Building multi-tenant architectures for agentic AI on AWS

# AWS Prescriptive Guidance

# AWS Prescriptive Guidance: Building multi-tenant architectures for agentic AI on AWS

# Table of Contents

# Building multi-tenant architectures for agentic AI on AWS

*Aaron Sempf and Tod Golding, Amazon Web Services*

*July 2025* (document history)

Agentic AI represents a disruptive paradigm shift that requires organizations to rethink how to build, deliver, and operate their systems. The agentic model has teams exploring new ways to decompose systems into one or more agents that create new paths, possibilities, and values.

Much of the agentic discussion centers around the tools, frameworks, and patterns that are used to build and implement agents. We must not only adopt good tools to create agents but also new integration protocols, authentication strategies, and discovery mechanisms that can serve as the basis of agentic architectures.

While the number of agentic tools grows, teams must also consider how their agents address more traditional architecture challenges. Scale, noisy neighbor, resilience, cost, and operational efficiency are fundamental topics that must be evaluated when you're designing, building, and deploying agents. Regardless of how autonomous and smart agents might be, we must also ensure that they achieve economies of scale, efficiency, and agility that align with business needs.

This guide's goal is to explore various dimensions of agentic footprints. This includes reviewing various agent deployment and consumption patterns and highlighting different strategies for creating agents that address architectural goals. It also means looking at how agents might be consumed in a multi-tenant environment by introducing internal constructs that are typically required in a multi-tenant setting.

## Intended audience

This guide is for architects, developers, and technology leaders who want to build AI-driven multi-tenant systems.

## Objectives

This guide helps you do the following:

- Understand multi-tenant agent deployments, exploring both siloed and pooled models, and how tenant context affects agent implementation
- Explore agent management, including onboarding, tenant isolation, and resource management across single- and multi-provider environments
- Evaluate aspects of multi-tenant agents, including data ownership, monitoring, and testing

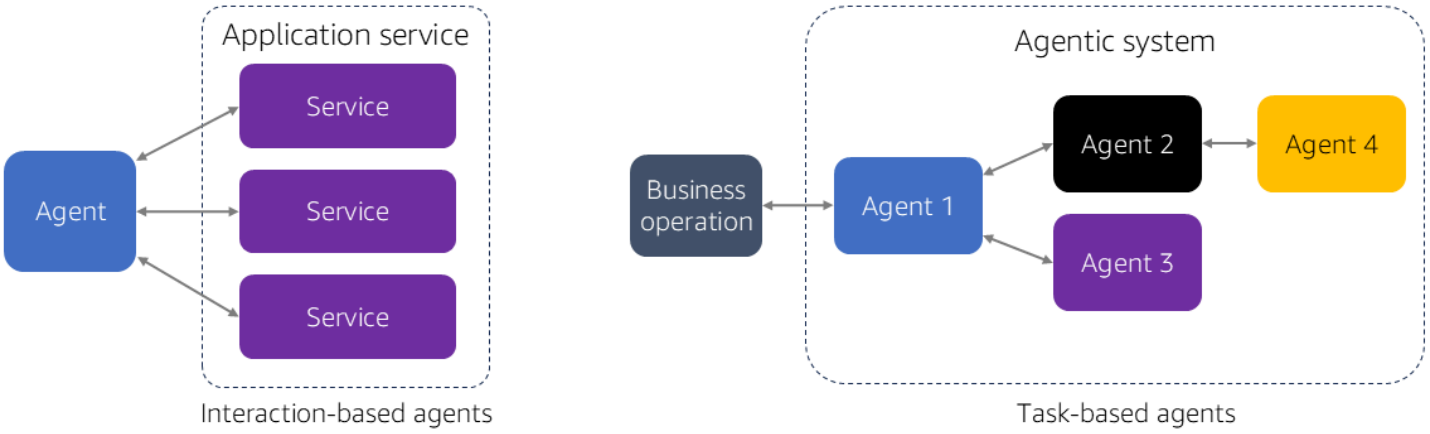
## About this content series

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

# Agent fundamentals

Before we discuss architectural details, we should outline the different roles that agents play because "agent" is an overloaded term that can be applied to many use cases. Let's start with some broad terms that can help categorize them.

At the outermost level, we need to start by classifying the role and nature of agents. This is challenging because there's a wide range of scenarios where agents can be applied to any number of problems. For this discussion, though, we focus on what it means to introduce an agent into an application or system. In this model, we emphasize how and where agents can best enrich your system's experience. The options you choose influence how your agents are built, integrated, and applied to different domains and use cases. The following diagram shows two agentic patterns that builders use.

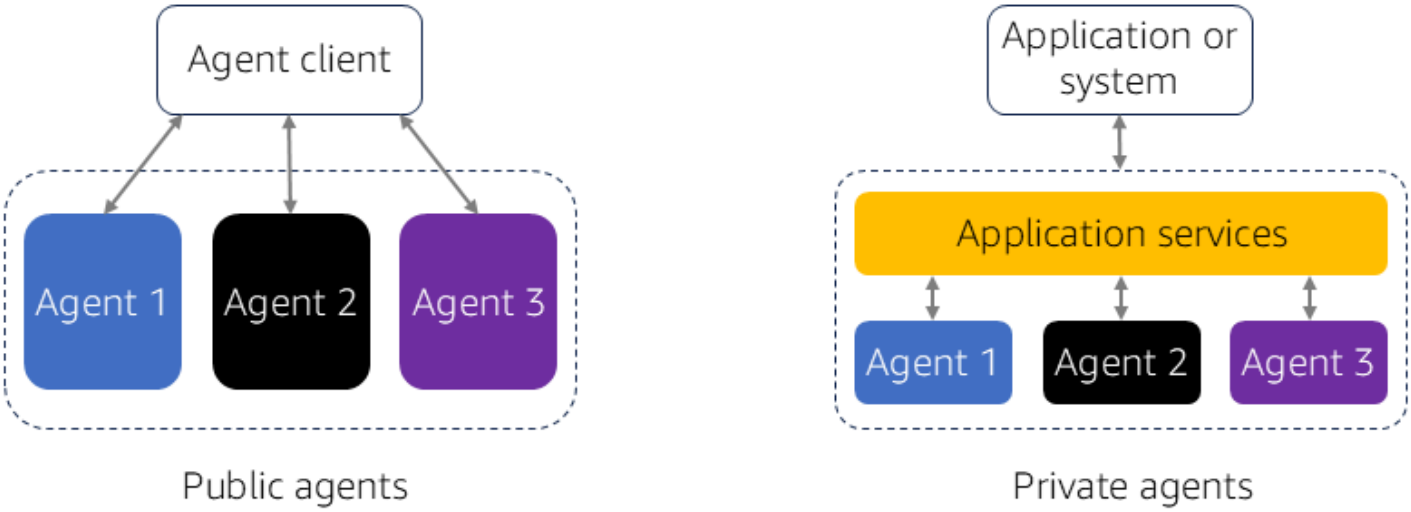



On the left-hand side of the diagram is an interaction-based agent. In this mode, an agent creates a view into an existing system to orchestrate interactions with the underlying services to achieve a goal or outcome. The key is that the agent is added to a system as an alternate approach to driving the system's features and capabilities. Imagine, for example, that an independent software vendor (ISV) has an accounting system with a UX that is used to perform operations. The interaction-based agent simplifies the interaction with these existing capabilities. It's less about learning how to reach a loosely defined goal and more about providing a way of orchestrating known pathways.

In contrast, the task-based system on the right-hand side of the diagram represents a different approach. The agents in that system use their knowledge and abilities to learn to complete tasks and drive business outcomes. You could argue that both models achieve business outcomes, but a task-based model relies on the agents themselves to determine how to achieve an outcome. Such agents are less deterministic and instead rely on their ability to learn and evolve. In contrast,

interaction-based agents are mostly designed to orchestrate a set of known capabilities. These differences affect how you build, scope, and integrate agents to support your business.

