getting started with MCP ... 30
Getting started with A2A ... 30
Tools ... 33
Tool categories ... 33
Protocol-based tools ... 33
Framework-native tools ... 34
Meta-tools ... 34
Protocol-based tools ... 34
Security features of MCP tools ... 35
Getting started with MCP tools ... 35
Explore AgentCore Gateway ... 36
Framework-native tools ... 36
Meta-tools ... 37
Workflow meta-tools ... 37
Agent graph meta-tools ... 37
Memory meta-tools ... 38

Tool integration strategy 38
Security best practices for tool integration 39
Authentication and authorization 39
Data protection 39
Monitoring and auditing 39
Conclusion 41
Resources 42
AWS Blogs 42
AWS Prescriptive Guidance 42
AWS resources 43
Other resources 43
Document history 44

Agentic AI frameworks, platforms, protocols, and tools on AWS

Aaron Sempf, Joshua Samuel, and Ansley Verzosa, Amazon Web Services

Agentic AI is a paradigm at the intersection of AI, distributed systems, and software engineering. It's a class of intelligent systems composed of autonomous, asynchronous software agents that use AI models and integrate with tools and resources. The agents exhibit agency: they perceive context, reason over goals, make decisions, and take purposeful actions on behalf of users or systems. These agents operate independently, often collaboratively, within distributed environments and are designed to pursue delegated objectives with embedded intelligence, memory, and intent.

On AWS, organizations can use agentic AI to automate complex workflows, enhance decision-making, and create more responsive systems. This guide describes the key components that you need to build agentic AI solutions:

- Frameworks profiles current agentic AI frameworks, including reviews of their benefits and use cases. Learn how these frameworks reduce undifferentiated heavy lifting across patterns, protocols, and tools. Understand key selection criteria to choose the right framework for your requirements.
- Platforms provides an overview of agentic AI platforms (managed agent, open source orchestration, and hybrid) and considerations for selection or design.
- Protocols explores standardized communication protocols for agent interactions. Agent-to-agent protocols are emerging, such as the open source Model Context Protocol (MCP) and Agent2Agent (A2A), along with other proprietary implementations. Discover how common protocols enable different agents to interact.
- Tools provides information about protocol-based tools, framework-native tools, and meta-tools. Organizations can build a toolkit that integrates with the key systems in their workflows, enabling both end-user and server-based agentic workflows.

Intended audience

This guide is for architects, developers, and technology leaders who want to use AI-driven software agents in cloud-native applications.

Objectives

This guide helps you do the following:

- Compare agentic AI frameworks to select the one that meets your use case requirements.
- Learn about agentic AI services that provide capabilities to turn individual agents into coordinated, adaptive systems.
- Understand the advantages of open protocols for building sustainable agentic AI architectures.
- Create a tool integration strategy that meets your requirements when you build agent systems.

About this content series

This guide is part of a series about agentic AI on AWS. For more information and to view the other guides in this series, see Agentic AI on the AWS Prescriptive Guidance website.

Frameworks

Foundations of agentic AI on AWS examines the core patterns and workflows that enable autonomous, goal-directed behavior. The choice of framework is central to implementing these patterns. A *framework* is prewritten software code that provides a structured environment, common functionality, and the tools and orchestration capabilities that you need to build and manage production-ready autonomous AI agents.

Effective agentic AI frameworks provide several essential capabilities that transform raw large language model (LLM) interactions into coordinated, intelligent systems capable of reasoning, collaboration, and action:

- **Agent orchestration** coordinates the flow of information and decision-making across single or multiple agents to achieve complex goals without human intervention.
- **Tool integration** enables agents to interact with external systems, APIs, and data sources to extend their capabilities beyond language processing. For more information, see Tools Overview in the Strands Agents documentation.
- **Memory management** provides persistent or session-based state to maintain context across interactions, essential for long-running or adaptive tasks. More advanced frameworks incorporate long-term memory to store summaries and user preferences, enabling personalized and contextually aware agentic experiences. For more information, see How to think about agent frameworks on the LangChain Blog.
- **Workflow definition** supports structured patterns like chains, routing, parallelization, and reflection loops that enable sophisticated autonomous reasoning.
- **Deployment and monitoring** facilitate the transition from development to production with observability for autonomous systems. For more information, see the Amazon Bedrock AgentCore general availability announcement.

These capabilities are implemented with varying approaches and emphases across the framework landscape, each offering distinct advantages for different autonomous agent use cases and organizational contexts.

This section profiles and compares the leading frameworks for building agentic AI solutions, with a focus on their strengths, limitations, and ideal use cases for autonomous operation:

- Strands Agents

- LangChain and LangGraph
- CrewAI
- Amazon Bedrock Agents
- Amazon Bedrock AgentCore
- AutoGen
- LlamaIndex
- Comparing agentic AI frameworks

> **Note**
>
> This section covers the frameworks that specifically support agency of the AI and doesn't cover frontend interfaces or generative AI without agency.

Strands Agents

AWS initially released Strands Agents as an open-source SDK, as described in the AWS Open Source Blog. You can use Strands Agents to build autonomous AI agents with a model-driven approach. It provides a flexible, extensible framework that works with AWS services and remains open to integration with third-party components. Strands Agents is suited for building autonomous solutions.

Key features of Strands Agents

Strands Agents includes the following key features:

- **Model-first design** – Built around the concept that the foundation model is the core of agent intelligence, enabling sophisticated autonomous reasoning. For more information, see Agent Loop in the Strands Agents documentation.
- **Multi-agent collaboration patterns** – Built-in coordination models such as Swarm, Graph, and Workflow patterns that enable scalable collaboration and governance across distributed agent networks. For more information, see Multi-agent Patterns in the Strands Agents documentation.
- **MCP integration** – Native support for the Model Context Protocol (MCP), enabling standardized context provision to LLMs for consistent autonomous operation.

- **AWS service integration** – Connection to Amazon Bedrock, AWS Lambda, AWS Step Functions, and other AWS services for autonomous workflows. For more information, see AWS Weekly Roundup (AWS Blog).
- **Foundation model selection** – Supports various foundation models including Anthropic Claude, Amazon Nova (Premier, Pro, Lite, and Micro) on Amazon Bedrock, and others so that you can choose a model for different autonomous reasoning capabilities. For more information, see Amazon Bedrock in the Strands Agents documentation.
- **LLM API integration** – Flexible integration with different LLM service interfaces including Amazon Bedrock, OpenAI, and others for production deployment. For more information, see Amazon Bedrock Basic Usage in the Strands Agents documentation.
- **Multimodal capabilities** – Support for multiple modalities including text, speech, and image processing for autonomous agent interactions. For more information, see Amazon Bedrock Multimodal Support in the Strands Agents documentation.
- **Extensible tool set** – Set of tools for AWS service interaction, with extensibility for custom tools that expand autonomous capabilities. For more information, see Tools Overview in the Strands Agents documentation.

When to use Strands Agents

Strands Agents is particularly well-suited for autonomous agent scenarios including:

- Organizations that build on AWS infrastructure who want native integration with AWS services for autonomous workflows
- Teams that require security, scalability, and compliance features for production autonomous systems
- Projects that need flexibility in model selection across different providers for specialized autonomous tasks
- Use cases that require integration with existing AWS workflows and resources for autonomous processes

Implementation approach for Strands Agents

The Strands Agents implementation approach is outlined in the Quickstart Guide for Python and Quickstart Guide for TypeScript. With the framework, you can:

- Select foundation models like Amazon Nova (Premier, Pro, Lite, or Micro) on Amazon Bedrock based on specific business requirements.
- Define custom tools that connect to enterprise systems and data sources.
- Process multiple modalities including text, images, and speech.
- Deploy agents that can autonomously respond to business queries and perform tasks.

