Foundations of agentic AI on AWS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Foundations of agentic AI on AWS

Table of Contents

Foundations of agentic AI on AWS ... **1**
Intended audience ... 2
Objectives ... 2
About this content series ... 2
Introduction to software agents ... **3**
From autonomy to distributed intelligence ... 3
Early concepts of autonomy ... 3
The actor model and asynchronous execution ... 4
Distributed intelligence and multi-agent systems ... 4
Nwana's typology and the rise of software agents ... 4
Nwana's agent typology ... 5
From typology to modern agentic principles ... 6
The three pillars of modern software agents ... 6
Autonomy ... 7
Asynchronicity ... 7
Agency as the defining principle ... 8
Agency with purpose ... 8
The purpose of software agents ... **9**
From the actor model to agent cognition ... 9
The agent function: perceive, reason, act ... 9
Autonomous collaboration and intentionality ... 10
Delegating intent ... 11
Operating in dynamic, unpredictable environments ... 11
Reducing human cognitive load ... 11
Enabling distributed intelligence ... 4
Acting with purpose, not only reaction ... 12
The evolution of software agents ... **13**
Foundations of software agents ... 14
1959 – Oliver Selfridge: the birth of autonomy in software ... 14
1973 – Carl Hewitt: the actor model ... 14
Maturing the field: from reasoning to action ... 14
1977 – Victor Lesser: multi-agent systems ... 14
1990s – Michael Wooldridge and Nicholas Jennings: the agent spectrum ... 15
1996 – Hyacinth S. Nwana: formalizing the agent concept ... 15

A parallel timeline: the rise of large language models 15
Timelines converge: the emergence of agentic AI 16
2023-2024 – enterprise-grade agent platforms 16
January-June 2025 – expanded enterprise capabilities 16
Emergence – agentic AI 17
Software agents to agentic AI 18
Core building blocks of software agents 18
Perception module 19
Cognitive module 20
Action module 21
Learning module 22
Traditional agent architecture: perceive, reason, act 23
Perceive module 23
Reason module 24
Act module 24
Generative AI agents: replacing symbolic logic with LLMs 25
Key enhancements 25
Achieving long-term memory in LLM-based agents 26
Combined benefits in agentic AI 27
Comparing traditional AI to software agents and agentic AI 27
Next steps 30
Resources 31
AWS references 31
Other references 31
Document history 33
Glossary 34
..... 34
A 35
B 38
C 40
D 43
E 47
F 49
G 51
H 52
I 53

L 55
M 57
O 61
P 63
Q 66
R 66
S 69
T 73
U 74
V 75
W 75
Z 76

Foundations of agentic AI on AWS

Aaron Sempf, Amazon Web Services

July 2025 (document history)

In a world of increasingly intelligent, distributed, and autonomous systems, the concept of an *agent*—an entity that can perceive its environment, reason about its state, and act with intent—has become foundational. Agents are not merely programs that execute instructions; they are goal-oriented, context-aware entities that make decisions on behalf of users, systems, or organizations. Their emergence reflects a shift in how you build and think about software: a shift from procedural logic and reactive automation to systems that operate with autonomy and purpose.

At the intersection of AI, distributed systems, and software engineering lies a powerful paradigm known as *agentic AI*. This new generation of intelligent systems consists of software agents that are capable of adaptive behavior, complex coordination, and delegated decision-making.

This guide introduces the principles that define modern software agents and outlines their evolution toward agentic AI. To explain this shift, the guide provides the conceptual background and then traces the evolution of software agents to agentic AI:

- Introduction to software agents defines software agents, compares them to traditional software components, and introduces the essential characteristics that differentiate agentic behavior from traditional automation by drawing from established frameworks.
- The purpose of software agents examines why software agents exist, what roles they fulfill, what problems they solve, and how they enable intelligent delegation, reduce cognitive load, and support adaptive behavior in dynamic environments.
- The evolution of software agents traces the intellectual and technological milestones that shaped software agents, from early concepts of autonomy and concurrency to the emergence of multi-agent systems and formal agent architectures, resulting in the convergence with generative AI.
- Software agents to agentic AI introduces agentic AI as the culmination of decades of progress that combines distributed agent models with foundation models, serverless compute, and orchestration protocols. This section describes how this convergence enables a new generation of intelligent, tool-using agents that operate with autonomy, asynchronicity, and true agency at scale.

Intended audience

This guide is designed for architects, developers, and technology leaders who want to understand the history, main concepts, and evolution of software agents to agentic AI before they adopt this technology for modern cloud solutions on AWS.

Objectives

Adopting agentic architectures helps organizations:

- Accelerate time to value: Automate and scale knowledge work, and reduce manual effort and latency.
- Improve customer engagement: Deliver intelligent assistants across domains.
- Reduce operational costs: Automate decision flows that previously required human input or oversight.
- Drive innovation and differentiation: Build intelligent products that adapt, learn, and compete in real time.
- Modernize legacy workflows: Reframe scripts and monoliths into modular reasoning agents.

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 series includes the following guides:

- *Foundations of agentic AI on AWS* (this guide)
- Agentic AI patterns and workflows on AWS
- Agentic AI frameworks, protocols, and tools on AWS
- 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.

Introduction to software agents

The concept of software agents has evolved significantly from its foundations in autonomous entities in the 1960s to its formal exploration in the early 1990s. As digital systems grow increasingly complex—from deterministic scripts to adaptive, intelligent applications—software agents have become essential building blocks for enabling autonomous, context-aware, and goal-driven behavior in computing systems. In the context of cloud-native and AI-enhanced architectures, particularly with the advent of generative AI, large language models (LLMs), and platforms such as Amazon Bedrock, software agents are being redefined through new lenses of capability and scale.

This introduction draws from the seminal work Software Agents: An Overview by Hyacinth S. Nwana (Nwana 1996). It defines software agents, discusses their conceptual roots, and extends the discussion into a contemporary framework to define three overarching principles of modern software agents: *autonomy*, *asynchronicity*, and *agency*. These principles distinguish software agents from other types of services or applications, and enable these agents to operate with purpose, resilience, and intelligence in distributed, real-time environments.

In this section

- From autonomy to distributed intelligence
- Nwana's typology and the rise of software agents
- The three pillars of modern software agents

From autonomy to distributed intelligence

Before the term *software agent* entered the mainstream, early computing research explored the idea of *autonomous digital entities*, which are systems that are capable of acting independently, reacting to inputs, and making decisions based on internal rules or objectives. These early ideas laid the conceptual groundwork for what would become the agent paradigm. (For a historical timeline, see the section The evolution of software agents later in this guide.)

Early concepts of autonomy

The notion of machines or programs that act independently from human operators has intrigued system designers for decades. Early work in cybernetics, artificial intelligence, and control systems examined how software could exhibit self-regulating behavior, respond dynamically to changes, and operate without continuous human supervision.

These ideas introduced *autonomy* as a core attribute of intelligent systems and set the stage for the emergence of software that could *decide and act*, instead of only *reacting* or *executing*.

The actor model and asynchronous execution

In the 1970s, the *actor model*, which was introduced in the paper A Universal Modular ACTOR Formalism for Artificial Intelligence (Hewitt et al. 1973), provided a formal framework for thinking about decentralized, message-driven computation. In this model, actors are independent entities that communicate exclusively by passing asynchronous messages, and enable scalable, concurrent, and fault-tolerant systems.

The actor model emphasized three key attributes that continue to influence modern agent design:

- Isolation of state and behavior
- Asynchronous interaction between entities
- Dynamic creation and delegation of tasks

These attributes aligned with the needs of distributed systems and prefigured the operational characteristics of software agents in cloud-native environments.

Distributed intelligence and multi-agent systems

As computing systems became more interconnected after the 1960s, researchers explored distributed artificial intelligence (DAI). This field focused on how multiple autonomous entities could work collaboratively or competitively across a system. DAI led to the development of multi-agent systems, where each agent has local goals, perception, and reasoning but also operates within a broader, interconnected environment.