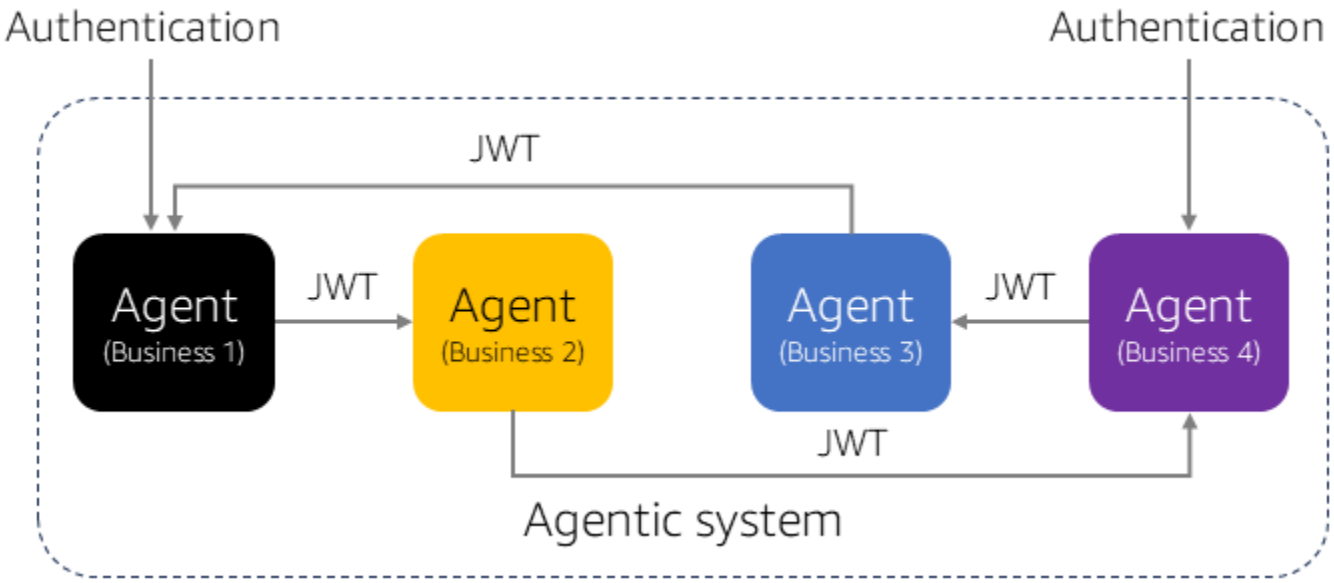
We also need terms that characterize how and where we deploy agents. Where an agent lives within your system's footprint can influence how it's built, scoped, and secured. The following diagram outlines two distinct models that could be applied to agents.




On the left-hand side of the diagram is a deployment system with three different agents. The agents are exposed to external clients that may be other agents or applications. For this model, agents are referred to as public agents.

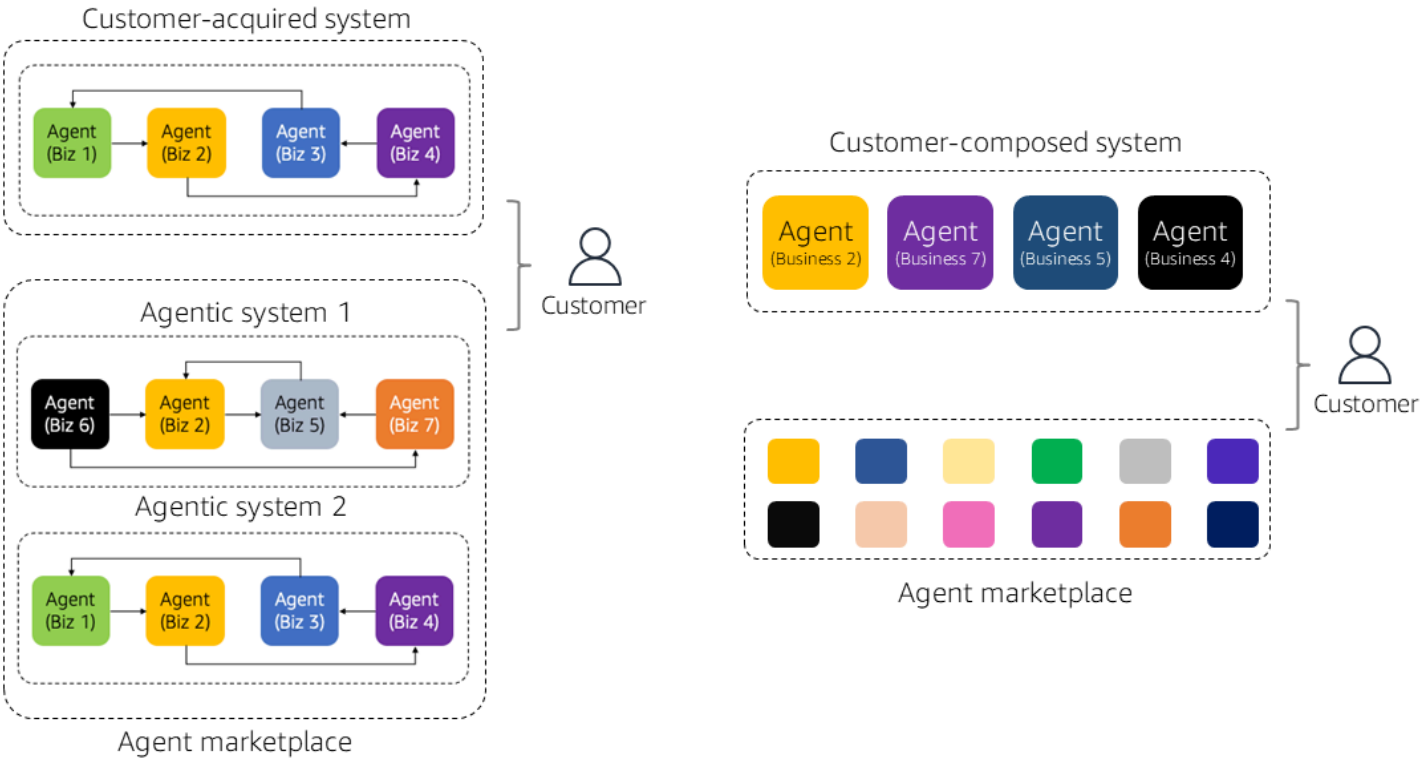
In contrast, the diagram on the right-hand side shows agents within the solution's implementation. In this case, there are a series of application services that are consumed by users or systems. These users interact with the application with no awareness of the fact that agents are part of the experience. The agents are then invoked and orchestrated by the underlying system's services. Agents deployed in this manner are referred to as private agents.

Much of an agent's value focuses on the public model where providers may be publishing their agents with the intention of integrating them with other third-party agents. The agents would then be part of a mesh or web of interconnected services that, collectively, are able to address many use cases. While these agents could be used in many domains, the business-to-business use case is a natural fit. The following diagram provides a conceptualized view of what it might look like to assemble a collection agent that solves a specific problem.




The diagram shows four business agents that work together to achieve a set of objectives. When agents are composed this way, they represent an agentic system, and there're many flavors of such systems. They could be a prepackaged set of collaborating agents that are commonly consumed as a single unit. Or the system could be dynamically assembled by customers who want to pick and choose a combination of agents that best meet their needs.

Both approaches offer viable pathways for agent integration. Some agents are built with the expectation that they will be integrated into specific systems where they can maximize their value, reach, and impact. This notion of agentic systems also raises questions about how agents are acquired, and there could be many ways to address this. The following diagram provides examples of how these agents and systems can be created through transactional experiences.

