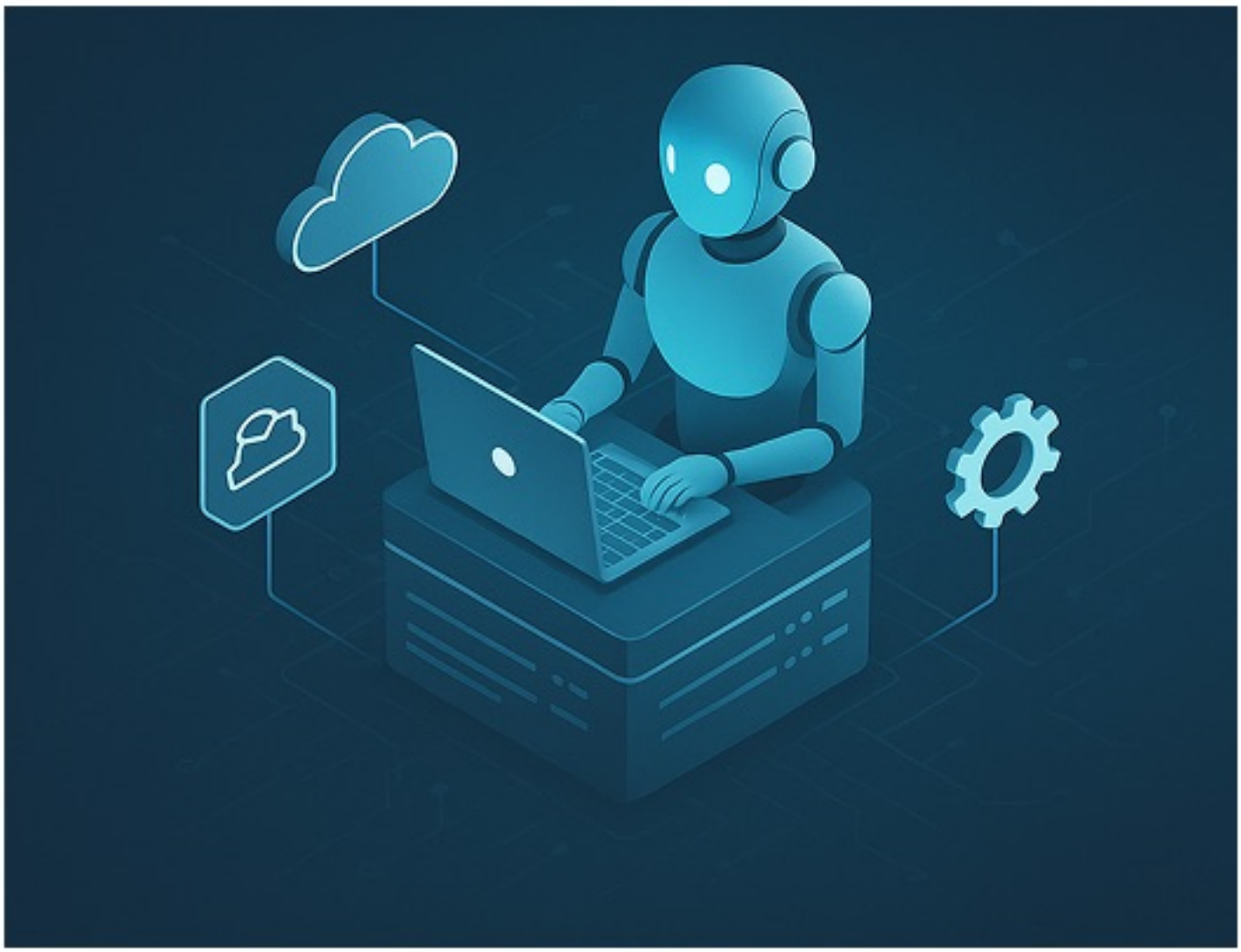
How to build AI AGENTS in cloud native systems