This vision of distributed intelligence, where decision-making is decentralized and emergent behavior arises from agent interaction, remains central to how modern agent-based systems are conceived and built.

Nwana's typology and the rise of software agents

The formalization of the software agent concept in the mid-1990s marked a turning point in the evolution of intelligent systems. Among the most influential contributions to this formalization is

Hyacinth S. Nwana's seminal paper, Software Agents: An Overview (Nwana 1996), which provided one of the first comprehensive frameworks for categorizing and understanding software agents across various dimensions.

In this paper, Nwana surveys the state of software agent research and identifies a growing divergence in how agents were being defined and implemented. The paper highlights the need for a common conceptual framework and proposes a typology that classifies agents according to their key capabilities. It reviews representative agent systems from academia and industry, distinguishes agents from traditional programs and objects, and outlines the challenges and opportunities in agent-based computing.

Nwana emphasizes that software agents are not a monolithic concept but exist along a spectrum of sophistication and capability. The typology serves to clarify this landscape and guide future design and research.

Nwana defines a software agent as a software entity that functions continuously and autonomously in a particular environment, which is often inhabited by other agents and processes. This definition emphasizes two key characteristics:

- Continuity: The agent operates persistently over time, without requiring constant human intervention.
- Autonomy: The agent has the capability to make decisions and act on them independently, based on its perception of the environment.

This definition, combined with Nwana's agent typology, emphasizes delegated authority (through autonomy) and proactivity as foundational characteristics of agents. It differentiates between agents and subroutines or services by highlighting the agent's ability to act independently on behalf of another entity and to initiate behavior in pursuit of goals, instead of only responding to direct commands.

Nwana's agent typology

To further differentiate among various types of agents, Nwana introduces a classification system based on six key attributes:

- Autonomy: The agent operates without direct intervention from humans or others.
- Social ability: The agent interacts with other agents or humans by using communication mechanisms.

- Reactivity: The agent perceives its environment and responds in a timely manner.
- Proactivity: The agent exhibits goal-directed behavior by taking the initiative.
- Adaptability and learning: The agent improves its performance over time through experience.
- Mobility: The agent can move across different system environments or networks.

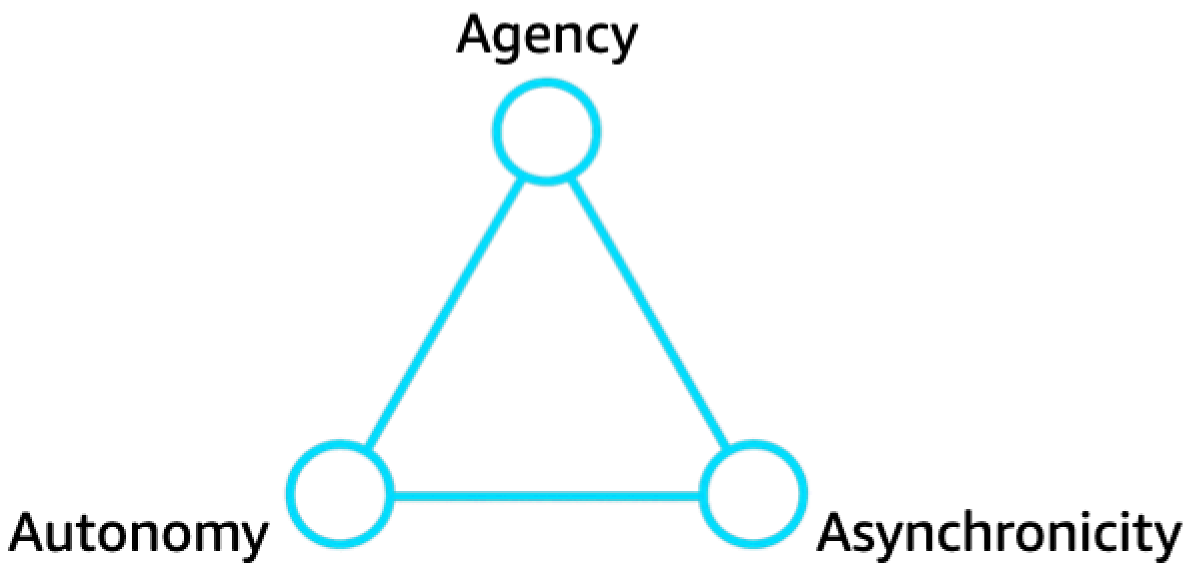
From typology to modern agentic principles

Nwana's work served as both a taxonomy and a foundational lens through which the computing community could evaluate the evolving forms of agency in software. His emphasis on autonomy, proactivity, and the concept of acting on behalf of a user or system laid the groundwork for what we now consider agentic behavior.

Although the technologies and environments have changed, especially with the rise of generative AI, serverless infrastructure, and multi-agent orchestration frameworks, the foundational insights from Nwana's work remain relevant. They provide a critical bridge between early agent theory and the three modern pillars of software agents.

The three pillars of modern software agents

In the context of today's AI-powered platforms, microservice architectures, and event-driven systems, software agents can be defined by three interdependent principles that distinguish them from standard services or automation scripts: autonomy, asynchronicity, and agency. In the following illustration and in subsequent diagrams, the triangle represents these three pillars of modern software agents.



Autonomy

Modern agents operate independently. They make decisions based on internal state and environmental context without requiring human prompts. This enables them to react to data in real-time, manage their own lifecycle, and adjust their behavior based on goals and situational inputs.

Autonomy is the foundation of agent behavior. It allows agents to function without continuous supervision or hardcoded control flows.

Asynchronicity

Agents are fundamentally asynchronous. This means that they respond to events, signals, and stimuli as they occur, without relying on blocking calls or linear workflows. This characteristic enables scalable, non-blocking communication, responsiveness in distributed environments, and loose coupling between components.

Through asynchronicity, agents can participate in real-time systems and coordinate with other services or agents fluidly and efficiently.

Agency as the defining principle

Autonomy and asynchronicity are necessary, but these features alone aren't sufficient to make a system a true software agent. The critical differentiator is agency, which introduces:

- Goal-directed behavior: Agents pursue objectives and evaluate progress toward them.
- Decision-making: Agents assess options and choose actions based on rules, models, or learned policies.
- Delegated intent: Agents act on behalf of a person, system, or organization and have an embedded sense of purpose.
- Contextual reasoning: Agents incorporate memory or models of their environment to guide behavior intelligently.

A system that is autonomous and asynchronous might still be a reactive service. What makes it a software agent is its ability to act with intention and purpose, to be *agentic*.

Agency with purpose

The principles of autonomy, asynchronicity, and agency enable systems to operate intelligently, adaptively, and independently across distributed environments. These principles are rooted in decades of conceptual and architectural evolution, and now underpin many of the most advanced AI systems being built today.

In this new era of generative AI, goal-oriented orchestration, and multi-agent collaboration, it's essential to understand what makes a software agent truly agentic. Recognizing agency as the defining characteristic helps us move beyond automation and into the realm of autonomous intelligence with purpose.

The purpose of software agents

As modern systems have become increasingly complex, distributed, and intelligent, the role of software agents has gained prominence across domains that range from autonomous operations to user-assistive technologies. But what is the underlying purpose of software agents? Why do we design systems that go beyond scripts, services, or static models, and instead delegate tasks to entities that are capable of perceiving, reasoning, and acting?

This section explores the fundamental purpose of software agents: to enable intelligent delegation of tasks within dynamic environments, with a focus on autonomy, adaptability, and purposeful action. It introduces the conceptual foundation of software agents, traces their cognitive structure, and outlines the real-world problems that they are uniquely equipped to solve.