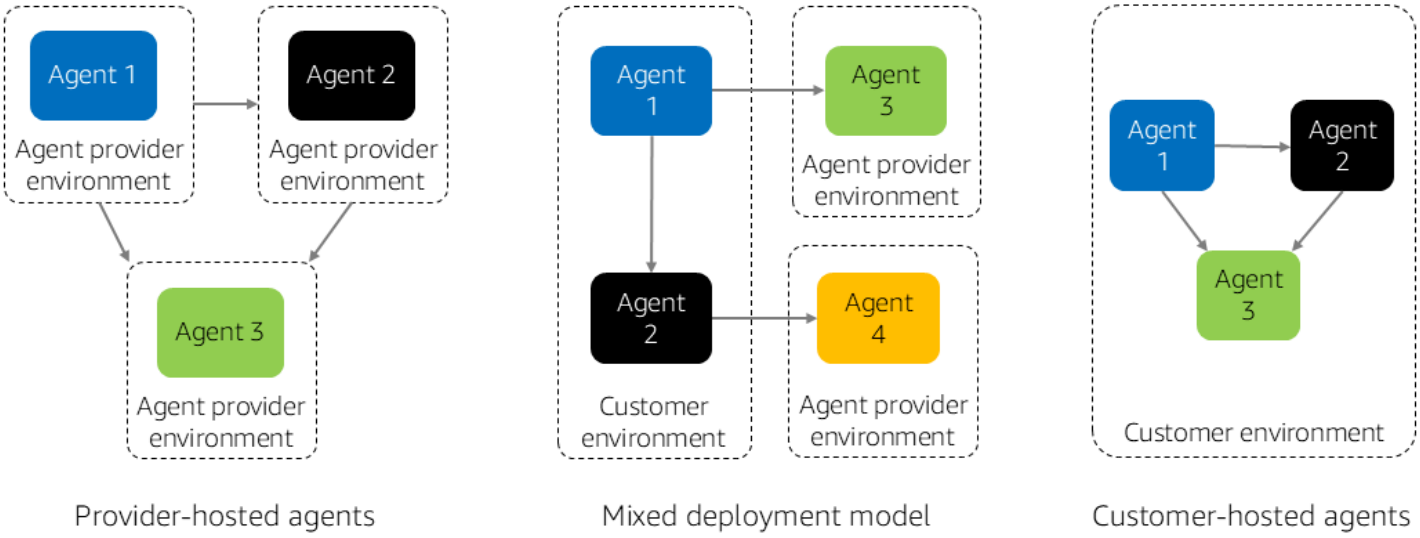



Two examples of marketplace experiences are shown. On the left-hand side, a marketplace is used to acquire prepackaged systems. In this scenario, the marketplace discovers and onboards systems that address broader objectives that require the integration and orchestration of multiple agents.

The example on the right-hand side shows a marketplace where agents are discovered and composed into agentic systems. In this scenario, customers can build any system of compatible, integrated agents to meet their needs. The ability to assemble agents in this manner depends on the compatibility model and integration requirements of individual agents.

# Agent hosting considerations

Now that you have a sense of the broader agentic concepts, let's discuss what it means to host and run these agents. We must think about how and where computations run, how they scale, how they operate, and how they're managed. At the same time, some patterns that we expect to see as agents are more widely applied and adopted. The following diagram shows an example of likely permutations.




Three distinct strategies are represented here. On the left-hand side of the diagram, you see a model where our agents are hosted, scaled, and managed within the environments of each agent provider. These agents are published and consumed as services, operating in what is labeled as an agent as a service (AaaS) model. On the right-hand side is a model where a provider's agents are all hosted in a dedicated customer environment.

In the middle of the diagram is a mixed deployment model that combines these two strategies, hosting some agents locally in the customer's environment and interacting with some agents that are hosted remotely in a provider's environment.

A fourth option (not shown) could be where agents are built as low or no-code services that are scaled and managed by agent infrastructure services. We won't cover these in detail because the architecture and hosting of managed agents is dictated primarily by the organization that owns the services.

You can imagine the range of factors that might influence the adoption of one of these models. Compliance, regulatory, and security constraints, for example, could push someone toward

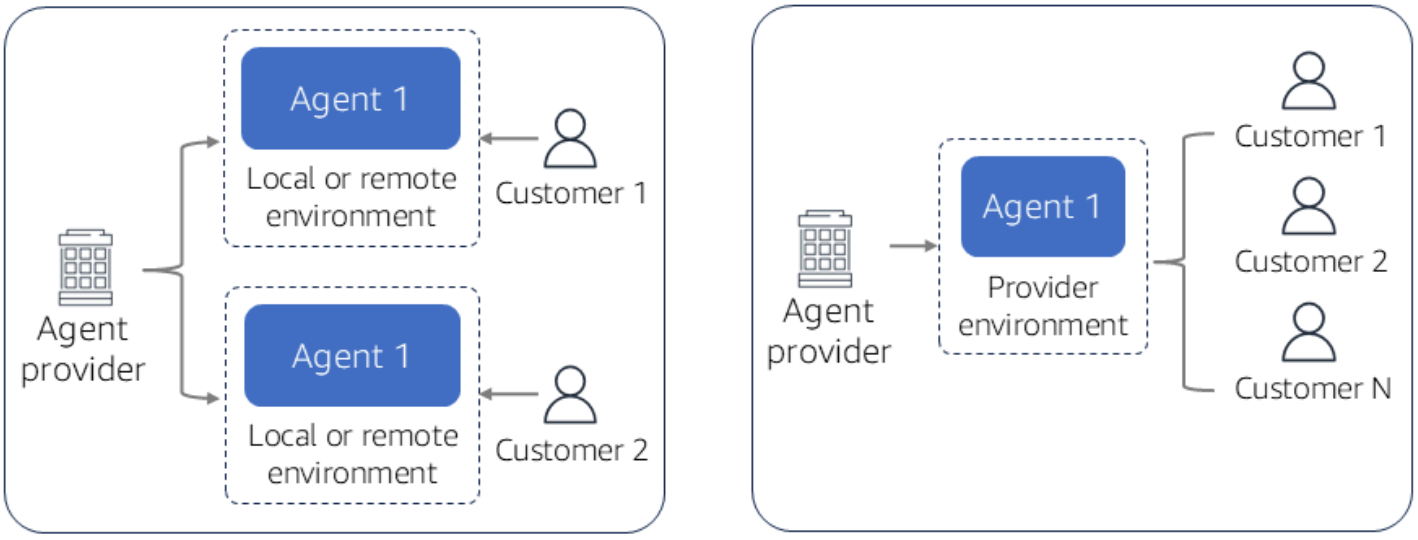
customer-hosted agents. Scale, agility, and efficiency could push organizations more toward the AaaS model.

The key concept here is that agents can and are deployed and hosted in many ways. It's your job to determine how agents can best be applied. The footprint, security, and deployment, among other factors, significantly affects how you approach building and operating agents. Private and public agents, for example, may have different designs and release lifecycles.

# Agents meet multi-tenancy

It's easy to think of agents as building blocks where agents are viewed as a series of autonomous components that are assembled to support the needs of a specific domain or business problem. Where it gets more interesting is when we start to think about how these agents are packaged and consumed by providers. In many respects, an agent becomes a source of cost and revenue for a business. Agent providers must consider the different personas that consume their services, the consumption profile of the personas, and the monetization strategies that allow agent providers to create pricing and tiering models that align with consumers.

Agent providers could support multiple models for deploying their agents to meet customer needs. The following diagram shows a conceptual view of the two main agent deployment models.