This implementation approach helps teams develop and deploy autonomous agents without building or training their own AI models.

Real-world example of Strands Agents

AWS Transform for .NET uses Strands Agents to power its application modernization capabilities, as described in AWS Transform for .NET, the first agentic AI service for modernizing .NET applications at scale (AWS Blog). This production service uses multiple specialized autonomous agents. The agents work together to analyze legacy .NET applications, plan modernization strategies, and run code transformations to cloud-native architectures. AWS Transform for .NET is an example of Strands Agents used in a production autonomous system.

LangChain and LangGraph

LangChain is an established framework for building agentic AI applications. LangGraph extends its capabilities to support complex, stateful agent workflows, as described in the LangChain Blog. Together, they provide orchestration capabilities for building autonomous AI agents that operate independently.

Key features of LangChain and LangGraph

LangChain and LangGraph include the following key features:

- **Component library** – Library of prebuilt components for various autonomous agent capabilities, enabling rapid development of specialized agents. For more information, see the LangChain documentation.
- **Foundation model selection** – Support for diverse foundation models including Anthropic Claude, Amazon Nova models (Premier, Pro, Lite, and Micro) on Amazon Bedrock, and others for different reasoning capabilities. For more information, see Inputs and outputs in the LangChain documentation.

- **LLM API integration** – Standardized interfaces for multiple large language model (LLM) service providers including Amazon Bedrock, OpenAI, and others for flexible deployment. For more information, see LLMs in the LangChain documentation.
- **Multimodal processing** – Built-in support for text, image, and audio processing to enable rich multimodal autonomous agent interactions. For more information, see Multimodality in the LangChain documentation.
- **Graph-based workflows** – LangGraph enables defining complex autonomous agent behaviors as state machines, supporting sophisticated decision logic. For more information, see the LangGraph Platform GA announcement.
- **Memory abstractions** – Multiple options for short-term and long-term memory management, which helps autonomous agents maintain context over time. For more information, see How to add memory to chatbots in the LangChain documentation.
- **Tool integration** – Integrations with a variety of tools, services, and APIs that extend autonomous agent capabilities. For more information, see Tools in the LangChain documentation.
- **LangGraph Platform** – Managed deployment and monitoring solution for production environments, supporting long-running autonomous agents. For more information, see the LangGraph Platform GA announcement.

When to use LangChain and LangGraph

LangChain and LangGraph are particularly well-suited for autonomous agent scenarios including:

- Complex multi-step reasoning workflows that require sophisticated orchestration for autonomous decision-making
- Projects that need access to a wide range of prebuilt components and integrations for diverse autonomous capabilities
- Teams with existing Python-based machine learning (ML) infrastructure and expertise that want to build autonomous systems
- Use cases that require complex state management across long-running autonomous agent sessions

Implementation approach for LangChain and LangGraph

LangChain and LangGraph provide a structured implementation approach for business stakeholders, as detailed in the LangGraph documentation. The framework enables organizations to:

- Define sophisticated workflow graphs that represent business processes.
- Create multi-step reasoning patterns with decision points and conditional logic.
- Integrate multimodal processing capabilities for handling diverse data types.
- Implement quality control through built-in review and validation mechanisms.

This graph-based approach allows business teams to model complex decision processes as autonomous workflows. Teams have clear visibility into each step of the reasoning process and the ability to audit decision paths.

Real-world example of LangChain and LangGraph

A telecommunications company implemented autonomous agents using LangChain and LangGraph to enhance its data engineering and operations workflows, as detailed in a LangChain Enterprise case study. The company built internal AI assistants that autonomously monitor performance metrics, retrieve information from documentation systems, and present actionable insights through natural language interactions.

The implementation used LangChain modular document loaders, vector integration, and support for multiple LLMs (OpenAI, Llama 3, and Gemini) to rapidly prototype and benchmark these pipelines. The team then used LangGraph to structure the multi-agent orchestration by deploying modular subagents. These agents perform collection, processing, summarization, and reasoning tasks. The team integrated these agents into their cloud systems through APIs.

CrewAI

CrewAI is an open-source framework focused specifically on autonomous multi-agent orchestration, available on GitHub. It provides a structured approach to creating teams of specialized autonomous agents that collaborate to solve complex tasks without human intervention. CrewAI emphasizes role-based coordination and task delegation.

Key features of CrewAI

CrewAI provides the following key features:

- **Role-based agent design** – You define autonomous agents with specific roles, goals, and backstories to enable specialized expertise. For more information, see Crafting Effective Agents in the CrewAI documentation.
- **Task delegation** – Built-in mechanisms for autonomously assigning tasks to agents based on their capabilities. For more information, see Tasks in the CrewAI documentation.
- **Agent collaboration** – Framework for autonomous inter-agent communication and knowledge sharing without human mediation. For more information, see Collaboration in the CrewAI documentation.
- **Process management** – Structured workflows for sequential and parallel autonomous task execution. For more information, see Processes in the CrewAI documentation.
- **Foundation model selection** – Support for various foundation models including Anthropic Claude, Amazon Nova models (Premier, Pro, Lite, and Micro) on Amazon Bedrock, and others to optimize for different autonomous reasoning tasks. For more information about Amazon Bedrock, see What is Amazon Bedrock? For more information about model selection in CrewAI, see LLMs in the CrewAI documentation.
- **LLM API integration** – Flexible integration with multiple large language model (LLM) service interfaces including Amazon Bedrock, OpenAI, and local model deployments. For more information, see Provider Configuration Examples in the CrewAI documentation.
- **Multimodal support** – Emerging capabilities for handling text, image, and other modalities for comprehensive autonomous agent interactions. For more information, see Using Multimodal Agents in the CrewAI documentation.

When to use CrewAI

CrewAI is particularly well-suited for autonomous agent scenarios including:

- Complex problems that benefit from specialized, role-based expertise working autonomously
- Projects that require explicit collaboration between multiple autonomous agents
- Use cases where team-based problem decomposition improves autonomous problem-solving
- Scenarios that require clear separation of concerns between different autonomous agent roles

Implementation approach for CrewAI

CrewAI provides a role-based approach for implementing teams of AI agents for business stakeholders, as described in Getting Started in the CrewAI documentation. The framework enables organizations to:

- Define specialized autonomous agents with specific roles, goals, and expertise areas.
- Assign tasks to agents based on their specialized capabilities.
- Establish clear dependencies between tasks to create structured workflows.
- Orchestrate collaboration between multiple agents to solve complex problems.

This role-based approach mirrors human team structures, making it intuitive for business leaders to understand and implement. Organizations can create autonomous teams with specialized expertise that collaborate to achieve business objectives, similar to how human teams operate. Unlike human teams, autonomous teams can work continuously without human intervention.

Real-world example of CrewAI

AWS has implemented autonomous multi-agent systems using CrewAI integrated with Amazon Bedrock, as described in the published CrewAI case study. AWS and CrewAI developed a vendor-neutral framework with security in mind. The CrewAI open-source "flows-and-crews" architecture integrates with Amazon Bedrock foundation models, memory systems, and compliance guardrails.