In this section

- From the actor model to agent cognition
- The agent function: perceive, reason, act
- Autonomous collaboration and intentionality

From the actor model to agent cognition

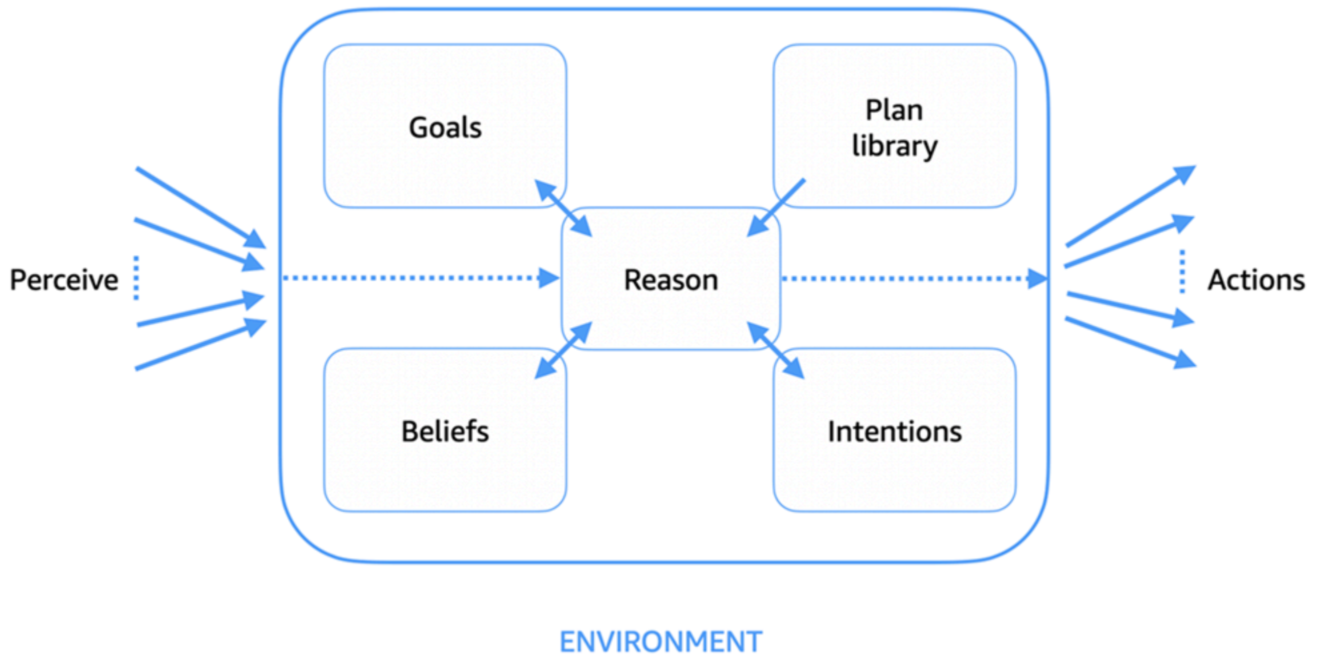
The purpose and structure of software agents are grounded in ideas that emerged from early computation models, particularly the actor model that was introduced by Carl Hewitt in the 1970s (Hewitt et al. 1973).

The actor model treats computation as a collection of independent, concurrently executing entities called *actors*. Each actor encapsulates its own state, interacts solely through asynchronous message passing, and can create new actors and delegate tasks.

This model provided the conceptual foundation for decentralized reasoning, reactivity, and isolation—all of which underpin the behavioral architecture of modern software agents.

The agent function: perceive, reason, act

At the core of every software agent is a cognitive cycle that is often described as the *perceive, reason, act* loop. This process is illustrated in the following diagram. It defines how agents operate autonomously in dynamic environments.



- **Perceive**: Agents gather information (for example, events, sensor inputs, or API signals) from the environment and update their internal state or beliefs.
- **Reason**: Agents analyze current beliefs, goals, and contextual knowledge by using a plan library or logic system. This process might involve goal prioritization, conflict resolution, or intention selection.
- **Act**: Agents select and execute actions that move them closer to achieving their delegated goals.

This architecture supports the ability of agents to function beyond rigid programming and enables flexible, context-sensitive, and goal-directed behavior. It forms the mental framework that guides the broader purposes of software agents.

Autonomous collaboration and intentionality

The purpose of software agents is to bring autonomy, context-awareness, and intelligent delegation to modern computing. Because agents are built on the principles of the actor model and embodied in the perceive, reason, act cycle, they enable systems that are not only reactive, but proactive and purposeful.

Agents empower software to decide, adapt, and act in complex environments. They represent users, interpret goals, and implement tasks at machine speed. As we move deeper into the era of agentic AI, software agents are becoming the operational interface between human intent and intelligent digital action.

Delegating intent

Unlike traditional software components, software agents exist to act on behalf of something else: a user, another system, or a higher-level service. They carry *delegated intent*, which means that they:

- Operate independently after initiation.
- Make choices that are aligned with the goals of the delegator.
- Navigate uncertainty and trade-offs in execution.

Agents bridge the gap between *instructions* and *outcomes*, which allows users to express intent at a higher level of abstraction instead of requiring explicit instructions.

Operating in dynamic, unpredictable environments

Software agents are designed for environments where conditions change constantly, data arrives in real time, and control and context are distributed.

Unlike static programs that require exact inputs or synchronous execution, agents adapt to their surroundings and respond dynamically. This is a vital capability in cloud-native infrastructure, edge computing, Internet of Things (IoT) networks, and real-time decision-making systems.

Reducing human cognitive load

One of the primary purposes of software agents is to reduce the cognitive and operational burden on humans. Agents can:

- Continuously monitor systems and workflows.
- Detect and respond to predefined or emergent conditions.
- Automate repetitive, high-volume decisions.
- React to environmental changes with minimal latency.

When decision-making shifts from users to agents, systems become more responsive, resilient, and human-centric, and can adapt in real time to new information or disruptions. This enables faster reaction turnaround as well as greater operational continuity in high-complexity or high-scale environments. The result is a shift in human focus, from micro-level decision-making to strategic oversight and creative problem-solving.

Enabling distributed intelligence

The ability of software agents to operate individually or collectively enables the design of multi-agent systems (MAS) that coordinate across environments or organizations. These systems can distribute tasks intelligently and negotiate, cooperate, or compete toward composite goals.

For example, in a global supply chain system, individual agents manage factories, shipping, warehouses, and last-mile delivery. Each agent operates with local autonomy: Factory agents optimize production based on resource constraints, warehouse agents adjust inventory flows in real time, and delivery agents reroute shipments based on traffic and customer availability.

These agents communicate and coordinate dynamically, and adapt to disruptions such as port delays or truck failures without centralized control. The system's overall intelligence emerges from these interactions and enables resilient, optimized logistics that are beyond the capabilities of a single component.

In this model, agents act as nodes in a broader intelligence fabric. They form emergent systems that are capable of solving problems that no single component could handle alone.

Acting with purpose, not only reaction

Automation alone is insufficient in complex systems. The defining purpose of a software agent is to act with purpose and to evaluate goals, weigh context, and make informed choices. This means that software agents pursue goals instead of only responding to triggers. They can revise beliefs and intentions based on experience or feedback. In this context, beliefs refer to the agent's internal representation of the environment (for example, "package X is in warehouse A"), based on its perceptions (input and sensors). Intentions refer to the plans that the agent chooses to achieve a goal (for example, "use delivery route B and notify the recipient"). Agents can also escalate, defer, or adapt actions as necessary.

This intentionality is what makes software agents not just reactive executors, but autonomous collaborators in intelligent systems.