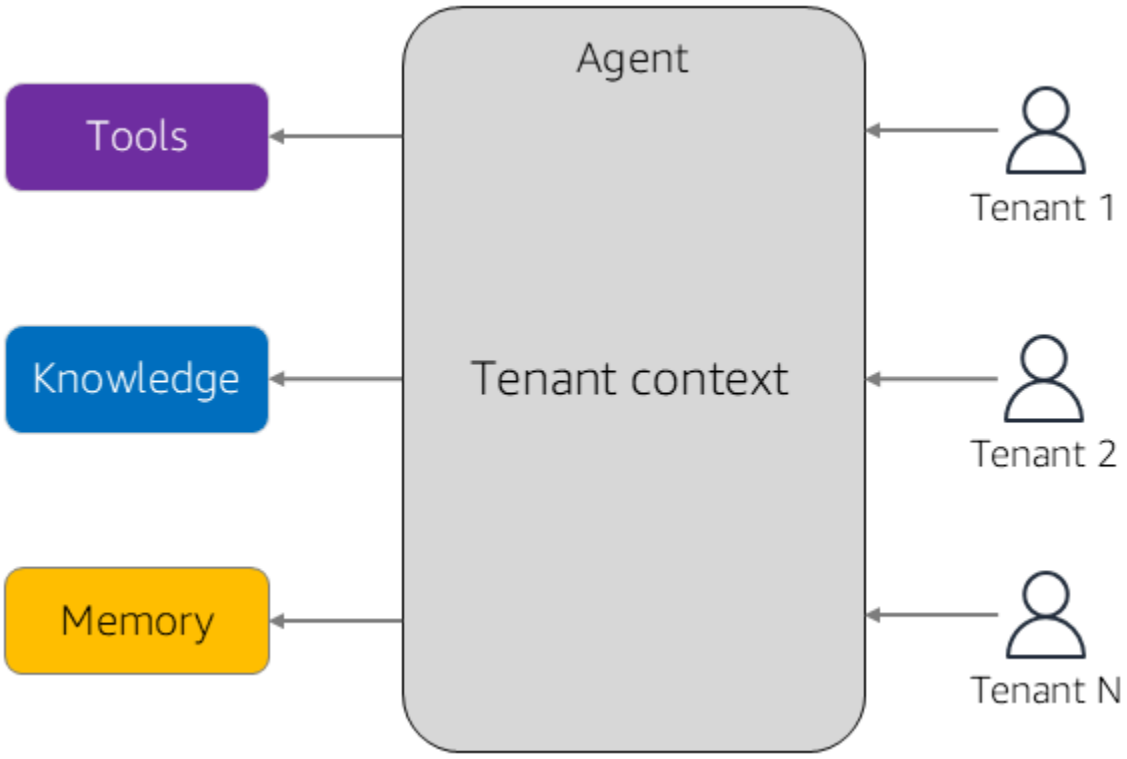

The left-hand side of the diagram shows the customer-dedicated agent model. An agent provider builds an agent by deploying a separate agent instance for each onboarded customer. With this approach, the capabilities of the agent and its ability to acquire knowledge would be limited to the scope of a given customer's environment. This ends up representing a per-customer experience that inherits some of the complexities and advantages of supporting dedicated customer environments.

In contrast, the diagram on the right-hand side of the diagram has a single agent that is deployed in the provider's environment. The agent processes requests from multiple customers, evolving and learning based on the collective experience of all customers. Each new customer that's added would simply represent another valid client of the agent. The agent runs like an agent as a

service (AaaS) model, using shared constructs to support a client's needs. In both instances, agent consumers can be applications, systems, or even other agents.

There are two ways you can look at the AaaS model. The model above delivers the same experience to all customers. This means the internals of the agent will not include any level of specialization that considers the context of the requesting client. Generally, for this mode, the assumption is that the nature of an agent's scope, goals, and value centers around a shared set of resources, knowledge, and outcomes that are applied universally to all clients.

The alternative approach to AaaS is where the context of clients influences the agent's experience and implementation. The following diagram provides a conceptual view of an AaaS agent footprint in this context.

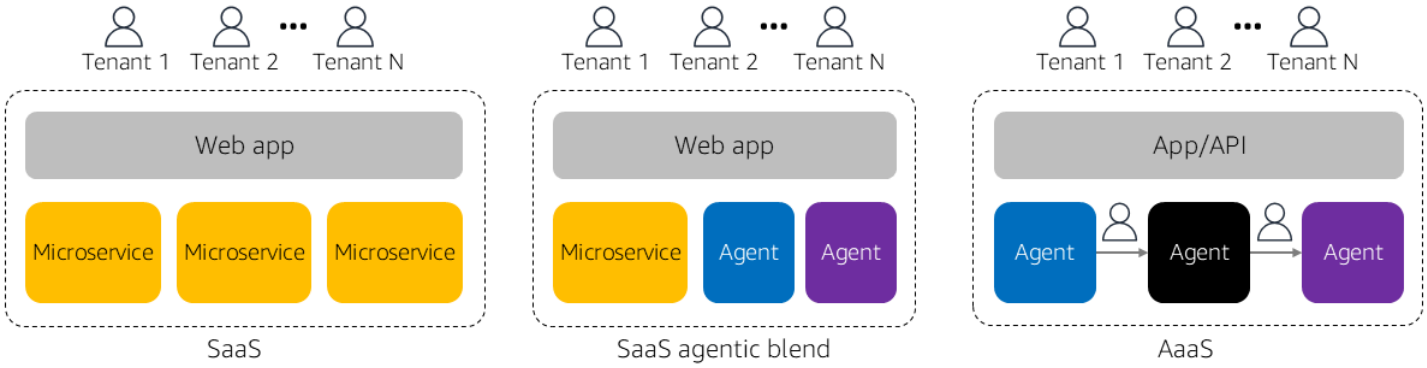



In this AaaS view, the origin and context of the incoming requests significantly affect the agent's footprint. The resources, actions, and tools that are part of the agent's underlying implementation may vary for each incoming tenant request. The value of an agent is connected to its ability to use tenant context to arrive at actions and outcomes that are influenced by tenant state, knowledge, and other factors. Some requests may yield a unique tenant outcome, and others may lead to more tailored per-tenant outcomes. This adds a new dimension to the agent's ability to learn, which could include being more contextual and acquiring and applying knowledge that enhances targeted outcomes.

For providers, the AaaS model offers many advantages. With multiple customers consuming a single agent, the provider has a better opportunity to achieve economies of scale, drive operational efficiency, control costs, and create a unified management experience. This has the potential for greater agility, innovation, and growth for the agent business.

These qualities overlap with the same principles that drive the adoption of the software as a service (SaaS) model. Essentially, the AaaS model is built as a multi-tenant service that inherits many of the same scale, resilience, isolation, onboarding, and operational attributes that are found in a SaaS environment. In many respects, the AaaS experience borrows heavily from the strategies and practices used by SaaS providers, but it's reasonable to separate these terms. For our purposes, the emphasis is mostly on the implications that come with building and operating agents that require multi-tenant support.

For a system that can treat all users equally and doesn't require the management of persistent, sensitive, or customer-specific data, the notion of tenancy would minimally affect their agents. For systems that are expected to serve multiple customers while preserving data isolation, customization, and context awareness, supporting multiple tenants could be an essential element of an agent's design, strategy, and goal. The following diagram shows how multi-tenancy can be used in agentic environments.




On the left-hand side of this diagram is a classic multi-tenant architecture. It includes a web application and a series of microservices that implement business logic. Multiple tenants consume the shared infrastructure of this environment, scaling to meet the shifting workloads of a tenant population that evolves. The environment is operated and managed through a single pane of glass for all tenants.

Imagine how this mental model maps to the agent on the right-hand side of this diagram. One agent runs an AaaS model that's consumed by one or more tenants. The agents could be from multiple providers with tenant context flowing between them because a single instance of one agent must process requests from multiple tenants.

The example in the middle of this diagram is a hybrid model where agents are part of the overall SaaS experience. Some parts of the system are implemented in a more traditional model and other parts of the system rely on agents. This pattern is likely to be common for many SaaS offerings—especially for organizations that are transitioning to an agentic experience. It's common for this model to persist because not all systems are delivered as pure AaaS. Also note that multi-tenancy still applies to the model's agents. While the agents may be embedded within a system, they may still process requests from multiple tenants.