Key elements of the implementation include:

- **Blueprints and open sourcing** – AWS and CrewAI released reference designs that map CrewAI agents to Amazon Bedrock models and observability tools. They also released exemplar systems such as a multi-agent AWS security audit crew, code modernization flows, and consumer packaged goods (CPG) back-office automation.
- **Observability stack integration** – The solution embeds monitoring with Amazon CloudWatch, AgentOps, and LangFuse, enabling traceability and debugging from proof-of-concept to production. For more information about Amazon CloudWatch, see What is Amazon CloudWatch?
- **Demonstrated return on investment (ROI)** – Early pilots demonstrated improvements, including 70 percent faster execution for a large code modernization project and about 90 percent reduction in processing time for a CPG back-office flow.

AutoGen

AutoGen is an open-source framework that Microsoft initially released. AutoGen focuses on enabling conversational and collaborative autonomous AI agents. It provides a flexible architecture for building multi-agent systems with an emphasis on asynchronous, event-driven interactions between agents for complex autonomous workflows.

Key features of AutoGen

AutoGen provides the following key features:

- **Conversational agents** – Built around natural language conversations between autonomous agents, enabling sophisticated reasoning through dialogue. For more information, see Multi-agent Conversation Framework in the AutoGen documentation.
- **Asynchronous architecture** – Event-driven design for non-blocking autonomous agent interactions, supporting complex parallel workflows. For more information, see Solving Multiple Tasks in a Sequence of Async Chats in the AutoGen documentation.
- **Human-in-the-loop** – Support for optional human participation in otherwise autonomous agent workflows when needed. For more information, see Allowing Human Feedback in Agents in the AutoGen documentation.
- **Code generation and running** – Specialized capabilities for code-focused autonomous agents that can write and run code. For more information, see Code Execution in the AutoGen documentation.
- **Customizable behaviors** – Flexible autonomous agent configuration and conversation control for diverse use cases. For more information, see agentchat.conversable_agent in the AutoGen documentation.
- **Foundation model selection** – Support for various foundation models including Anthropic Claude, Amazon Nova models (Premier, Pro, Lite, and Micro) on Amazon Bedrock, and others for different autonomous reasoning capabilities. For more information, see LLM Configuration in the AutoGen documentation.
- **Large language model (LLM) API integration** – Standardized configuration for multiple LLM service interfaces including Amazon Bedrock, OpenAI, and Azure OpenAI. For more information, see oai.openai_utils in the AutoGen API Reference.
- **Multimodal processing** – Support for text and image processing to enable rich multimodal autonomous agent interactions. For more information, see Engaging with Multimodal Models: GPT-4V in AutoGen in the AutoGen documentation.

When to use AutoGen

AutoGen is particularly well-suited for autonomous agent scenarios including:

- Applications that require natural conversational flows between autonomous agents for complex reasoning
- Projects that need both fully autonomous operation and optional human oversight capabilities
- Use cases that involve autonomous code generation, running, and debugging without human intervention
- Scenarios that require flexible, asynchronous autonomous agent communication patterns

Implementation approach for AutoGen

AutoGen provides a conversational implementation approach for business stakeholders, as detailed in Getting Started in the AutoGen documentation. The framework enables organizations to:

- Create autonomous agents that communicate through natural language conversations.
- Implement asynchronous, event-driven interactions between multiple agents.
- Combine fully autonomous operation with optional human oversight when needed.
- Develop specialized agents for different business functions that collaborate through dialogue.

This conversational approach makes the autonomous system's reasoning transparent and accessible to business users. Decision-makers can observe the dialogue between agents to understand how conclusions are reached and optionally participate in the conversation when human judgment is required.

Real-world example of AutoGen

Magentic-One is an open-source, generalist multi-agent system designed to autonomously solve complex, multi-step tasks across diverse environments, as described in the Microsoft AI Frontiers blog. At its core is the Orchestrator agent, which decomposes high-level goals and tracks progress by using structured ledgers. This agent delegates subtasks to specialized agents (such as WebSurfer, FileSurfer, Coder, and ComputerTerminal) and adapts dynamically by re-planning when necessary.

The system is built on the AutoGen framework and is model-agnostic, defaulting to GPT-4o. It delivers competitive performance across benchmarks such as GAIA, AssistantBench, and WebArena

without task-specific tuning. Additionally, it supports modular extensibility and evaluation through AutoGenBench.

LLamaIndex

LlamaIndex is a data framework designed specifically for connecting large language models (LLMs) with external data sources to enable sophisticated Retrieval Augmented Generation (RAG) and agentic AI applications. The framework provides abstractions and accelerated development workflows for agentic systems, custom orchestration patterns, and system integrations that reduce time-to-production for knowledge-driven AI solutions.

Key features of LlamaIndex

LlamaIndex provides a comprehensive set of capabilities that makes it particularly well-suited for enterprise agentic AI applications:

- **Data-centric architecture** – Excels at ingesting, indexing, and retrieving information from over 100 data formats including PDFs, Microsoft Word documents, spreadsheets, and more. The framework transforms enterprise data into queryable knowledge bases optimized for AI agents. For more information, see the LlamaIndex documentation.
- **Production-ready deployment** – LlamaIndex offers both open-source frameworks and managed services through LlamaCloud, providing enterprise-grade features including security controls, scalability, observability integrations, and deployment flexibility. For more information, see the LlamaIndex framework documentation.
- **Advanced document processing** – LlamaCloud provides document parsing, extraction, indexing, and retrieval capabilities that handle complex layouts, nested tables, multi-modal content, and even handwritten notes. This sophisticated parsing enables agents to work effectively with real-world enterprise documents that contain charts, diagrams, and complex formatting. For more information, see the LlamaCloud documentation.
- **Workflows orchestration** – LlamaAgents provides an event-driven, async-first orchestration engine for building multi-step agentic systems. Workflows support complex patterns including loops, parallel execution, conditional branching, and stateful resumption, making them well-suited for complex agent interactions. For more information, see the LlamaIndex workflows documentation.
- **Agentic retrieval capabilities** – Advanced retrieval modes including hybrid search, semantic search, and auto-routing that select an effective retrieval strategy for each query. The framework

supports composite retrieval across multiple knowledge bases with reranking for enhanced accuracy. For more information, see the LlamaIndex RAG documentation.

- **Observability and evaluation** – LlamaIndex integrates with a variety of observability and evaluation tools. This integration capability helps you to trace and debug your applications, evaluate their performance, and monitor costs. For more information, see the Tracing and Debugging and Evaluating LlamaIndex documentation.

When to use LlamaIndex

LlamaIndex is particularly well-suited for agentic AI scenarios that emphasize data-intensive workflows and knowledge management:

- Document-heavy applications that require agents to process, analyze, and extract insights from large volumes of enterprise documents such as contracts, reports, manuals, and regulatory filings
- Rapid prototyping to production scenarios where organizations want to quickly build and deploy document-centric agents without extensive infrastructure management overhead
- RAG-first architectures that prioritize retrieval accuracy and context relevance, especially when working with complex, multi-modal documents containing tables, images, and structured data
- Multi-agent document workflows that require specialized agents for different aspects of document processing, such as parsing, analysis, summarization, and compliance checking

Implementation approach for LlamaIndex

LlamaIndex provides both low-level building blocks and high-level abstractions that accommodate different implementation approaches:

- Rapid development of functional RAG applications in just a few lines of code by using LlamaIndex high-level APIs. This approach makes LlamaIndex accessible for business teams and developers who are new to agentic AI.
- Enterprise integration through LlamaHub for popular enterprise systems including SharePoint, Amazon Simple Storage Service (Amazon S3), databases, and APIs. This approach enables integration with existing data infrastructure.
- Flexible deployment options that include open-source self-hosted deployments for greater control, or LlamaCloud managed services for reduced operational overhead and enterprise features.