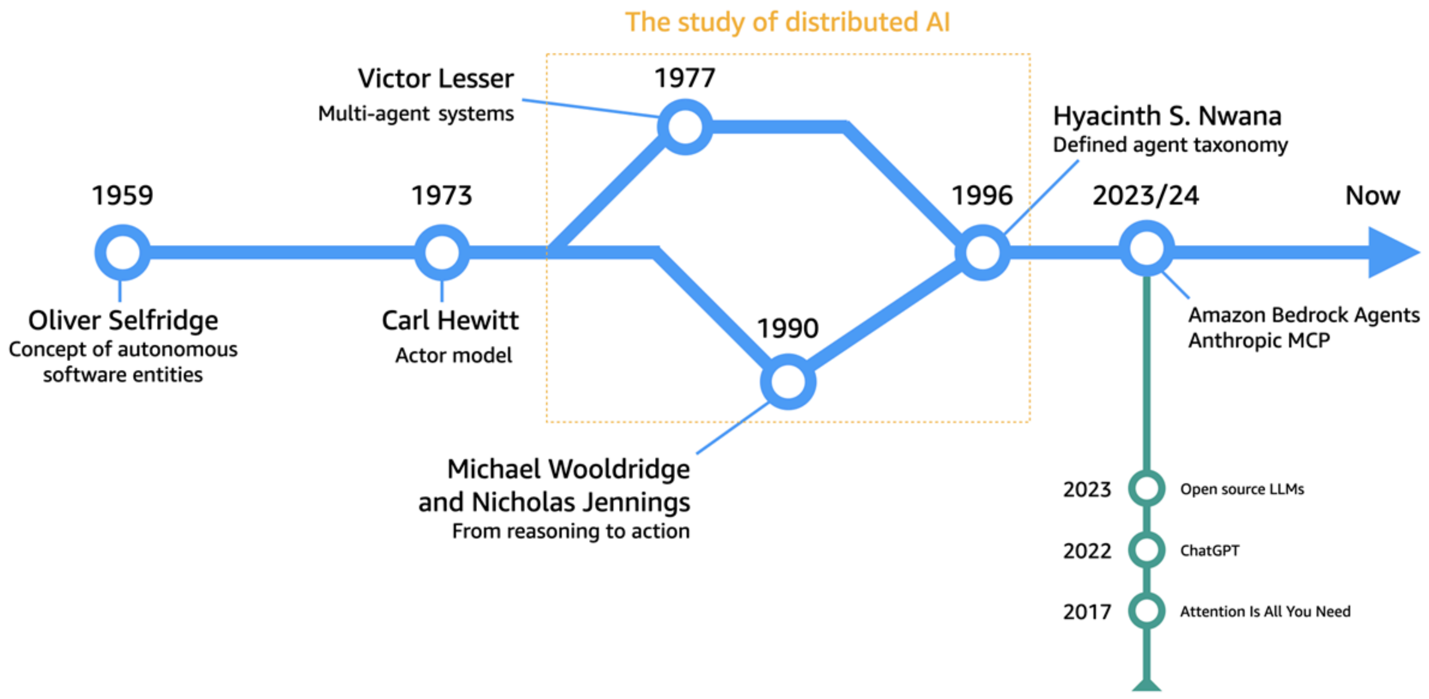
The evolution of software agents

The journey from simple automated systems to intelligent, autonomous, and goal-directed software agents reflects decades of evolution in computer science, artificial intelligence, and distributed systems.

This evolution was followed by the rise of machine learning, which shifted the paradigm from handcrafted rules to statistical pattern recognition. These systems could learn from data and enabled advances in perception, classification, and decision-making.

Large language models (LLMs) represent a convergence of scale, architecture, and unsupervised learning. LLMs can reason, generate, and adapt tasks with little or no task-specific training. By combining LLMs with scalable cloud-native infrastructure and composable architectures, we are now achieving the full vision of agentic AI: intelligent software agents that can operate with autonomy, context-awareness, and adaptability at enterprise scale.

This section explores the history of software agents from foundational theory to modern practice, as illustrated in the following diagram. It highlights the convergence of distributed artificial intelligence (DAI) and transformer-based generative AI, and identifies the key milestones that have shaped the emergence of agentic AI.



In this section

- Foundations of software agents
- Maturing the field: from reasoning to action
- A parallel timeline: the rise of large language models
- Timelines converge: the emergence of agentic AI

Foundations of software agents

1959 – Oliver Selfridge: the birth of autonomy in software

The roots of software agents trace back to Oliver Selfridge, who introduced the concept of *autonomous software entities (demons)*—programs that are capable of perceiving their environment and acting independently (Selfridge 1959). His early work in machine perception and learning laid the philosophical groundwork for future notions of agents as independent, intelligent systems.

1973 – Carl Hewitt: the actor model

A pivotal advancement came with Carl Hewitt's actor model (Hewitt et al. 1973), which is a formal computational model that describes agents as independent, concurrent entities. In this model, agents can encapsulate their own state and behavior, communicate by using asynchronous message passing, and dynamically create other actors and delegate tasks to them.

The actor model provided both the theoretical foundation and the architectural paradigm for distributed, agent-based systems. This model prefigured modern concurrency implementations such as the Erlang programming language and the Akka framework.

Maturing the field: from reasoning to action

1977 – Victor Lesser: multi-agent systems

In the late 1970s, distributed artificial intelligence (DAI) emerged. It was championed by Victor Lesser, who is widely recognized for pioneering multi-agent systems (MAS). His work focused on how independent software entities could cooperate, coordinate, and negotiate (see the Resources section). This development led to systems that were capable of solving complex problems collectively—an essential leap in building distributed intelligence.

1990s – Michael Wooldridge and Nicholas Jennings: the agent spectrum

By the 1990s, the distributed intelligence field had matured with contributions from researchers such as Michael Wooldridge and Nicholas Jennings. These scholars categorized agents along a spectrum, from reactive to deliberative, from non-cognitive systems to goal-driven, reasoning agents (Wooldridge and Jennings 1995). Their work emphasized that agents were no longer abstract ideas but were being applied across a wide range of practical domains, from robotics to enterprise software.

These researchers also introduced a shift in focus: from centralized reasoning to distributed action. Agents were no longer just thinkers—they were doers that operated in real-time environments with autonomy and purpose.

1996 – Hyacinth S. Nwana: formalizing the agent concept

In 1996, Hyacinth S. Nwana published the influential paper Software Agents: An Overview, which provided the most comprehensive classification of agents to date. His typology included attributes such as autonomy, social ability, reactivity, proactivity, learning, and mobility, and differentiated between software agents and traditional software constructs.

Nwana also offered a now widely accepted definition, paraphrased: *A software agent is a software-based computer program that acts for a user or other program in a relationship of agency, which derives from the notion of delegation.*

This formalization was instrumental in transitioning software agents from theoretical constructs to real-world applications. It gave rise to a generation of agent-based systems across fields such as telecommunications, workflow automation, and intelligent assistants.

Nwana's work sits at the convergence point of early distributed AI research and the operational architectures of modern agents. It is a crucial bridge between the cognitive theory of agents and their practical deployment in today's systems.

A parallel timeline: the rise of large language models

While agent frameworks evolved, a parallel and convergent revolution was happening in natural language processing and machine learning:

- **2017 – transformers**: The paper Attention Is All You Need (Vaswani et al. 2017) introduced the transformer architecture, which dramatically improved how machines process and generate language.

- **2022 – ChatGPT**: OpenAI released a chat-based interface to GPT-3.5 called ChatGPT, which enabled natural, interactive conversation with a general-purpose AI system.
- **2023 – open source LLMs**: The releases of Llama, Falcon, and Mistral made powerful models widely accessible and accelerated the development of agent frameworks in open source and enterprise environments.

These innovations turned language models into reasoning engines that are capable of parsing context, planning actions, and chaining responses, and LLMs became key enablers of intelligent software agents.

Timelines converge: the emergence of agentic AI

2023-2024 – enterprise-grade agent platforms

The convergence of distributed software agent architectures and transformer-based LLMs culminated in the rise of agentic AI.

- Amazon Bedrock Agents introduced a fully managed way to build goal-driven, tool-using software agents by using foundation models from Amazon Bedrock.
- The Model Concept Protocol (MCP) from Anthropic defined a method for large language models to access, and interact with, external tools, environments, and memory. This is key for contextual, persistent, and autonomous behavior.

These two milestones represent the synthesis of agency and intelligence. Agents were no longer limited to static workflows or rigid automation. They could now reason across multiple steps, coordinate with tools and APIs, maintain contextual state, and learn and adapt over time.

January-June 2025 – expanded enterprise capabilities

In the first half of 2025, the agentic AI landscape expanded significantly with new enterprise capabilities. In February 2025, Anthropic released Claude 3.7 Sonnet, which was the first hybrid reasoning model on the market, and the MCP specification gained widespread adoption.