Agentic AI frameworks, protocols, and tools on AWS



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Table of Contents

Agentic AI frameworks, protocols, and tools on AWS

Aaron Sempf and Joshua Samuel, Amazon Web Services (AWS)

July 2025 (document history)

Agentic AI is a powerful paradigm at the intersection of AI, distributed systems, and software engineering. It's a class of intelligent systems that's composed of autonomous, asynchronous software agents. The agents exhibit agency, can 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 leverage agentic AI to automate complex workflows, enhance decision-making processes, and create more responsive systems. This guide provides information about key components that are necessary to build effective agentic AI solutions:

- Agentic AI 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.
- Agentic protocols explores essential 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 protocols to interact seamlessly.
- Tools provides information about protocol-based tools (such as the MCP), 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 seeking to harness the power of AI-driven software agents within modern cloud-native applications.

Objectives

This guide helps you do the following:

- Compare different agentic AI frameworks to select the most appropriate one for your use case.
- Understand the advantages of open protocols for building sustainable agentic AI architectures.
- Create an appropriate tool integration strategy when building agent systems.

About this content series

This guide is part of a set of publications that provide architectural blueprints and technical guidance for building AI-driven software agents on AWS. The AWS Prescriptive Guidance series includes the following guides:

- Foundations of agentic AI on AWS
- Agentic AI patterns and workflows on AWS
- *Agentic AI frameworks, protocols, and tools on AWS* (this guide)
- Building serverless architectures for agentic AI on AWS
- Building multi-tenant architectures for agentic AI on AWS

For more information about this content series, see Agentic AI.

Agentic AI frameworks

Foundations of agentic AI on AWS examines the core patterns and workflows that enable autonomous, goal-directed behavior. At the heart of implementing these patterns lies the choice of framework. A *framework* is the software foundation that provides the structure, tools, and orchestration capabilities needed to build production-ready autonomous AI agents.

Effective agentic AI frameworks provide several essential capabilities that transform raw large language model (LLM) interactions into robust, autonomous agents capable of independent operation:

- **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 autonomous tasks. 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 LangGraph Platform GA 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

- AutoGen
- 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

Strands Agents is an open-source SDK that was initially released by AWS, as described in the AWS Open Source Blog. Strands Agents is designed for building autonomous AI agents with a model-first approach It provides a flexible, extensible framework designed to work seamlessly with AWS services while remaining open to integration with third-party components. Strands Agents is ideal for building fully 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.
- **MCP integration** – Native support for the Model Context Protocol (MCP), enabling standardized context provision to LLMs for consistent autonomous operation.
- **AWS service integration** – Seamless connection to Amazon Bedrock, AWS Lambda, AWS Step Functions, and other AWS services for comprehensive 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 to optimize 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 comprehensive autonomous agent interactions. For more information, see Amazon Bedrock Multimodal Support in the Strands Agents documentation.
- **Tool ecosystem** – Rich 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 enterprise-grade 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 tight integration with existing AWS workflows and resources for end-to-end autonomous processes

Implementation approach for Strands Agents

Strands Agents provides a straightforward implementation approach for business stakeholders, as outlined in its Quickstart Guide. The framework allows organizations to:

- 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 enables business teams to rapidly develop and deploy autonomous agents without deep technical expertise in AI model development.

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 employs multiple specialized autonomous agents. The agents work together to analyze legacy .NET applications, plan modernization strategies, and execute code transformations to cloud-native architectures without human intervention. AWS Transform for .NET demonstrates the production readiness of Strands Agents for enterprise autonomous systems.

LangChain and LangGraph

LangChain is one of the most established frameworks in the agentic AI ecosystem. LangGraph extends its capabilities to support complex, stateful agent workflows as described in the LangChain Blog. Together, they provide a comprehensive solution for building sophisticated autonomous AI agents with rich orchestration capabilities for independent operation.