- Applications can start with simple query engines and progressively add agentic capabilities, multi-agent orchestration, and complex workflows as requirements evolve.

Real-world example of LlamaIndex

This example focuses on a subsidiary of an aerospace company that specializes in aviation navigation and operations solutions. The subsidiary needed to address a growing challenge: uncoordinated AI chatbot pilot projects. These pilots resulted in duplicated work, long development cycles, compliance roadblocks, and isolated implementations across the organization.

The subsidiary developed a unified agent framework, a reusable, template-based solution built on the LlamaIndex open-source framework that makes agent creation more efficient. The team compared several frameworks, both chain-oriented and graph-based. They selected LlamaIndex for three advantages: its flexible design, modular components, and production-ready orchestration controls.

The framework reduced agent development and deployment time by 87%, from 512 to 64 hours, by enabling teams to build agents with approximately 50 lines of code and a JSON configuration file. The teams used a unified framework with built-in security, compliance, and privileged system access.

For more details, see LlamaIndex customer case studies.

Comparing agentic AI frameworks

When selecting an agentic AI framework for autonomous agent development, consider how each option aligns with your specific requirements. Evaluate not only technical capabilities but also organizational fit, including team expertise, existing infrastructure, and long-term maintenance requirements. Many organizations might benefit from a hybrid approach that uses multiple frameworks for different components of their autonomous AI solutions.

The following table compares the maturity levels (strongest, strong, adequate, or weak) of each framework across key technical dimensions. For each framework, the table also includes information about production deployment options and learning curve complexity.

Framework	AWS integration	Autonomous multi-	Autonomous workflow	Multimodal	Foundation model selection	LLM API integration	Production	Learning curve

		agent support	complexity	capabilities			deployment	
Amazon BedrockAgents	Strongest	Adequate	Adequate	Strong	Strong	Strong	Fully managed	Low
AutoGen	Weak	Strong	Strong	Adequate	Adequate	Strong	Do it yourself (DIY)	Steep
CrewAI	Weak	Strong	Adequate	Weak	Adequate	Adequate	DIY	Moderate
LangChain / LangGraph	Adequate	Strong	Strongest	Strongest	Strongest	Strongest	Managed service or DIY	Steep
LlamaIndex	Adequate	Adequate	Strong	Adequate	Strong	Strong	Managed service or DIY	Moderate
Strands Agents	Strongest	Strong	Strongest	Strong	Strong	Strongest	DIY	Moderate

Considerations in choosing an agentic AI framework

When developing autonomous agents, consider the following key factors:

- **AWS infrastructure integration** – Organizations that have invested heavily in AWS can benefit from the native integrations of Strands Agents with AWS services for autonomous workflows. For more information, see AWS Weekly Roundup (AWS Blog).
- **Foundation model selection** – Consider which framework provides strong support for your preferred foundation models (for example, Amazon Nova models on Amazon Bedrock or Anthropic Claude), based on your autonomous agent's reasoning requirements. For more information, see Building Effective Agents on the Anthropic website.
- **LLM API integration** – Evaluate frameworks based on their integration with your preferred large language model (LLM) service interfaces (for example, Amazon Bedrock or OpenAI) for

production deployment. For more information, see Model Interfaces in the Strands Agents documentation.

- **Multimodal requirements** – For autonomous agents that need to process text, images, and speech, consider the multimodal capabilities of each framework. For more information, see Multimodality in the LangChain documentation.
- **Autonomous workflow complexity** – More complex autonomous workflows with sophisticated state management might favor the advanced state machine capabilities of LangGraph.
- **Autonomous team collaboration** – Projects that require explicit role-based autonomous collaboration between specialized agents can benefit from the team-oriented architecture of CrewAI.
- **Autonomous development paradigm** – Teams that prefer conversational, asynchronous patterns for autonomous agents might prefer the event-driven architecture of AutoGen.
- **Managed or code-based approach** – Organizations that want a fully managed experience with minimal coding should consider Amazon Bedrock Agents. Organizations that require deeper customization might prefer Strands Agents or other frameworks with specialized capabilities that align with specific autonomous agent requirements.
- **Production readiness for autonomous systems** – Consider deployment options, monitoring capabilities, and enterprise features for production autonomous agents.

Platforms

Agentic AI platforms provide the foundational runtime, orchestration, and integration layers that you need to deploy, scale, and manage production-grade agentic systems. Frameworks define how you build agents, and protocols govern how agents communicate. Platforms provide the environment where these agents operate and collaborate at scale.

Agentic platforms combine model execution, context management, tool integration, observability, and governance capabilities into unified environments. These platforms enable organizations to move from experimentation to enterprise-scale deployment.

In this section:

- Why platforms matter
- Types of agentic AI platforms
- Platform selection considerations
- Amazon Bedrock Agents
- Amazon Bedrock AgentCore

Why platforms matter

Agentic AI solutions help organizations operationalize autonomous systems in production. They offer the following capabilities:

- Provide **runtime orchestration** for hosting, scaling, and coordinating agents.
- Manage **state, context, and memory** across multi-agent workflows.
- Offer **security, identity, and governance** controls aligned with enterprise standards.
- Integrate with **tools** and external systems through standard APIs or protocols.
- Enable **observability and auditability** across agent interactions and event flows.
- Support **cross-model interoperability**, allowing agents to use multiple foundation models in a single environment.

These capabilities help turn individual agents into coordinated, adaptive systems that are designed to operate within enterprise and regulatory requirements.

Types of agentic AI platforms

Agentic AI solutions typically fall into one or more of the following categories:

- **Managed agent** – Fully managed solutions provide built-in infrastructure, memory, and orchestration capabilities. They help reduce operational overhead and accelerate time to production.
- **Open-source orchestration** – Open-source agentic frameworks offer flexibility and transparency for organizations that prefer customizable environments or on-premises deployment.
- **Hybrid enterprise** – Hybrid solutions integrate managed and self-hosted components, combining the scalability of cloud-managed services with the control of enterprise systems.

Platform selection considerations

When you select or design an agentic AI solution, consider the following:

- **Integration depth** – Evaluate how well the solution integrates with existing data sources, tools, and protocols.
- **Scalability** – Confirm that the solution can scale dynamically to support autonomous workloads and multi-agent collaboration.
- **Security and compliance** – Assess data privacy, encryption, and governance features against organizational and regional requirements.
- **Extensibility** – Choose solutions with modular architectures that let you add new tools, models, or agents over time.
- **Observability** – Choose solutions that provide detailed telemetry, traceability, and audit logs for agentic interactions.
- **Cost efficiency** – Consider serverless or usage-based models to optimize cost for variable workloads.

Amazon Bedrock Agents

Amazon Bedrock Agents is a fully managed service that enables you to build and configure autonomous agents in your applications. It can orchestrate interactions between foundation models, data sources, software applications, and user conversations. Its streamlined approach to

creating agents doesn't require you to provision capacity, manage infrastructure, or write custom code. For more information, see What is Amazon Bedrock?