It's natural to ask whether multi-tenancy really matters. You could argue that an agent processes requests, so supporting tenancy may have little effect. But as we dig deeper into multi-tenant agentic implications, tenancy may directly affect how agents influence how tools, memory, data, and other agent parts are accessed, deployed, and configured to support individual tenants. Tenancy also influences how scaling, throttling, pricing, tiering, and other business aspects apply to your agent's architecture.

One takeaway from this is that there are agentic use cases that require multi-tenancy support. The challenge is to determine how multi-tenancy shapes the overall design and architecture of your agentic experience. For some agents, multi-tenant support represents a differentiating capability, allowing agents to apply tenant-specific context to agents that deliver targeted outcomes.

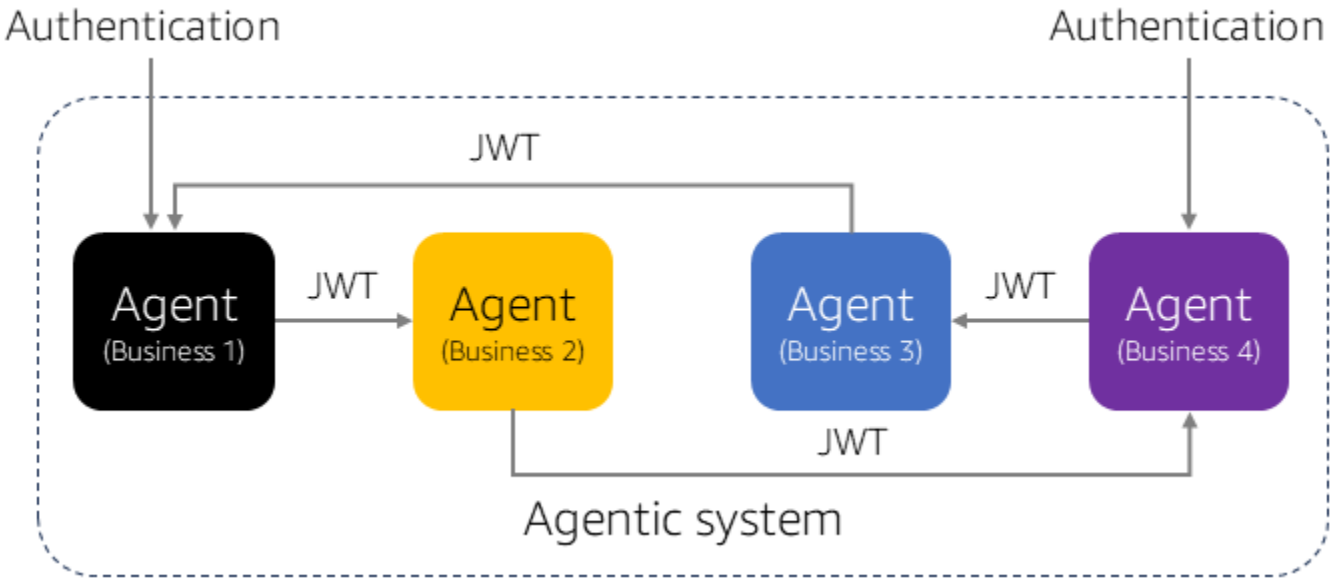
In subsequent sections, you'll see how the terminology and design patterns that we create to describe multi-tenant SaaS architectures will be useful. These concepts can be adopted by the AaaS model by borrowing useful aspects, which introduces new agent-specific concepts where they're needed.

# Identity, tenant context, and agentic systems

Adding tenant context to individual agents isn't particularly challenging. In many instances, teams can rely on typical mechanisms that bind users and systems to tenants and pass tenant-aware tokens to agents. This is relevant when we consider how tenant context and identity supports multiple agents. In this model, tenants must be bound to an identity that spans all collaborating agents.

In general, the agentic domain requires a more cross-cutting identity model that aligns with the current and emerging needs of agentic systems. Agent providers require identity mechanisms that support unique security, compliance, and authorization models that come with operating agentic systems. This is especially challenging in environments where systems are composed by customers or other agents. Each onboarded agent must connect its identity and tenant context to agent

interactions. The following diagram highlights the potential identity and tenant context challenges that are part of agent-to-agent (a2a) interactions.




This diagram shows a series of provider-built agents interacting as part of the agentic system we covered. It's now retrofitted with identity and tenant context. This scenario is an example of an agentic system that supports multiple entry points. We assume that each agent in this system requires its own authentication mechanism to resolve the system or user to a given tenant. As these agents interact, tenant context is passed to a JSON web token (JWT) that will be used to authorize access and inject tenant context into the agent.

Conceptually, the main difference with this scenario is that agents deploy and operate independently, which means that each agent must be able to resolve its identity and authorize access. The key is that its identity must have some distributed ability to handle the needs of the broader agentic system. There must also be alignment on how agents share tenant context.

## Applying SaaS business value to AaaS

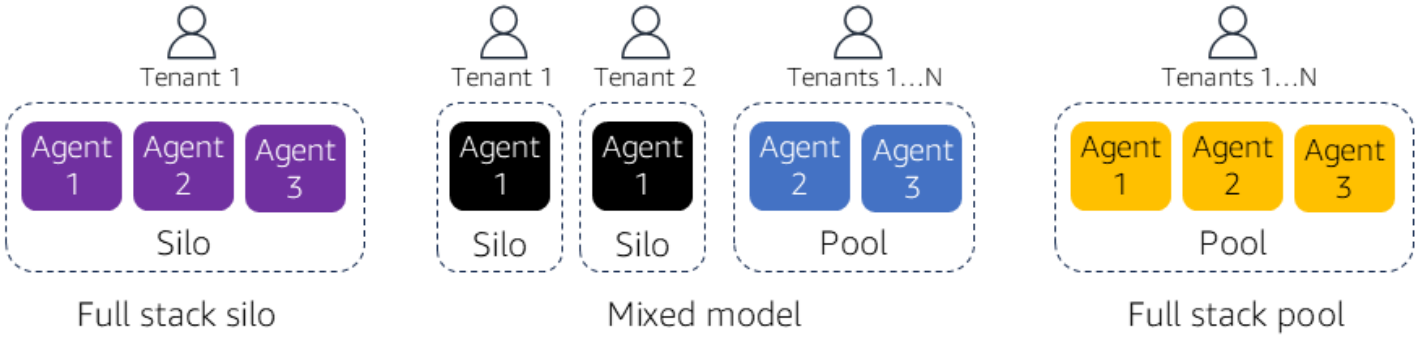
Generally, when we look at running any system in an as-a-service model, we consider the nature of the experience and how its technical and operational footprint drives business outcomes. When adopting SaaS, for example, organizations use economies of scale, operational efficiencies, cost profiles, and agility to drive growth, margins, and innovation.

Agents delivered as AaaS are likely to target similar business outcomes. By supporting multiple tenants, an agent can align resource consumption with tenant activities. This yields economies of scale that come with traditional SaaS environments. AaaS also allows organizations to manage,

operate, and deploy agents in a way that enables frequent releases and drives agility for agent providers. The key is that the AaaS model doesn't depend on technology. It creates and drives business strategies that promote growth, streamline adoption, and simplify operations.

# Agent deployment models

In a basic AaaS experience, a provider may deploy agents using various patterns. There are myriad factors that influence how agents are deployed to meet customer, performance, compliance, geography, and security needs. Different deployment strategies affect how an agent is designed, implemented, and consumed. It's here that we can introduce classic multi-tenant terms to label different deployment strategies. The following diagram shows different permutations for deploying agents in an AaaS environment.