AI coding assistants such as Amazon Q Developer, Cursor, and WindSurf integrated MCP to standardize code generation, repository analysis, and development workflows. The MCP March 2025 release introduced significant enterprise-ready features, including OAuth 2.1 security integration, expanded resource types for diverse data access, and enhanced connectivity options through Streamable HTTP. Building on this foundation, AWS announced in May 2025 that it was joining the MCP steering committee and contributing to new agent-to-agent communication capabilities. This further strengthens the protocol's position as an industry standard for agentic AI interoperability.

In May 2025, AWS strengthened customer options for building agentic AI workflows by open sourcing the Strands Agents framework. This provider-independent and model-agnostic framework enables developers to use foundation models across platforms while maintaining deep AWS service integration. As highlighted in the AWS Open Source Blog, Strands Agents follows a model-first design philosophy that places foundation models at the core of agent intelligence. This makes it easier for customers to build and deploy sophisticated AI agents for their specific use cases.

Emergence – agentic AI

The evolution of software agents, from early ideas of autonomy to modern, LLM-enabled orchestration, has been long and layered. What began with Oliver Selfridge's vision of perceiving programs has grown into a robust ecosystem of intelligent, context-aware, goal-driven software agents that can collaborate, adapt, and reason.

The convergence of distributed artificial intelligence (DAI) and transformer-based generative AI marks the beginning of a new era in which software agents are no longer only tools, but autonomous actors in intelligent systems.

Agentic AI represents the next evolution in software systems. It provides a class of intelligent agents that are autonomous, asynchronous, and agentic, and can act with delegated intent and operate purposefully within dynamic, distributed environments. Agentic AI unifies the following:

- The architectural lineage of multi-agent systems and the actor model
- The cognitive model of perceive, reason, act
- The generative power of LLMs and transformers
- The operational flexibility of cloud-native and serverless computing

Software agents to agentic AI

Software agents are autonomous digital entities that are designed to perceive their environment, reason about their goals, and act accordingly. Unlike traditional software programs that follow fixed logic, agents adapt their behavior based on contextual inputs and decision frameworks. This makes them ideal for dynamic, distributed environments such as cloud-native systems, robotics, intelligent automation, and now, generative AI orchestration.

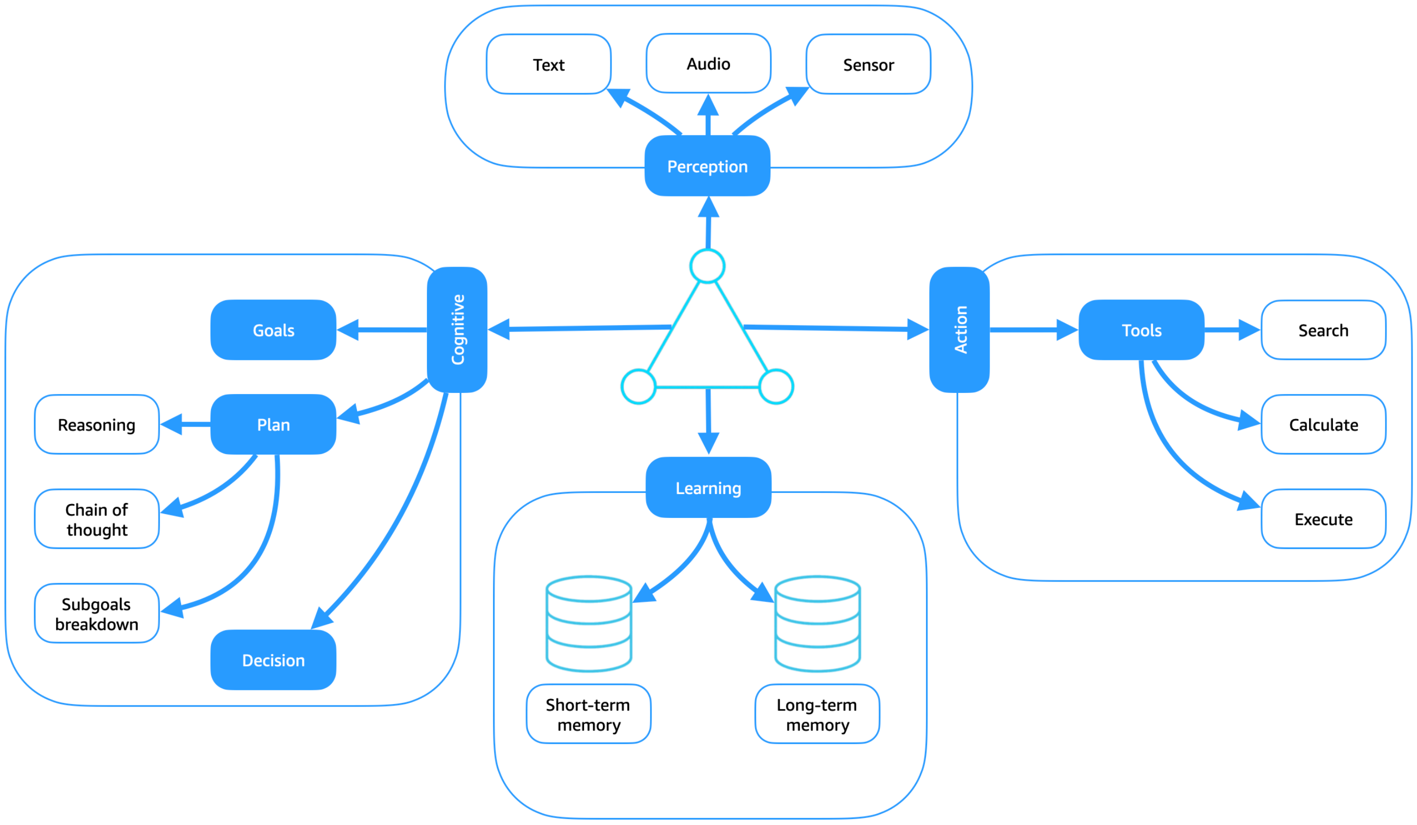
This section introduces the core building blocks of software agents and explains how these components interact within traditional architectures based on the perceive, reason, act model. It discusses how generative AI, particularly large language models (LLMs), has transformed the way software agents reason and plan. This marks a fundamental shift from rule-based systems to the data-driven, learned intelligence of agentic AI.

In this section

- Core building blocks of software agents
- Traditional agent architecture: perceive, reason, act
- Generative AI agents: replacing symbolic logic with LLMs
- Comparing traditional AI to software agents and agentic AI

Core building blocks of software agents

The following diagram presents the key functional modules found in most intelligent agents. Each component contributes to the agent's ability to operate autonomously in complex environments.



In the context of the perceive, reason, act loop, an agent's reasoning capability is distributed across both its cognitive and learning modules. Through the integration of memory and learning, the agent develops adaptive reasoning grounded in past experience. As the agent acts within its environment, it creates an emergent feedback loop: Each action influences future perceptions, and the resulting experience is incorporated into memory and internal models through the learning module. This continuous loop of perception, reasoning, and action enables the agent to improve over time and completes the full perceive, reason, act cycle.

Perception module

The perception module enables the agent to interface with its environment through diverse input modalities such as text, audio, and sensors. These inputs form the raw data that all reasoning and action are based on. Text inputs might include natural language prompts, structured commands, or documents. Audio inputs encompass spoken instructions or environmental sounds. Sensor inputs include physical data such as visual feeds, motion signals, or GPS coordinates. The core function of perception is to extract meaningful features and representations from this raw data. This allows the agent to construct an accurate and actionable understanding of its current context. The process might involve feature extraction, object or event recognition, and semantic interpretation, and forms the critical first step in the perceive, reason, act loop. Effective perception ensures that downstream reasoning and decision-making are grounded in relevant, up-to-date situational awareness.

Cognitive module

The cognitive module serves as the deliberative core of the software agent. It is responsible for interpreting perceptions, forming intent, and guiding purposeful behavior through goal-driven planning and decision-making. This module transforms inputs into structured reasoning processes, which enables the agent to operate intentionally rather than reactively. These processes are managed through three key submodules: goals, planning, and decision-making.