Key features of Amazon Bedrock Agents

Amazon Bedrock Agents includes the following key features:

- **Fully managed service** – Complete infrastructure management with no need to provision capacity or manage underlying systems. For more information, see Automate tasks in your application using AI agents in the Amazon Bedrock documentation.
- **API-driven development** – Define and run agents through simple API calls by specifying models, instructions, tools, and configuration parameters. For more information, see Create and configure agent manually in the Amazon Bedrock documentation.
- **Action groups** – Define specific actions your agent can perform by creating action groups with API schemas. For more information, see Use action groups to define actions for your agent to perform in the Amazon Bedrock documentation.
- **Knowledge base integration** – Connect to Amazon Bedrock Knowledge Bases to augment agent responses with your organization's data. For more information, see Augment response generation for your agent with knowledge base in the Amazon Bedrock documentation.
- **Advanced prompt templates** – Customize agent behavior through prompt templates for pre-processing, orchestration, knowledge base response generation, and post-processing. For more information, see Enhance agent's accuracy using advanced prompt templates in Amazon Bedrock in the Amazon Bedrock documentation.
- **Tracing and observability** – Track the agent's step-by-step reasoning process using built-in tracing capabilities. For more information, see Track agent's step-by-step reasoning process using trace in the Amazon Bedrock documentation.
- **Versioning and aliases** – Create multiple versions of your agent and deploy them through aliases for controlled rollouts. For more information, see Deploy and use an Amazon Bedrock agent in your application in the Amazon Bedrock documentation.

When to use Amazon Bedrock Agents

Amazon Bedrock Agents is particularly well-suited for autonomous agent scenarios including:

- Organizations that want a fully managed experience for building and deploying agents without managing infrastructure

- Projects that require rapid development and deployment of agents through configuration rather than code
- Use cases that benefit from integration with other Amazon Bedrock capabilities, such as Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails
- Teams without the in-house resources to build agents from scratch but need production-ready autonomous capabilities

Implementation approach for Amazon Bedrock Agents

Amazon Bedrock Agents provides a configuration-based implementation approach for business stakeholders. The service enables organizations to:

- Define agents through the AWS Management Console or API calls without writing complex code.
- Create action groups that specify the APIs and operations that the agent can perform.
- Connect knowledge bases to provide domain-specific information to the agent.
- Test and iterate on agent behavior through a visual interface.

This managed approach allows business teams to rapidly develop and deploy autonomous agents without deep technical expertise in AI model development or infrastructure management.

Real-world example of Amazon Bedrock Agents

A financial operations (FinOps) solution described in this AWS blog post uses the Amazon Bedrock multi-agent framework to create an AI-driven cloud cost management assistant. The Amazon Nova foundation model powers the solution, where a central FinOps Supervisor agent delegates tasks to specialized agents. These agents fetch and analyze AWS spend data by using AWS Cost Explorer and generate cost-saving recommendations by using AWS Trusted Advisor.

The system includes user authentication through Amazon Cognito, a front-end hosted on AWS Amplify, and AWS Lambda action groups for real-time analysis and forecasting. Finance teams can ask natural language queries such as "What were my costs in February 2025?" The system responds with detailed breakdowns, optimization suggestions, and forecasts within a scalable, serverless architecture that is deployed by using AWS CloudFormation.

Amazon Bedrock AgentCore

Amazon Bedrock AgentCore is an agentic service that helps you build, deploy, and operate highly capable agents securely at scale by using any framework, model, or protocol. With AgentCore, you can do the following, all without infrastructure management:

- Build agents faster.
- Enable agents to take actions across tools and data.
- Run agents securely with low-latency and extended runtimes.
- Monitor agents in production.

AgentCore reduces the effort of building specialized agent infrastructure, so you can move your agents to production faster. You can use its services together or independently, and they are compatible with frameworks such as CrewAI, LangGraph, LlamaIndex, and Strands Agents. AgentCore is also compatible with foundation models that are available in or outside of Amazon Bedrock, providing flexibility in your model choice.

AgentCore is composed of several key services:

- Amazon Bedrock AgentCore Runtime – Provides a secure, serverless, scalable environment to host and run your agents, without needing to manage any infrastructure required for deploying and running AI agents or tools.
- Amazon Bedrock AgentCore Memory – Offers a managed memory system, enabling agents to retain context from interactions for more personalized and coherent conversations by maintaining both immediate and long-term knowledge.
- Amazon Bedrock AgentCore Gateway – Simplifies the process of creating, securing, and finding the right tools for agents. With AgentCore Gateway, developers can convert APIs, AWS Lambda functions, and existing services into Model Context Protocol (MCP)-compatible tools and make them available to agents.
- Amazon Bedrock AgentCore Identity – Provides a secure, scalable agent identity and access management service that accelerates AI agent development. With AgentCore Identity, you can assign unique, verifiable identities to agents, enabling fine-grained access control and secure agent-powered interactions with enterprise systems.
- Amazon Bedrock AgentCore built-in tools – Enables you to use built-in tools to enhance your development and testing workflow. Use these tools to interact with your application, enabling AI

agents to write and run code securely in sandbox environments. Use the browser tool to enable AI agents to interact with websites at scale.

- Amazon Bedrock AgentCore Observability – Delivers logging and monitoring capabilities, giving you real-time visibility into your agent's performance and behavior to facilitate debugging and optimization.

Key features of AgentCore

AgentCore includes the following key features:

- **Fully managed and extensible** – AgentCore is a fully managed service, which means that AWS handles the underlying infrastructure and maintenance. It is also extensible, which allows you to customize and enhance the functionality of your agents. For more information, see Get started with Amazon Bedrock AgentCore Runtime in the AgentCore documentation.
- **Long-term and short-term memory** – Deliver more personalized and relevant interactions by equipping agents with a memory system to recall context from current conversations and long-term knowledge. For more information, see Get started with AgentCore Memory in the AgentCore documentation.
- **Simplified tool development and integration** – Enable your agents to discover and use tools through a single, secure endpoint. Quickly turn your existing enterprise resources into agent-ready tools with just a few lines of code, freeing developers to focus on building unique capabilities. For more information, see Get started with AgentCore Gateway in the AgentCore documentation.
- **Secure and scalable infrastructure** – AgentCore provides a secure and scalable environment for deploying and operating agents. It includes features for identity and access management, data encryption, and network security. For more information, see Get started with Amazon Bedrock AgentCore Identity in the AgentCore documentation.
- **Integration with a wide range of tools** – Allows you to integrate your agents with a variety of tools, including a code interpreter and a browser tool that you can build by using the AgentCore built-in tools. For more information, see Get started with AgentCore Code Interpreter and Get started with AgentCore Browser in the AgentCore documentation.
- **Comprehensive observability and monitoring** – Gain deep visibility into your agents with comprehensive tools to trace, debug, and monitor their performance in production. Visualize the agent's entire execution path to audit its reasoning and resolve failures. Use real-time dashboards and standardized telemetry data to track key operational metrics. For more

information, see Add observability to your Amazon Bedrock AgentCore resources in the AgentCore documentation.

When to use AgentCore

AgentCore is particularly well-suited for autonomous agent scenarios including:

- Organizations that want to accelerate development and lower operational overhead with a fully managed service that handles infrastructure, security, built-in tools, observability, and scaling
- Projects that need flexibility with modular services that work together or independently and are compatible with any framework, like CrewAI or LangGraph, and any foundation model from any source
- Use cases that require stateful, conversational agents that need to maintain context and learn from past interactions to provide personalized and relevant responses
- Agents that perform complex tasks through simple integration with diverse applications, data sources, and APIs

Implementation approach for AgentCore

AgentCore is designed for organizations that want to move AI agents from proof of concept to production, including agents built by using open source or custom agent frameworks. With AgentCore, organizations can do the following:

- Deploy agents securely on serverless infrastructure, supporting any framework and model, with session isolation and built-in identity and access management to help support your security and compliance requirements. Quickly create AgentCore Runtime agents for leading agent frameworks by using the starter toolkit.
- Enhance agents by integrating persistent memory for context retention, and simplify tool development and integration through AgentCore Gateway. Use the built-in browser tool and code interpreter for advanced workflows.
- Trace, debug, and monitor AI agents in production by using observability dashboards powered by Amazon CloudWatch and OpenTelemetry, and track key metrics of AgentCore resources (runtime, memory, gateway, and tools).
- Accelerate deployment and innovation with fully managed, modular services. Use the services as composable building blocks together or independently with any agent framework and model provider. This flexibility helps organizations move from prototype to production faster.