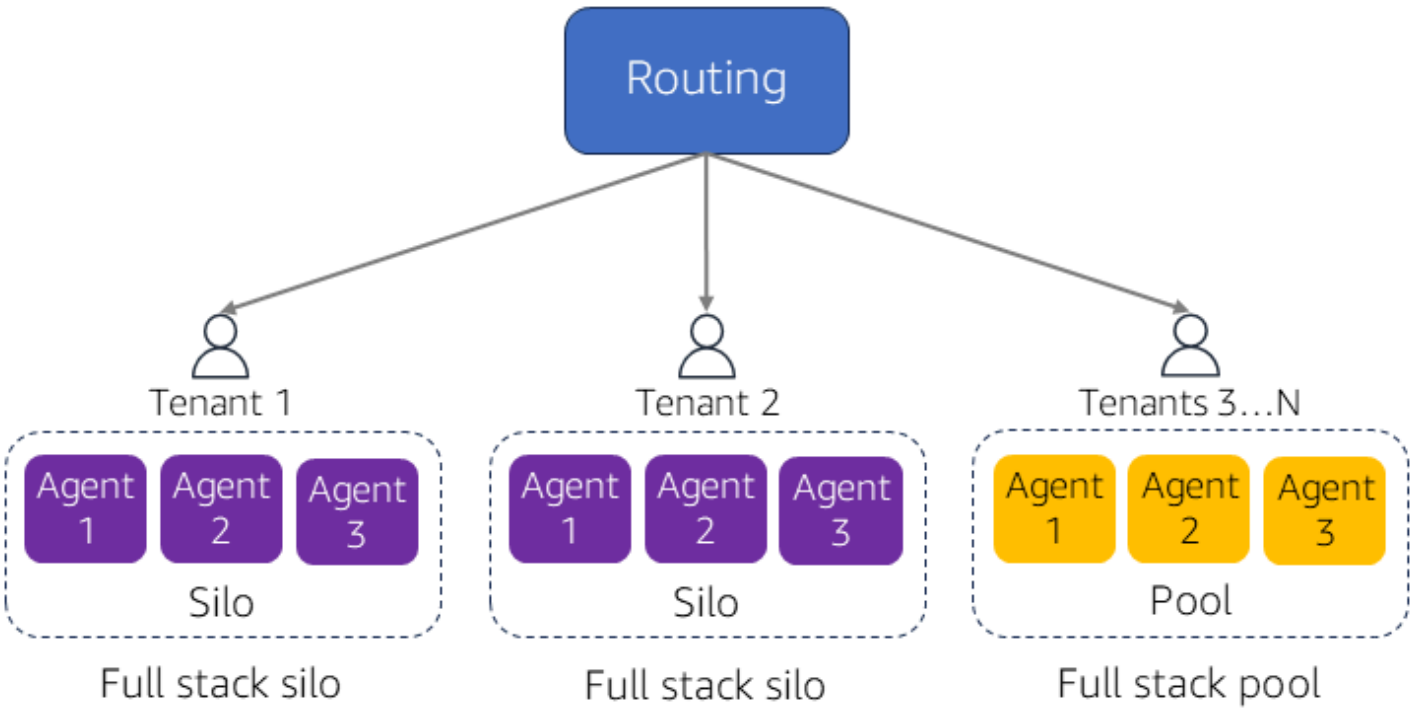



This diagram represents three modes of agent deployment. On the left-hand side is a siloed model, where each tenant is provided with a fully isolated experience and a dedicated set of agents. In this scenario, agents do not share compute, resources, or execution environments across tenants.

The middle example illustrates a hybrid model, where tenants use a combination of siloed and pooled agents. For instance, Agent 1 is deployed in siloed mode—each tenant receives a dedicated instance—while agents 2 and 3 operate in a pooled model, sharing resources across tenants.

On the right-hand side is a fully pooled model, where all agents are shared across tenants, offering a classic multi-tenant deployment. In this scenario, tenants leverage common compute, memory, and service infrastructure for agent execution.

The idea is that agents can operate in different deployment models, with compute and dependent resources either dedicated (siloed) or shared (pooled) across tenants. These deployment strategies are not mutually exclusive. Agent services often support a spectrum of customer needs, combining both models to balance performance, isolation, cost, and scalability. The following diagram shows an agentic system that supports multiple deployment configurations within the same operational environment.

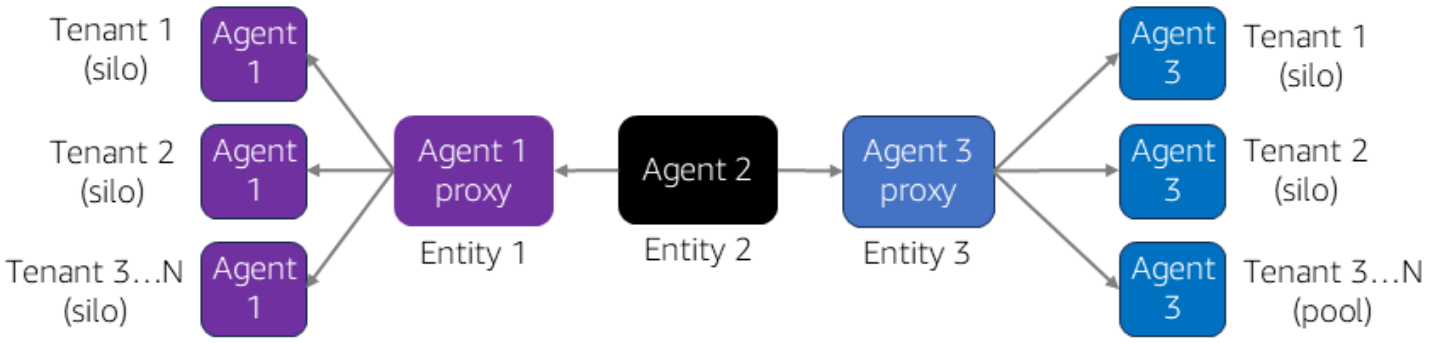



In this diagram, an agent provider has three agents that are deployed through agent as a service (AaaS). They support two types of tenants. On the left-hand side, two tenants have compliance and performance requirements that they address through a full-stack silo model. The remaining tenant on the right-hand side runs in a pooled model where tenants share resources.

If the goal is agility and operational efficiency, try to limit the effects associated with supporting per-tenant deployment models. This means putting routing and other experience mechanisms in place that allow agents to be managed, operated, and deployed through a single pane of glass.

If you build an agent in a low- or no-code environment, there will be no notion of siloed or pooled agents. Instead, agents may be fully managed by another agent. Siloed and pooled models apply more to environments where an organization controls the agent's construction and footprint. In this case, teams should consider which deployment model to support.

On the surface, these deployment models don't directly affect how an agent functions in a broader system. An agent may have no direct awareness of other agents that are deployed in a silo or pooled model. Instead, these deployment strategies can be implemented as part of a routing construct within an environment. The following diagram shows an example of how siloed and pooled models can be implemented using a routing strategy.