Key features of LangChain and LangGraph

LangChain and LangGraph include the following key features:

- **Component ecosystem** – Extensive library of pre-built 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 and long-term memory management, which is essential for autonomous agents that maintain context over time. For more information, see How to add memory to chatbots in the LangChain documentation.
- **Tool integration** – Rich ecosystem of tool integrations across various services and APIs, extending 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 large ecosystem 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

Vodafone has implemented autonomous agents using LangChain (and LangGraph) to enhance its data engineering and operations workflows, as detailed in their LangChain Enterprise case study. They built internal AI assistants that autonomously monitor performance metrics, retrieve information from documentation systems, and present actionable insights—all through natural language interactions.

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

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** – Autonomous agents are defined with specific roles, goals, and back stories 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 appropriate 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, see LLMs in the CrewAI documentation.
- **LLM API integration** – Flexible integration with multiple 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 implementation of teams of AI agents approach for business stakeholders, as detailed 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 areas that collaborate to achieve business objectives, similar to how human teams operate. However, the autonomous team 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 detailed in the CrewAI published case study. AWS and CrewAI developed a secure, vendor-neutral framework. The CrewAI open-source "flows-and-crews" architecture seamlessly 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.
- **Demonstrated return on investment (ROI)** – Early pilots showcase major improvements—70#percent faster execution for a large code modernization project and about 90#percent reduction in processing time for a CPG back-office flow.

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.

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** – Seamlessly 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 tight integration with other Amazon Bedrock capabilities like Knowledge Bases and 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 cost-effective 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 secure user access 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—all within a scalable, serverless architecture deployed by using AWS CloudFormation.

AutoGen

AutoGen is an open-source framework that was released initially by Microsoft. 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** – Strong 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 execution** – 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.
- **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, execution, 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 achieves state-of-the-art performance across benchmarks like GAIA, AssistantBench, and WebArena—all without task-specific tuning. Additionally, it supports modular extensibility and rigorous evaluation through AutoGenBench suggestions.

Comparing agentic AI frameworks

When selecting an agentic AI framework for autonomous agent development, consider how each option aligns with your specific requirements. Consider not only its technical capabilities but also its organizational fit, including team expertise, existing infrastructure, and long-term maintenance

requirements. Many organizations might benefit from a hybrid approach, leveraging multiple frameworks for different components of their autonomous AI ecosystem.

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-agent support	Autonomous workflow complexity	Multimodal capabilities	Foundation model selection	LLM API integration	Production deployment	Learning curve
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	Platform or DIY	Steep
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 heavily invested in AWS will benefit most 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 the best 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 better align with specific autonomous agent requirements.
- **Production readiness for autonomous systems** – Consider deployment options, monitoring capabilities, and enterprise features for production autonomous agents.

Agentic 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 navigate the agent-to-agent protocol landscape with a focus on open standards that maximize flexibility and interoperability. (For information about agent-to-tool protocols, see Tool integration strategy later in this guide.)

This section highlights the Model Context Protocol (MCP), an open standard originally developed by Anthropic in 2024. Today, AWS actively supports 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. For more information, see Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP (AWS Blog).

In this section

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 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 agent ecosystem. Most importantly, protocol portability preserves your freedom to adopt new agent frameworks and capabilities as they emerge. Choosing open protocols protects 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 guarantees** – Better assurance that your extensions will 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 seamless agent collaboration with OAuth-based security

A2A protocol	Cross-platform agent ecosystems	• Backed by Google • Newer standard with more limited adoption compared to MCP
AutoGen multi-agent	Research-focused multi-agent systems	• Backed by Microsoft • Strong for complex agent interactions
CrewAI	Role-based agent teams	• Independent implement ation • Good for simulating organizational structures

Deciding among protocol options

When implementing agent-to-agent communication, match your specific communication requirements with the appropriate protocol capabilities. Different interaction patterns require different protocol features. The following table outlines common communication patterns and recommends the most suitable 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
Ecosystem 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 the most 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.

Enterprise protocol considerations

Enterprises should consider the following actions:

- Prioritize open protocols like MCP for strategic, long-term agent implementations.
- Implement abstraction layers when using proprietary protocols to ease future migrations.
- Participate in standards development to influence protocol evolution.
- Consider hybrid approaches that use open protocols for core infrastructure and proprietary protocols for specific use cases.