This managed approach enables organizations to quickly and securely build, deploy, and run enterprise-grade AI agents and multi-agent systems at any scale.

Real-world example of AgentCore

AWS has observed that one of Latin America's largest banks has used AI/ML for years to deliver a personalized and secure digital banking experience. The bank is expanding its agentic AI services by using AgentCore to provide customers with intuitive interactions, enhanced security, and greater automation. AgentCore is expected to support the bank's efforts to meet customer commitments at scale. AgentCore provides the bank's developers with the tools and flexibility to build and manage agents, while helping support compliance with financial regulations.

Protocols

AI agents require standardized communication protocols to interact with other agents and services. Organizations implementing agent architectures face significant challenges around interoperability, vendor independence, and future-proofing their investments.

This section helps you choose between agent-to-agent protocols and agent-tool protocols, with a focus on open standards that support flexibility and interoperability.

In this section:

- Why protocol selection matters
- Agent-to-agent protocols
- Selecting agentic protocols
- Implementation strategy for agentic protocols
- Getting started with MCP
- Getting started with A2A

Why protocol selection matters

Protocol selection fundamentally shapes how you can build and evolve your AI agent architecture. By choosing protocols that support portability between agent frameworks, you gain the flexibility to combine different agent systems and workflows to meet your specific needs.

Open protocols enable you to integrate agents across multiple frameworks. For example, use LangChain for rapid prototyping and implement production systems with Strands Agents, communicating through a common protocol, such as the Model Context Protocol (MCP) or the Agent2Agent (A2A) protocol. This flexibility reduces dependency on specific AI providers, simplifies integration with existing systems, and enables you to enhance agent capabilities over time.

Well-designed protocols also establish consistent security patterns for authentication and authorization across your agents. Protocol portability helps you adopt new agent frameworks and capabilities as they emerge. Choosing open protocols helps protect your investment in agent development while maintaining interoperability with third-party systems.

Advantages of open protocols

When implementing your own extensions or building custom agent systems, open protocols offer compelling advantages:

- **Documentation and transparency** – Typically provide comprehensive documentation and transparent implementations
- **Community support** – Access to broader developer communities for troubleshooting and best practices
- **Interoperability** – Increased likelihood that your extensions work across different implementations
- **Future compatibility** – Reduced risk of breaking changes or deprecation
- **Influence on development** – Opportunity to contribute to protocol evolution

Agent-to-agent protocols

The following table provides an overview of agentic protocols that enable multiple agents to collaborate, delegate tasks, and share information.

Protocol	Ideal for	Considerations
MCP inter-agent communication	Organizations seeking flexible agent collaboration patterns	• An extension to the Model Context Protocol (MCP) proposed by AWS that builds on its existing foundation for agent-to-agent communication • Enables agent collaboration with OAuth-based authentication and authorization
Agent2Agent (A2A) protocol	Agent collaboration across different frameworks and vendors	• Backed by Google • Newer standard with more limited adoption compared to MCP

Deciding among protocol options

When implementing agent-to-agent communication, match your specific communication requirements with protocol capabilities. Different interaction patterns require different protocol features. The following table outlines common communication patterns and suggests protocol choices for each scenario.

Pattern	Description	Ideal protocol choice
Simple request and response	One-off interactions between agents	MCP with stateless flows
Stateful dialogues	Ongoing conversations with context	MCP with session management
Multi-agent collaboration	Complex interactions between multiple agents	MCP inter-agent or AutoGen
Team-based workflows	Hierarchical agent teams with defined roles	MCP inter-agent, CrewAI, or AutoGen

Beyond communication patterns, several technical and organizational factors can influence your protocol selection. The following table outlines key considerations that can help you evaluate which protocol aligns most closely with your specific implementation requirements.

Consideration	Description	Example
Security model	Authentication and authorization requirements	OAuth 2.0 in MCP
Deployment environment	Where agents will run and communicate	Distributed or single machine
Framework compatibility	Integration with existing agent frameworks	LangChain or Strands Agents
Scalability needs	Expected growth in agent interactions	Streaming capabilities of MCP

Selecting agentic protocols

For most organizations building production agent systems, the Model Context Protocol (MCP) offers a comprehensive and well-supported foundation for agent-to-agent communication. MCP benefits from active development contributions from AWS and the open-source community.

Selecting the right agentic protocols is important for organizations looking to implement agentic AI effectively. Considerations differ based on organizational context.

Agentic protocol selection considerations

Organizations should consider the following best practices when selecting protocols for agentic AI systems:

- **Prioritize open standards** – Organizations should adopt open protocols such as MCP to help support long-term interoperability and extensibility, and to increase flexibility in choosing technology providers.
- **Evaluate protocol maturity** – All organizations should assess the health, adoption, and community support of each protocol to help support sustainability and reduce technical debt.
- **Use managed services** – Where possible, use managed or serverless implementations of agentic protocols to reduce operational complexity and accelerate deployment.

Implementation strategy for agentic protocols

To effectively implement agentic protocols across your organization, consider the following strategic steps:

1. **Start with standards alignment** – Adopt established open protocols where possible.
2. **Create abstraction layers** – Implement adapters between your systems and specific protocols.
3. **Contribute to open standards** – Participate in protocol development communities.
4. **Monitor protocol evolution** – Stay informed about emerging standards and updates.
5. **Test interoperability regularly** – Verify that your implementations remain compatible.

Getting started with MCP

AWS actively supports the Model Context Protocol (MCP) through contributions to the protocol's development and implementation. AWS is collaborating with leading open-source agent frameworks, including LangGraph, CrewAI, and LlamaIndex, to shape the future of inter-agent communication on the protocol.

To implement the MCP in your agent architecture, take the following actions:

1. Explore MCP implementations in frameworks like the Strands Agents SDK.
2. Review the Model Context Protocol technical documentation.
3. Read Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP (AWS Blog) to learn about agent interoperability.
4. Join the MCP community to influence the protocol's evolution.

MCP provides a communication layer that enables agents to interact with external data and services, and you can also use it to enable agents to interact with other agents. The protocol's Streamable HTTP transport implementation gives you a set of interaction patterns so that you don't have to build custom communication mechanisms. These patterns support both stateless request/response flows and stateful session management with persistent IDs.

By adopting open protocols like MCP, you position your organization to build agent systems that remain flexible, interoperable, and adaptable as AI technology evolves.

For information about agent-to-tool protocol implementation, see Tool integration strategy later in this guide.

Getting started with A2A

The Agent2Agent (A2A) protocol enables decentralized collaboration between agents through a shared semantic layer. Instead of routing all work through a central orchestrator, A2A allows agents to discover each other, advertise their capabilities, negotiate tasks, and share context using a lightweight JSON-based protocol. Each agent publishes a capability manifest.