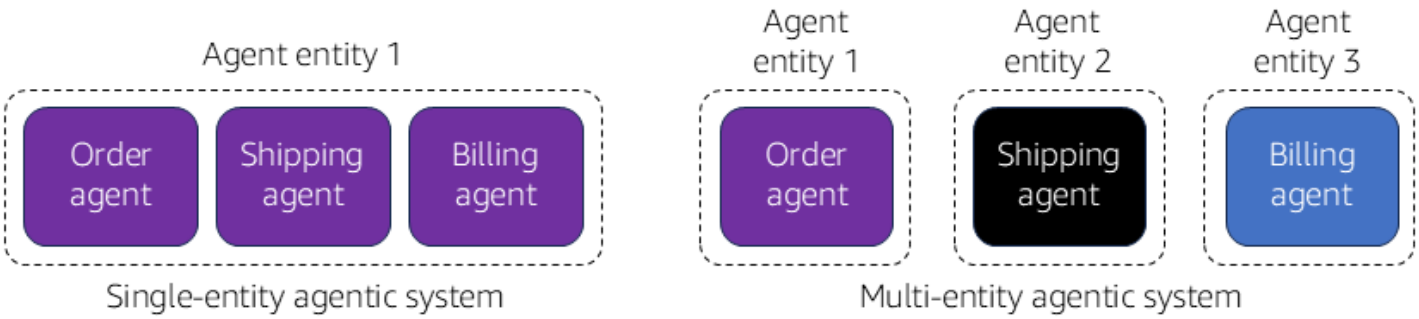

This example includes three agents from three different providers. Each agent provider has the option to implement its own deployment strategy. For example, agent 1 uses a proxy to distribute inbound requests to a set of siloed tenant agents. Agent 2 requires no routing and supports all tenant requests through one pooled agent. Agent 3 is a hybrid-model deployment where some tenants are siloed and others are pooled.

If and how you choose to support these deployment models depends on the nature of your solution. You may have no need to support either model. You may, however, have instances where you must consider supporting this strategy, such as with compliance, noisy neighbor, performance, or tiering.

# Introducing and applying tenant context

If we build agents that support multi-tenancy, we must start by considering how to set up tenant context, which will be used to apply tenant-specific policies, strategies, and mechanisms within the agent's implementation.

At the most basic level, you can introduce tenant context into agents though the common tools and mechanisms that we use in classic multi-tenant architectures. This could be through an API key, OAuth, or various other validation mechanisms. Many examples of this focus on resolving an authenticated system or user to a JSON web token (JWT) key that holds tenant context. The JWT is then propagated through the system. This gets more interesting when we consider how to compose agentic systems. The following diagram shows an example of two varieties of agentic environments.




In this diagram, the model on the left-hand side represents an agentic system where all of the agents are owned, managed, and hosted by a single entity. When you have full control of the entire experience, you can use typical strategies to pass tenants through each agent.

The model on the right-hand side, which may be more common, represents a system of agents that span multiple entities. The agents are independently built, managed, and operated, so they each have their own authentication and authorization schemes. The challenge here is that we need a universal way to resolve and share tenant context among these agents. This relies on a more distributed model where each agent must be able to authenticate systems or users and resolve them to a tenant according to applied mechanisms.

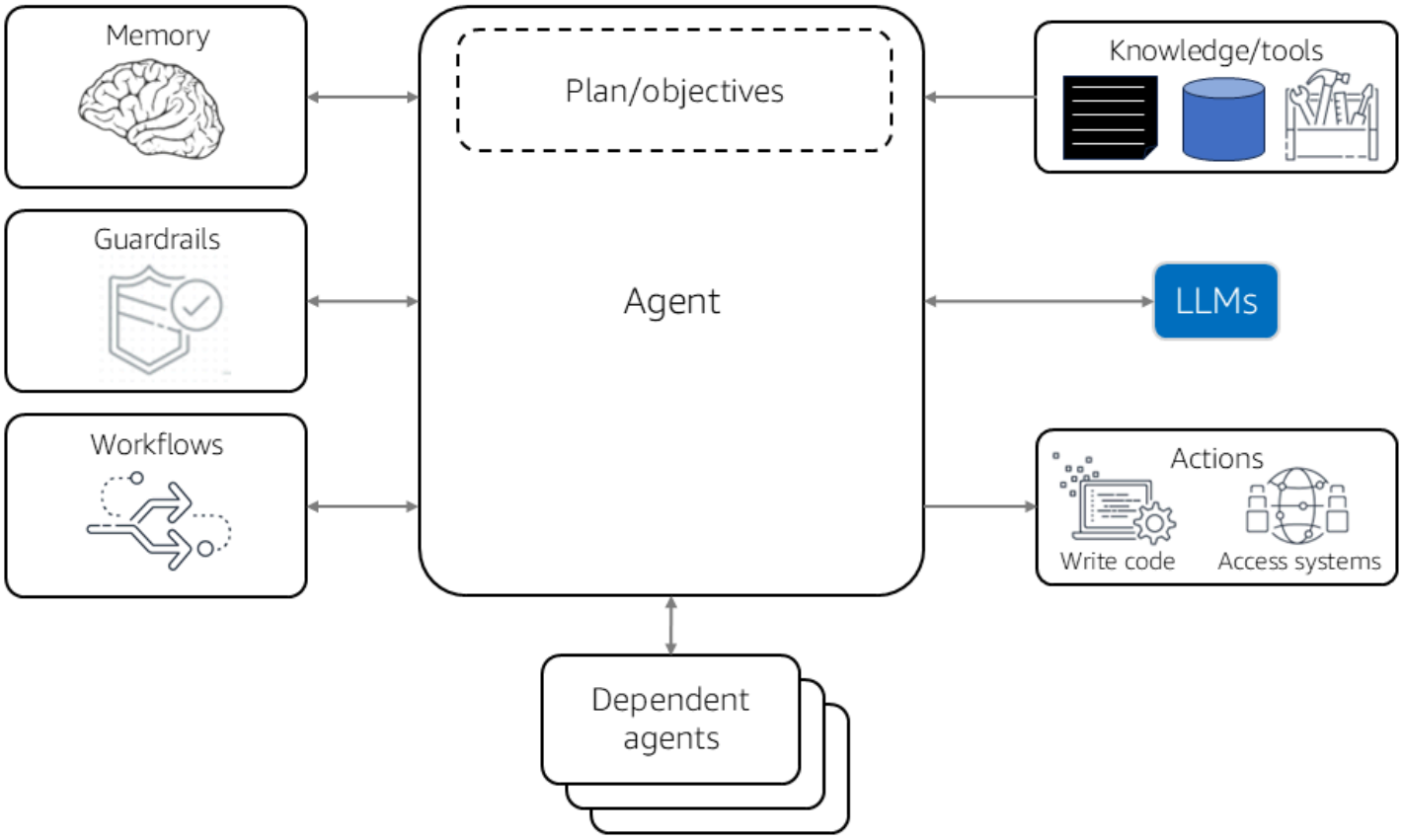
# Building tenant-aware agents

Multi-tenancy influences how we implement individual agents. As an agent processes requests, consider how tenant context affects how an agent accesses data, makes decisions, and invokes

actions. To better understand how and where multi-tenancy affects your agent's profile, first determine how constructs can be part of any agent.

The challenge is that the scope, nature, and design of agents is anything but concrete because providers make their own choices about the design of an agent experience. Ultimately, the point of an agent is that it's an autonomous learning service that can access a range of tools, data sources, and memory to determine how best to solve a task.

It's less important to know exactly which strategies and patterns an agent uses. In a multi-tenant model, it's more important to identify how various parts of an agent are configured, accessed, and applied. Consider a potential agent environment that relies on a series of resources and mechanisms to achieve its goals. The following diagram shows an example of such an agent.