Startup and SMB protocol considerations

Startups and small-to-medium (SMB) businesses should consider the following actions:

- Balance speed and flexibility by starting with well-supported proprietary protocols for rapid development.
- Plan migration paths to use more open standards as your needs mature.
- Evaluate protocol adoption trends to avoid investing in declining standards.
- Consider managed services that abstract protocol complexity.

Government and regulated industry protocol considerations

Government and regulated industries should consider the following actions:

- Emphasize open standards to ensure long-term access and avoid vendor lock-in.
- Prioritize protocols with strong security models and authentication mechanisms.
- Consider data sovereignty implications of remote and local deployment models.
- Document protocol decisions for compliance and governance requirements.

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

To implement the Model Context Protocol (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 can also be used to enable agents to interact with other agents. The protocol's Streamable HTTP transport implementation gives developers a comprehensive set of interaction patterns without having to reinvent the wheel. 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.

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 maximize your freedom and flexibility. The 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

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 tools** – Proprietary tools that are specific to Anthropic Claude models.

Framework-native tools

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

- **Strands Agents Python tools** – Lightweight, quick-to-implement tools that are specific to the Strands Agents framework.
- **LangChain tools** – Python-based tools that are tightly integrated with the LangChain ecosystem.
- **LlamaIndex tools** – Tools that are optimized 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 the most 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	Tools run locally but communicate over HTTP	More complex local tools with separation of concerns	Better isolation but still low latency
Remote SSE-based	Tools run on remote servers	Production environments and shared tools	Scalable and centrally managed

The official Model Context Protocol 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.

Framework-native tools

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

The Strands Agents SDK offers Python-based tools characterized by their lightweight design that requires minimal overhead for simple operations. They enable quick implementation and allow developers to create tools with just a few lines of code. In addition, they're tightly integrated to work seamlessly within the Strands Agents framework.

The following example demonstrates how to create a simple weather tool using Strands Agents. Developers can quickly transform Python functions into agent-accessible tools with minimal code overhead and automatically generate appropriate 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 better interoperability and future flexibility 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	Microsoft ecosystem
LangChain	Python classes	Large ecosystem of pre-built tools	Framework lock-in

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 sophisticated 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

MCP's resource system 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 impacts 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 tool ecosystem that remains flexible and powerful as AI technology advances.

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

1. **Adopt MCP as your foundation** – MCP provides a standardized way to connect agents to tools with strong security features. Start with MCP as your primary tool protocol for:
   - Strategic tools that will be used across multiple agent implementations.
   - Security-sensitive tools that require robust authentication and authorization.
   - Tools that need remote execution 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 leverages 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

Make use of the following robust 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 MCP security model addresses these concerns comprehensively. 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 three essential components for successful implementation: *frameworks* that provide the foundation, *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.

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 ecosystem, support for agent-to-agent communication patterns, and tool integration capabilities. AWS has embraced MCP as a strategic protocol, actively contributing to its development and implementing it across services like the Strands Agents SDK. By using MCP 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

- 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
- Introducing Strands Agents, an Open Source AI Agents SDK
- Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP
- AWS Transform for .NET, the first agentic AI service for modernizing .NET applications at scale
- AWS Weekly Roundup: Strands Agents

AWS Prescriptive Guidance

- Foundations of agentic AI on AWS
- Agentic AI patterns and workflows on AWS
- Building serverless architectures for agentic AI on AWS
- Building multi-tenant architectures for agentic AI on AWS
- Retrieval Augmented Generation options and architectures on AWS

AWS resources

- Amazon Bedrock documentation
- Amazon Nova documentation
- AWS MCP Servers (GitHub)

Other resources

- AutoGen documentation (Microsoft)

- Building effective agents (Anthropic)
- CrewAI GitHub repository
- LangChain documentation
- LangGraph platform
- 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. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	July 14, 2025

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more

information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries.

Detective guardrails detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners,

migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata.

The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your

organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.