The following example shows a simplified A2A capability manifest that advertises an agent's supported actions, required inputs, and operational metadata to enable discovery and task negotiation:

```
{
  "can": ["summarize.text", "extract.keywords"],
  "needs": ["document.input"],
  "meta": { "version": "1.0.3", "latencyMs": 120 }
}
```

This model enables dynamic capability matching, mid-task delegation, and cross-organizational collaboration. Agents can self-organize around tasks, form temporary working groups, and adapt as new capabilities enter or exit the system.

A2A supports interactions ranging from simple stateless requests to multi-step negotiation sessions. These include:

- **Direct peer-to-peer messaging** for low-latency collaboration
- **Semantic task negotiation**, where agents select the most suitable peer
- **Capability-based discovery**, enabling emergent division of labor
- **Session anchoring** for stateful multi-step interactions

By adopting open, agent-native protocols like A2A, organizations create AI systems that are modular, interoperable, and capable of cross-boundary collaboration. A2A helps agent systems remain flexible and evolvable as new agents, teams, or external systems are introduced, without requiring rigid orchestration layers or prior coupling.

To implement the A2A protocol in your agent architecture, take the following actions:

1. **Review the A2A Protocol Specification** – Read the latest version of the Agent2Agent (A2A) Protocol Specification to learn how capability manifests, negotiation flows, and the agent handshake operate.
2. **Explore A2A-compatible runtimes** – Evaluate frameworks such as the Strands Agents SDK, or custom runtime layers that support A2A-style capability manifests and peer-to-peer negotiation.
3. **Implement a capability manifest for your agents** – Define each agent's `can`, `needs`, and `meta` fields to enable discovery, matchmaking, and intent-level collaboration.
4. **Experiment with A2A negotiation patterns** – Use the request–offer–accept loop, structured capability queries, or gossip-based discovery to understand how agents reason about who should handle a task.

5. **Test A2A in a mixed infrastructure environment** – Combine A2A peer negotiation with event routing that's native to AWS through Amazon EventBridge to evaluate hybrid coordination patterns.
6. **Join the A2A community** – Engage with the open working group to stay up to date with extensions, security recommendations, and cross-vendor interoperability improvements, and contribute to the development of the protocol.

Tools

AI agents deliver value by interacting with external tools, APIs, and data sources to perform useful tasks. The right tool integration strategy directly impacts your agent's capabilities, security posture, and long-term flexibility.

This section helps you navigate the tool integration landscape with a focus on open standards that support flexibility and portability. This section highlights the Model Context Protocol (MCP) for tool integration and reviews framework-specific tools and specialized meta-tools that enhance agent workflows.

In this section:

- Tool categories
- Protocol-based tools
- Framework-native tools
- Meta-tools
- Tool integration strategy
- Security best practices for tool integration

Tool categories

Building agent systems involves three main categories of tools.

Protocol-based tools

Protocol-based tools use standardized protocols for agent-to-tool communication:

- **MCP tools** – Open standard tools that work across frameworks with both local and remote execution options.
- **OpenAI function calling** – Proprietary tools that are specific to OpenAI models.
- **Anthropic tool use** – A variation on OpenAI function calling for proprietary tools specific to Anthropic Claude models.

Framework-native tools

Framework-native tools are built directly into specific agent frameworks:

- **Strands Agents tools** – Lightweight, quick-to-implement tools that are specific to the Strands Agents framework.
- **LangChain tools** – Python-based tools that integrate tightly with the LangChain framework.
- **LlamaIndex tools** – Tools designed for data retrieval and processing within LlamaIndex.

Meta-tools

Meta-tools enhance agent workflows without directly taking external actions:

- **Workflow tools** – Manage agent execution flow, branching logic, and state management.
- **Agent graph tools** – Coordinate multiple agents in complex workflows.
- **Memory tools** – Provide persistent storage and retrieval of information across agent sessions.
- **Reflection tools** – Enable agents to analyze and improve their own performance.

Protocol-based tools

When considering protocol-based tools, the Model Context Protocol (MCP) provides a comprehensive and flexible foundation for tool integration. As stated in the AWS Open Source blog post on agent interoperability, AWS has embraced MCP as a strategic protocol, actively contributing to its development.

The following table describes options for MCP tool deployment.

Deployment model	Description	Ideal for	Implementation
Local stdio-based	Tools run in the same process as the agent	Development, testing, and simple tools	Quick to implement with no network overhead
Local server-sent events (SSE)-based (legacy)	Tools run locally but communicate over HTTP	More complex local tools with separation of concerns	Better isolation but still low latency

Remote HTTP Streamable	Tools run on remote servers	Production environments and shared tools	Scalable and centrally managed

The official MCP SDKs are available for building MCP tools:

- Python SDK – Comprehensive implementation with full protocol support
- TypeScript SDK – JavaScript/TypeScript implementation for web applications
- Java SDK – Java implementation for enterprise applications

These SDKs provide the building blocks for creating MCP-compatible tools in your preferred language, with consistent implementations of the protocol specification.

In addition, AWS has implemented MCP in the Strands Agents SDK. The Strands Agents SDK provides a straightforward way to create and use MCP-compatible tools. Comprehensive documentation is available in the Strands Agents GitHub repository. For simpler use cases or when working outside of the Strands Agents framework, the official MCP SDKs offer direct implementations of the protocol in multiple languages.

Security features of MCP tools

Security features of MCP tools include the following:

- **OAuth 2.0/2.1 authentication** – Industry-standard authentication
- **Permission scoping** – Fine-grained access control for tools
- **Tool capability discovery** – Dynamic discovery of available tools
- **Structured error handling** – Consistent error patterns

Getting started with MCP tools

To implement MCP for tool integration, take the following actions:

1. Explore the Strands Agents SDK for a production-ready MCP implementation.
2. Review the MCP technical documentation to understand core concepts.
3. Use the practical examples described in this AWS Open Source Blog post.

4. Start with simple local tools before progressing to remote tools.
5. Join the MCP community to influence the protocol's evolution.

Explore AgentCore Gateway

Amazon Bedrock AgentCore Gateway helps developers build, deploy, discover, and connect to MCP tools and other target endpoints at scale. With AgentCore Gateway, developers can convert APIs, AWS Lambda functions, and existing services into MCP-compatible tools. Then, developers can make these tools available to agents through AgentCore Gateway endpoints with minimal code. AgentCore Gateway supports OpenAPI, Smithy, and Lambda as input types, and provides both ingress authentication and egress authentication in a fully managed service.

Framework-native tools

Although the Model Context Protocol (MCP) provides a flexible foundation, framework-native tools offer advantages for specific use cases.

The Strands Agents SDK offers lightweight, Python-based tools that require minimal overhead for simple operations. You can create tools with just a few lines of code. These tools are also tightly integrated with the Strands Agents framework.

The following example demonstrates how to create a simple weather tool using Strands Agents. You can transform Python functions into agent-accessible tools with minimal code, and Strands Agents automatically generates tool documentation from the function's docstring.

```
#Example of a simple Strands native tool

@tool

def weather(location: str) -> str:

"""Get the current weather for a location""" #

Implementation here

return f"The weather in {location} is sunny."
```

For rapid prototyping or simple use cases, framework-native tools can accelerate development. However, for production systems, MCP tools provide greater interoperability across frameworks than framework-native tools.

The following table provides an overview of other framework-specific tools.