Goals submodule

The goals submodule defines the agent's intent and direction. Goals can be explicit (for example, "navigate to a location" or "submit a report") or implicit (for example, "maximize user engagement" or "minimize latency"). They are central to the agent's reasoning cycle, and provide a target state for its planning and decisions.

The agent continuously evaluates progress toward its goals and might reprioritize or regenerate goals based on new perceptions or learning. This goal awareness keeps the agent adaptable in dynamic environments.

Planning submodule

The planning submodule constructs strategies to achieve the agent's current goals. It generates action sequences, decomposes tasks hierarchically, and selects from predefined or dynamically generated plans.

To operate effectively in non-deterministic or changing environments, planning is not static. Modern agents can generate chain-of-thought sequences, introduce subgoals as intermediate steps, and revise plans in real time when conditions shift.

This submodule connects closely with memory and learning, and allows the agent to refine its planning over time based on past outcomes.

Decision-making submodule

The decision-making submodule evaluates available plans and actions to select the most appropriate next step. It integrates input from perception, the current plan, the agent's goals, and environmental context.

Decision-making accounts for:

- Trade-offs between conflicting goals
- Confidence thresholds (for example, uncertainty in perception)
- Consequences of actions
- The agent's learned experience

Depending on the architecture, agents might rely on symbolic reasoning, heuristics, reinforcement learning, or language models (LLMs) to make informed decisions. This process ensures keeps the agent's behavior context-aware, goal-aligned, and adaptive.

Action module

The action module is responsible for executing the agent's selected decisions and interfacing with the external world or internal systems to produce meaningful effects. It represents the Act phase of the perceive, reason, act loop, where intent is transformed into behavior.

When the cognitive module selects an action, the action module coordinates execution through specialized submodules, where each submodule aligns with the agent's integrated environment:

- Physical actuation: For agents that are embedded in robotic systems or IoT devices, this submodule translates decisions into real-world physical movements or hardware-level instructions.

  Examples: steering a robot, triggering a valve, turning on a sensor.
- Integrated interaction: This submodule handles non-physical but externally visible actions such as interacting with software systems, platforms, or APIs.

  Examples: sending a command to a cloud service, updating a database, submitting a report by calling an API.
- Tool invocation: Agents often extend their capabilities by using specialized tools to accomplish sub-tasks such as the following:
  - Search: querying structured or unstructured knowledge sources
  - Summarization: compressing large text inputs into high-level overviews
  - Calculation: performing logical, numerical, or symbolic computation

  Tool invocation enables complex behavior composition through modular, callable skills.

Learning module

The learning module enables agents to adapt, generalize, and improve over time based on experience. It supports the reasoning process by continuously refining the agent's internal models, strategies, and decision policies by using feedback from perception and action.

This module operates in coordination with both short-term and long-term memory:

- Short-term memory: Stores transient context, such as dialogue state, current task information, and recent observations. It helps the agent maintain continuity within interactions and tasks.
- Long-term memory: Encodes persistent knowledge from past experiences, including previously encountered goals, outcomes of actions, and environmental states. Long-term memory enables the agent to recognize patterns, reuse strategies, and avoid repeating mistakes.

Learning modes

The learning module supports a range of paradigms, such as supervised, unsupervised, and reinforcement learning, which support different environments and agent roles:

- Supervised learning: Updates internal models based on labeled examples, often from human feedback or training datasets.

  Example: learning to classify user intent based on previous conversations.
- Unsupervised learning: Identifies hidden patterns or structures in data without explicit labels.

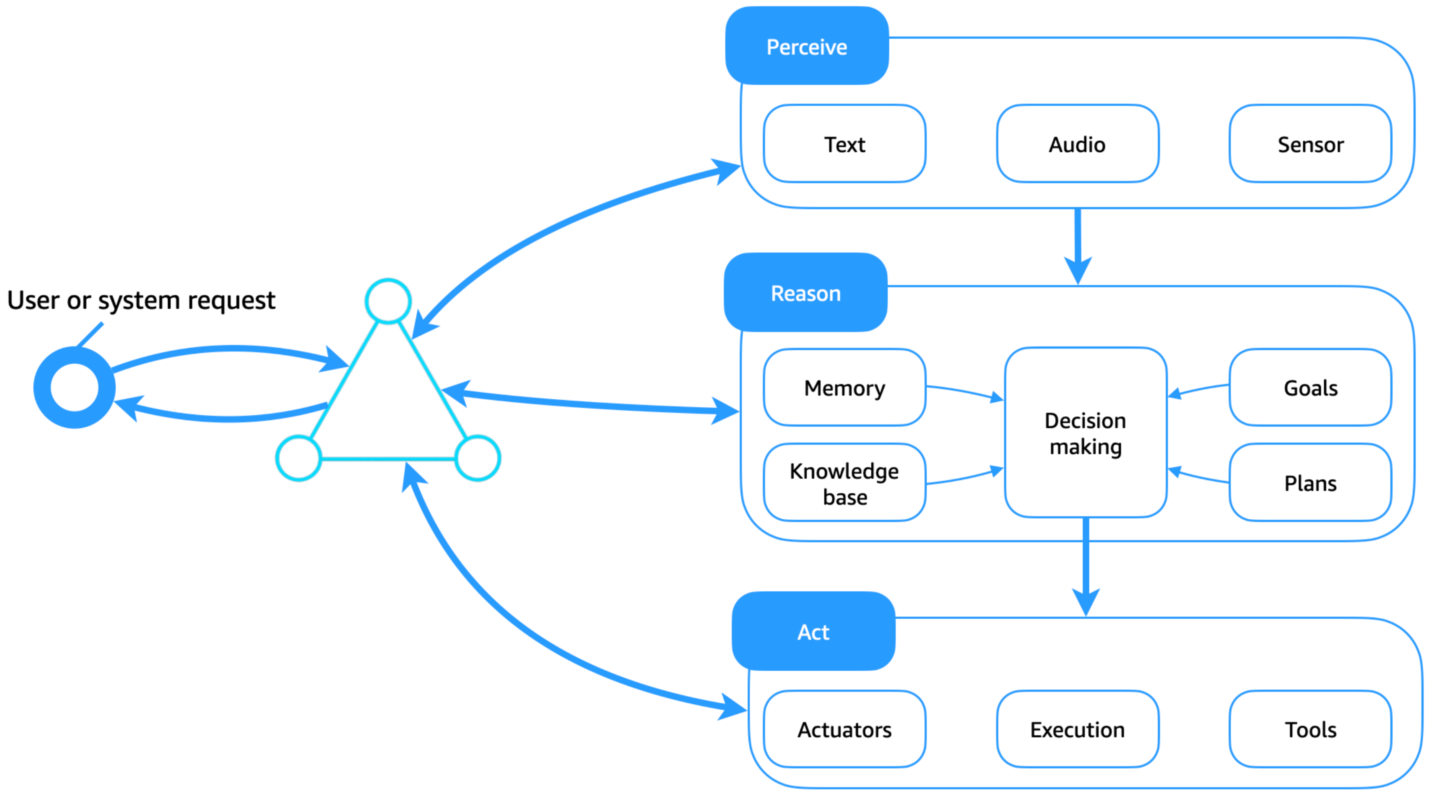
  Example: clustering environmental signals to detect anomalies.
- Reinforcement learning: Optimizes behavior through trial and error by maximizing cumulative reward in interactive environments.

  Example: learning which strategy leads to the fastest task completion.

Learning integrates tightly with the agent's cognitive module. It refines planning strategies based on past outcomes, enhances decision-making through the evaluation of historical success, and continuously improves the mapping between perception and action. Through this closed learning and feedback loop, agents evolve beyond reactive execution to become self-improving systems that are capable of adapting to new goals, conditions, and contexts over time.

Traditional agent architecture: perceive, reason, act

The following diagram illustrates how the building blocks discussed in the previous section operate under the perceive, reason, act cycle.