Framework	Tool type	Advantages	Considerations
AutoGen	Function definitions	Strong multi-agent support	Best suited to Microsoft tooling
LangChain	Python classes	Large library of pre-built tools	Tools are specific to the LangChain framework
LlamaIndex	Python functions	Optimized for data operations	Limited to LlamaIndex

Meta-tools

Meta-tools don't directly interact with external systems. Instead, they enhance agent capabilities by implementing agentic patterns. This section discusses workflow, agent graph, and memory meta-tools.

Workflow meta-tools

Workflow meta-tools manage the flow of agent execution:

- **State management** – Maintain context across multiple agent interactions
- **Branching logic** – Enable conditional execution paths
- **Retry mechanisms** – Handle failures with configurable retry strategies

Example frameworks with workflow meta-tools include LangGraph and Strands Agents workflow capabilities.

Agent graph meta-tools

Agent graph meta-tools coordinate multiple agents working together:

- **Task delegation** – Assign subtasks to specialized agents
- **Result aggregation** – Combine outputs from multiple agents

- **Conflict resolution** – Resolve disagreements between agents

Frameworks like AutoGen and CrewAI specialize in agent graph coordination.

Memory meta-tools

Memory meta-tools provide persistent storage and retrieval:

- **Conversation history** – Maintain context across sessions
- **Knowledge bases** – Store and retrieve domain-specific information
- **Vector stores** – Enable semantic search capabilities

The resource system in Model Context Protocol (MCP) provides a standardized way to implement memory meta-tools that work across different agent frameworks.

Tool integration strategy

Your choice of tool integration strategy directly affects what your agents can accomplish and how easily your system can evolve. Prioritize open protocols like the Model Context Protocol (MCP) while strategically using framework-native tools and meta-tools. That way, you can build a set of tools that remains flexible as AI technology advances.

The following strategic approach to tool integration helps you increase flexibility while meeting your organization's immediate needs:

1. **Adopt MCP as your foundation** – MCP provides a standardized way to connect agents to tools and supports authentication and authorization. Start with MCP as your primary tool protocol for:
   - Strategic tools that multiple agent implementations use.
   - Security-sensitive tools that require authentication and authorization.
   - Tools that run remotely in production environments.
2. **Use framework-native tools when appropriate** – Consider framework-native tools for:
   - Rapid prototyping during initial development.
   - Simple non-critical tools with minimal security requirements.
   - Framework-specific functionality that uses unique capabilities.

3. **Implement meta-tools for complex workflows** – Add meta-tools to enhance your agent architecture:
   - Start simple with basic workflow patterns.
   - Add complexity as your use cases mature.
   - Standardize interfaces between agents and meta-tools.
4. **Plan for evolution** – Build with future flexibility in mind:
   - Document tool interfaces independently of implementations.
   - Create abstraction layers between agents and tools.
   - Establish migration paths from proprietary to open protocols.

Security best practices for tool integration

Tool integration directly impacts your security posture. This section outlines best practices to consider for your organization.

Authentication and authorization

Use the following access controls:

- **Use OAuth 2.0/2.1** – Implement industry-standard authentication for remote tools.
- **Implement least privilege** – Grant tools only the permissions they need.
- **Rotate credentials** – Regularly update API keys and access tokens.

Data protection

To help safeguard data, adopt the following measures:

- **Validate inputs and outputs** – Implement schema validation for all tool interactions.
- **Encrypt sensitive data** – Use TLS for all remote tool communications.
- **Implement data minimization** – Only pass necessary information to tools.

Monitoring and auditing

Maintain visibility and control by using these mechanisms:

- **Log all tool invocations** – Maintain comprehensive audit trails.
- **Monitor for anomalies** – Detect unusual tool usage patterns.
- **Implement rate limiting** – Prevent abuse through excessive tool calls.

The Model Context Protocol (MCP) security model provides guidance for addressing these concerns. For more information, see Security considerations in the MCP documentation.

Conclusion

The landscape of agentic AI continues to evolve rapidly, offering organizations powerful new ways to build intelligent, autonomous systems. This guide has explored four essential components for successful implementation: *frameworks* that provide the foundation, *platforms* that provide the environment, *protocols* that enable communication, and *tools* that extend capabilities.

As frameworks mature, you can expect increased interoperability, standardization around protocols like Model Context Protocol (MCP), and more sophisticated orchestration capabilities for autonomous agents. Organizations that establish expertise with these frameworks today will be well-positioned to build increasingly autonomous, intelligent agents that deliver significant business value.

Platforms provide the execution, governance, and lifecycle environment in which agentic systems operate. They handle concerns such as identity, security boundaries, observability, memory management, session grounding, and safe interaction with tools and data. In AWS environments, platforms like managed agent runtimes and orchestration services allow organizations to deploy, monitor, evolve, and govern autonomous agents and agentic systems at scale. Platforms bridge foundational frameworks with real-world operational requirements.

The choice of agent protocols represents a strategic decision that balances immediate development needs with long-term flexibility and interoperability. By prioritizing open protocols and creating appropriate abstraction layers, organizations can build agent systems that remain adaptable to evolving technologies while meeting current business requirements.

For most organizations, MCP represents a strong foundation due to its open standard, growing community, support for agent-to-agent communication patterns, and tool integration capabilities. AWS supports MCP and Agent2Agent (A2A) as strategic protocols, contributes to their development, and implements them in offerings such as the Strands Agents SDK. By using MCP or A2A alongside appropriate framework-native tools and meta-tools, you can build agent systems that deliver immediate value while remaining adaptable to future innovations.

Resources

Use the following AWS and other resources related to autonomous agent development.

AWS Blogs

- Amazon Bedrock AgentCore Memory: Building context-aware agents
- Best practices for building robust generative AI applications with Amazon Bedrock Agents – Part 1
- Best practices for building robust generative AI applications with Amazon Bedrock Agents – Part 2
- Build powerful RAG pipelines with LlamaIndex and Amazon Bedrock
- Build trustworthy AI agents with Amazon Bedrock AgentCore Observability
- Evaluate RAG responses with Amazon Bedrock, LlamaIndex, and RAGAS
- Introducing the Amazon Bedrock AgentCore Code Interpreter
- Introducing Amazon Bedrock AgentCore Gateway: Transforming enterprise AI agent tool development
- Introducing Amazon Bedrock AgentCore Identity: Securing agentic AI at scale
- Introducing Strands Agents, an Open Source AI Agents SDK
- Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP
- Securely launch and scale your agents and tools on Amazon Bedrock AgentCore Runtime
- AWS Transform for .NET, the first agentic AI service for modernizing .NET applications at scale
- AWS Weekly Roundup: Strands Agents

AWS Prescriptive Guidance

- Agentic AI patterns and workflows on AWS
- Building multi-tenant architectures for agentic AI on AWS
- Building serverless architectures for agentic AI on AWS
- Foundations of agentic AI on AWS
- Operationalizing agentic AI on AWS

- Retrieval Augmented Generation options and architectures on AWS
- Security for agentic AI on AWS

AWS resources

- Amazon Bedrock documentation
- Amazon Bedrock AgentCore documentation
- Amazon Bedrock AgentCore Starter Toolkit (GitHub repository)
- Amazon Nova documentation
- AWS MCP Servers (GitHub repository)

Other resources

- AutoGen documentation (Microsoft)
- Building effective agents (Anthropic)
- CrewAI GitHub repository
- LangChain documentation
- LangGraph platform
- LlamaIndex documentation
- Model Context Protocol documentation
- Strands Agents documentation
- Strands Agents tools overview
- Strands Agents quickstart guide

Document history

The following table describes significant changes to this guide.

Change	Description	Date
New section	Added Platforms section	January 16, 2026
Initial publication	—	July 14, 2025