Perceive module

The perceive module acts as the agent's sensory interface with the external world. It transforms raw environmental input into structured representations that inform reasoning. This includes handling multimodal data such as text, audio, or sensor signals.

- Text input may come from user commands, documents, or dialogue.
- Audio input includes spoken instructions or environmental sounds.
- Sensor input captures real-world signals such as motion, visual feeds, or GPS.

When the raw input has been ingested, the perception process performs feature extraction, followed by object or event recognition and semantic interpretation to create a meaningful model of the current situation. These outputs provide structured context for downstream decision-making and anchor the agent's reasoning in real-world observations.

Reason module

The reason module is the cognitive core of the agent. It evaluates context, formulates intent, and determines appropriate actions. This module orchestrates goal-driven behavior by using both learned knowledge and reasoning.

The reason module consists of tightly integrated submodules:

- Memory: Maintains dialogue state, task context, and episodic history in both short-term and long-term formats.
- Knowledge base: Provides access to symbolic rules, ontologies, or learned models (such as embeddings, facts, and policies).
- Goals and plans: Defines desired outcomes and constructs action strategies to achieve them. Goals can be dynamically updated and plans can be adaptively modified based on feedback.
- Decision-making: Acts as the central arbitration engine by weighing options, evaluating trade-offs, and selecting the next action. This submodule factors in confidence thresholds, goal alignment, and contextual constraints.

Together, these components allow the agent to reason about its environment, update beliefs, select paths, and behave in a coherent, adaptive manner. The reason module closes the gap between perception and behavior.

Act module

The act module executes the agent's selected decision by interfacing with either the digital or the physical environment to carry out tasks. This is where intention becomes action.

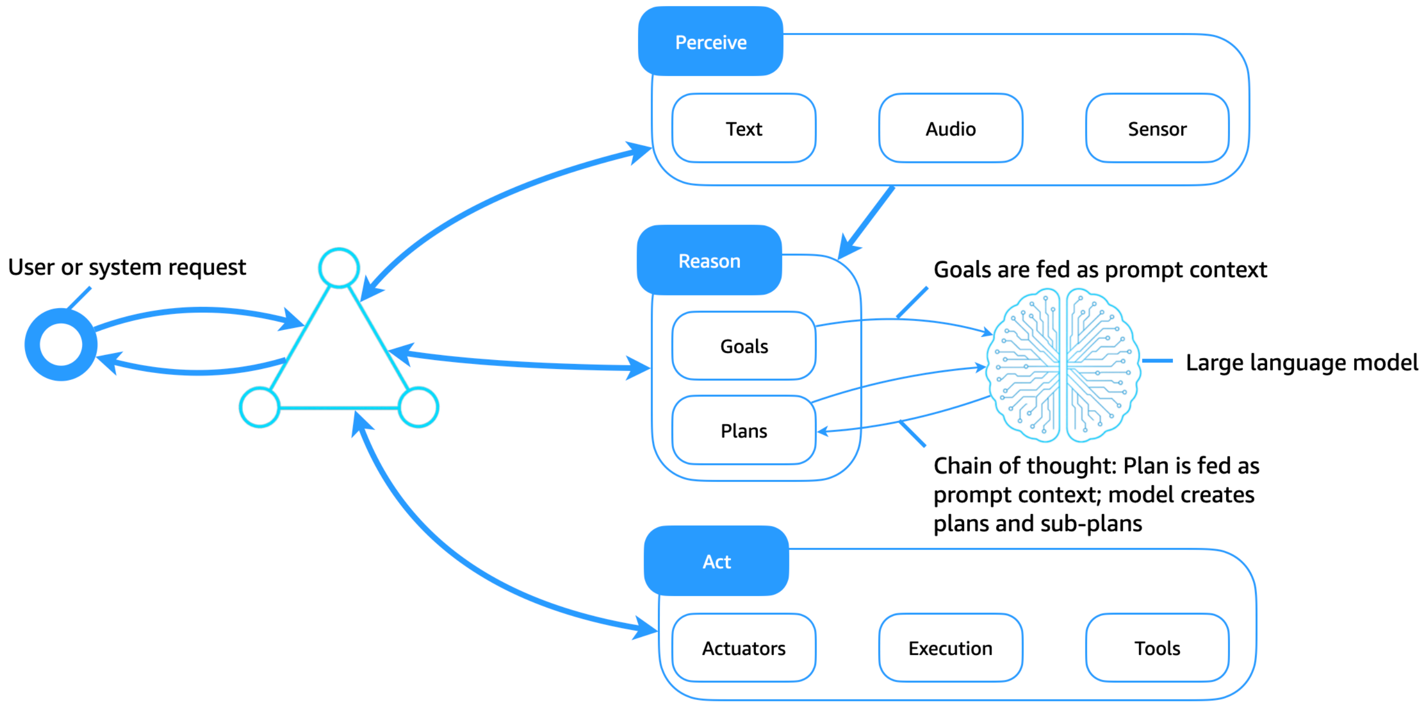
This module includes three functional channels:

- Actuators: For agents that have a physical presence (such as robots and IoT devices), controls hardware-level interactions such as movement, manipulation, or signaling.
- Execution: Handles software-based actions, including invoking APIs, dispatching commands, and updating systems.
- Tools: Enables functional capabilities such as search, summarization, code execution, calculation, and document handling. These tools are often dynamic and context-aware, which extends the agent's utility.

The outputs of the act module feed back into the environment and close the loop. These outcomes are perceived by the agent again. They update the agent's internal state and inform future decisions, thus completing the perceive, reason, act cycle.

Generative AI agents: replacing symbolic logic with LLMs

The following diagram illustrates how large language models (LLMs) now serve as a flexible and intelligent cognitive core for software agents. In contrast to traditional symbolic logic systems, which rely on static plan libraries and hand-coded rules, LLMs enable adaptive reasoning, contextual planning, and dynamic tool use, which transform how agents perceive, reason, and act.


Perceive
Text
Audio
Sensor
User or system request
Reason
Goals
Plans
Goals are fed as prompt context
Large language model
Chain of thought: Plan is fed as prompt context; model creates plans and sub-plans
Act
Actuators
Execution
Tools

Key enhancements

This architecture enhances the traditional agent architecture as follows:

- LLMs as cognitive engines: Goals, plans, and queries are passed into the model as prompt context. The LLM generates reasoning paths (such as chain of thought), decomposes tasks into sub-goals, and decides on next actions.
- Tool use through prompting: LLMs can be directed through tool use agents or reasoning and acting (ReAct) prompting to call APIs and to search, query, calculate, and interpret outputs.
- Context-aware planning: Agents generate or revise plans dynamically based on the agent's current goal, input environment, and feedback, without requiring hardcoded plan libraries.

- Prompt context as memory: Instead of using symbolic knowledge bases, agents encode memory, plans, and goals as prompt tokens that are passed to the model.
- Learning through few-shot, in-context learning: LLMs adapt behaviors through prompt engineering, which reduces the need for explicit retraining or rigid plan libraries.

Achieving long-term memory in LLM-based agents

Unlike traditional agents, which stored long-term memory in structured knowledge bases, generative AI agents must work within the context window limitations of LLMs. To extend memory and support persistent intelligence, generative AI agents use several complementary techniques: agent store, Retrieval-Augmented Generation (RAG), in-context learning and prompt chaining, and pretraining.

Agent store: external long-term memory

Agent state, user history, decisions, and outcomes are stored in a long-term agent memory store (such as a vector database, object store, or document store). Relevant memories are retrieved on demand and injected into the LLM prompt context at runtime. This creates a persistent memory loop, where the agent retains continuity across sessions, tasks, or interactions.

RAG

RAG enhances LLM performance by combining retrieved knowledge with generative capabilities. When a goal or query is issued, the agent searches a retrieval index (for example, through a semantic search of documents, earlier conversations, or structured knowledge). The retrieved results are appended to the LLM prompt, which grounds the generation in external facts or personalized context. This method extends the agent's effective memory and improves reliability and factual correctness.

In-context learning and prompt chaining

Agents maintain short-term memory by using in-session token context and structured prompt chaining. Contextual elements, such as the current plan, previous action outcomes, and agent status, are passed between calls to guide behavior.

Continued pretraining and fine-tuning

For domain-specific agents, LLMs can be continued pretrained on custom collections such as logs, enterprise data, or product documentation. Alternatively, instruction fine-tuning or reinforcement learning from human feedback (RLHF) can embed agent-like behavior directly into the model. This shifts reasoning patterns from prompt-time logic into the model's internal representation, reduces prompt length, and improves efficiency.

Combined benefits in agentic AI

These techniques, when they're used together, enable generative AI agents to:

- Maintain contextual awareness over time.
- Adapt behavior based on user history or preferences.
- Make decisions by using up-to-date, factual, or private knowledge.
- Scale to enterprise use cases with persistent, compliant, and explainable behaviors.

By augmenting LLMs with external memory, retrieval layers, and continued training, agents can achieve a level of cognitive continuity and purpose that couldn't be achieved previously through symbolic systems alone.

Comparing traditional AI to software agents and agentic AI

The following table provides a detailed comparison of traditional AI, software agents, and agentic AI.

Characteristic	Traditional AI	Software agents	Agentic AI
Examples	Spam filters, image classifiers, recommendation engines	Chatbots, task schedulers, monitorin g agents	AI assistants, autonomous developer agents, multi-agent LLM orchestrations
Execution model	Batch or synchronous	Event-driven or scheduled	Asynchronous, event-driven, and goal-driv en
Autonomy	Limited; often requires human or external orchestra tion	Medium; operates independently within predefined bounds	High; acts independe ntly with adaptive strategies

Characteristic	Traditional AI	Software agents	Agentic AI
Reactivity	Reactive to input data	Reactive to environment and events	Reactive and proactive; anticipates and initiates actions
Proactivity	Rare	Present in some systems	Core attribute; drives goal-directed behavior
Communication	Minimal; usually standalone or API-bound	Inter-agent or agent-human messaging	Rich multi-agent and human-in-the-loop interaction
Decision-making	Model inference only (classification, prediction, and so on)	Symbolic reasoning , or rule-based or scripted decisions	Contextual, goal-base d, dynamic reasoning (often LLM-enhan ced)
Delegated intent	No; performs tasks defined directly by user	Partial; acts on behalf of users or systems that have limited scope	Yes; acts with delegated goals, often across services, users, or systems
Learning and adaptation	Often model-centric (for example., ML training)	Sometimes adaptive	Embedded learning, memory, or reasoning (for example, feedback, self-corr ection)
Agency	None; tools for humans	Implicit or basic	Explicit; operates with purpose, goals, and self-direction
Context awareness	Low; stateless or snapshot-based	Moderate; some state tracking	High; uses memory, situational context, and environment models

Characteristic	Traditional AI	Software agents	Agentic AI
Infrastructure role	Embedded in apps or analytics pipelines	Middleware or service layer component	Composable agent mesh integrated with cloud, serverless, or edge systems

In summary:

- Traditional AI is tool-centric and functionally narrow. It focuses on prediction or classification.
- Traditional software agents introduce autonomy and basic communication, but they are often rule-bound or static.
- Agentic AI brings together autonomy, asynchrony, and agency. It enables intelligent, goal-driven entities that can reason, act, and adapt within complex systems. This makes agentic AI ideal for the cloud-native, AI-driven future.

Next steps

This guide discussed the history and foundations of agentic AI, which represents the evolution of traditional software agents into autonomous, intelligent systems that are powered by generative AI. It described how early software agents followed predefined rules and logic to automate tasks within fixed boundaries, and explained how agentic AI builds on this foundation by incorporating large language models, which enable agents to reason, learn, and adapt dynamically in open-ended environments.

You can explore agentic AI in depth by reviewing the following guides in this series:

- Agentic AI patterns and workflows on AWS discusses the foundational blueprints and modular constructs used to design, compose, and orchestrate goal-oriented AI agents.
- Agentic AI frameworks, protocols, and tools on AWS covers the software foundations, toolkits, and protocols to consider when you build your agentic AI solutions.
- Building serverless architectures for agentic AI on AWS discusses serverless architectures as a natural foundation of modern AI workloads and describes how you can build AI-native serverless architectures in the AWS Cloud.
- Building multi-tenant architectures for agentic AI on AWS describes the use of AI agents in multi-tenant settings, including hosting considerations, deployment models, and control planes.

Resources

For more information about the concepts discussed in this guide, see the following guides and articles.

AWS references

- Amazon Bedrock Agents
- Amazon Q Developer
- Strands Agents SDK

Other references

- Hewitt, Carl, Peter Bishop, and Richard Steiger. "A Universal Modular ACTOR Formalism for Artificial Intelligence." *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (1973): 235-245. https://www.ijcai.org/Proceedings/73/Papers/027B.pdf
- Lesser, Victor R., relevant publications (see full list):
  - Lesser, Victor R. and Daniel D. Corkill. "Functionally Accurate, Cooperative Distributed Systems." *IEEE Transactions on Systems, Man, and Cybernetics* 11, no. 1 (1981): 81-96. https://ieeexplore.ieee.org/abstract/document/4308581
  - Decker, Keith S. and Victor R. Lesser. "Communication in the Service of Coordination." *AAAI Workshop on Planning for Interagent Communication* (1994). https://www.researchgate.net/profile/Victor-Lesser/publication/2768884_Communication_in_the_Service_of_Coordination/links/00b7d51cc2a0750cb4000000/Communication-in-the-Service-of-Coordination.pdf
  - Durfee, Edmund H., Victor R. Lesser, and Daniel D. Corkill. "Trends in Cooperative Distributed Problem Solving." *IEEE Transactions on knowledge and data Engineering* (1989). http://mas.cs.umass.edu/Documents/ieee-tkde89.pdf
  - Durfee, Edmund H., V.R. Lesser, and D.D. Corkill, "Distributed Artificial Intelligence." *Cooperation Through Communication in a Distributed Problem Solving-Network* (1987): 29-58. https://www.academia.edu/download/79885643/durf94_1.pdf
  - Lâasri, Brigitte, Hassan Lâasri, Susan Lander, and Victor Lesser. "A Generic Model for Intelligent Negotiating Agents." *International Journal of Cooperative Information Systems* 01, no. 02 (1992): 291-317. https://doi.org/10.1142/S0218215792000210

  - Lander, Susan E. and Victor R. Lesser. “Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents.” *IJCAI'93: Proceedings of the 13th international joint conference on Artifical intelligence* (1993): 438-444. https://www.ijcai.org/Proceedings/93-1/Papers/062.pdf
  - Lander, Susan, Victor R. Lesser, and Margaret E. Connell. “Conflict Resolution Strategies for Cooperating Expert Agents” *CKBS'90: Proceedings of the International Working Conference on Cooperating Knowledge Based Systems* (October 1990): 183-200. https://doi.org/10.1007/978-1-4471-1831-2_10
  - Prasad, M.V. Nagendra, Victor Lesser, and Susan E. Lander. “Learning Experiments in a Heterogeneous Multi-agent System.” *IJCAI-95 Workshop on Adaptation and Learning in Multi-agent Systems* (1995): 59-64. https://www.researchgate.net/publication/2784280_Learning_Experiments_in_a_Heterogeneous_Multi-agent_System
- Nwana, Hyacinth S. "Software Agents: An Overview." *Knowledge Engineering Review* 11, no. 3 (October/November 1996): 205-244. https://teaching.shu.ac.uk/aces/rh1/elearning/multiagents/introduction/nwana.pdf
- Selfridge, Oliver G. "Pandemonium: A Paradigm for Learning." *Mechanization of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory* 1 (1959): 511–529. https://aitopics.org/download/classics:504E1BAC
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*. Advances in Neural Information Processing Systems 30 (2017): 5998-6008. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- Wooldridge, Michael and Nicholas R. Jennings. "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review* 10, no. 2 (January 1995): 115-152. https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/integrated1/woodridge_intelligent_agents.pdf

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.