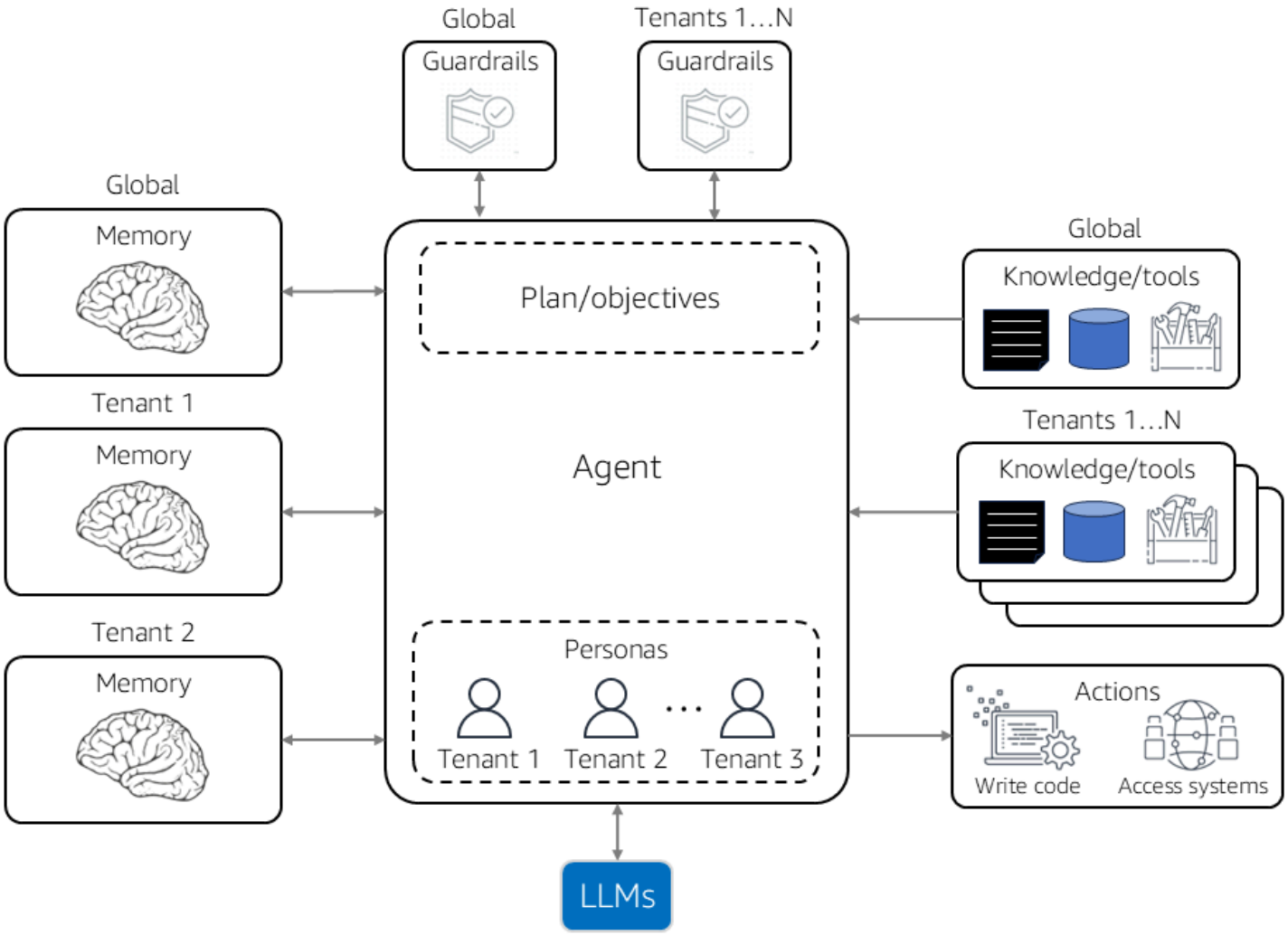



This diagram represents a comprehensive range of agentic possibilities, showcasing various tools and mechanisms that could be combined to accomplish a goal. On the left-hand side of the diagram, note how an agent depends on memory as part of its context, guardrails for defining the policies that guide its activities, and workflows that are directed at specific tasks. Some might argue that workflows shouldn't be included in this context, but there may be scenarios where workflows are integral to an agentic experience.

The right side of the diagram shows how inputs like knowledge and tools can supply additional insights and context that enhance the agent's capabilities. The agent then outputs actions, such as writing code or accessing systems. The bottom of the diagram shows how agents depend on one or more internal or third-party agents that can be orchestrated as part of a broader system.

We can now think about what it means to introduce multi-tenancy. Tenancy forces us to consider how and where an agent introduces strategies and mechanisms that dictate behaviors and actions. This adds another dimension to how we think about agents in terms of their knowledge, learning, tools, and memory.

Let's now consider how to modify this model to support multi-tenancy. The following diagram shows an example of a multi-agent model.




In this diagram, we introduce tenant personas that are intended to shape how an agent integrates tenant context. For example, on the left-hand side of the diagram, agent memory is altered to support tenant-specific memory. The same is true on the right-hand side of the diagram where

the agent supports tenant-specific knowledge and tools. The same support is also applied to guardrails.

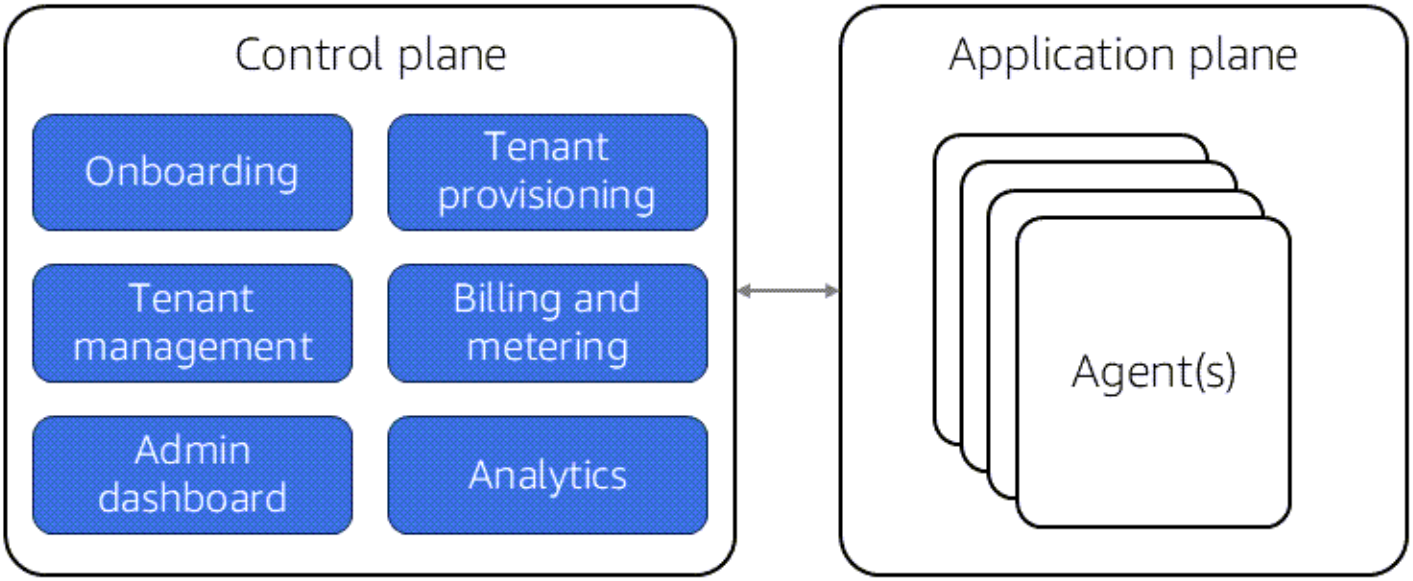
This may be an extreme example because not all aspects of a multi-tenant agent require per-tenant resources. The point is that you should consider how tailoring your agent for specific tenants can enhance its effectiveness. This approach allows your agent to increase its impact and value, provide more relevant context in its responses, and develop specialized capabilities. The agent will then be able to learn, adapt, and perform tasks that are uniquely suited to different personas.

The main idea is that tenant context directly affects how you build agents. It can also shape tenant interactions with external entities, including other agents. Building a multi-tenant agent introduces traditional challenges such as noisy neighbors, tenant isolation, tiering, throttling, and cost management. Your agent's design and architecture must address these foundational multi-tenant concepts, which we'll explore in the next section.

# Employing control planes in agentic environments

Multi-tenant best practices often divide implementations into two distinct parts: a control plane and an application plane. The control plane provides a single pane of glass to access operational, management, and orchestration mechanisms that span the environment's tenants. The application plane is where the business logic, features, and functional capabilities reside.

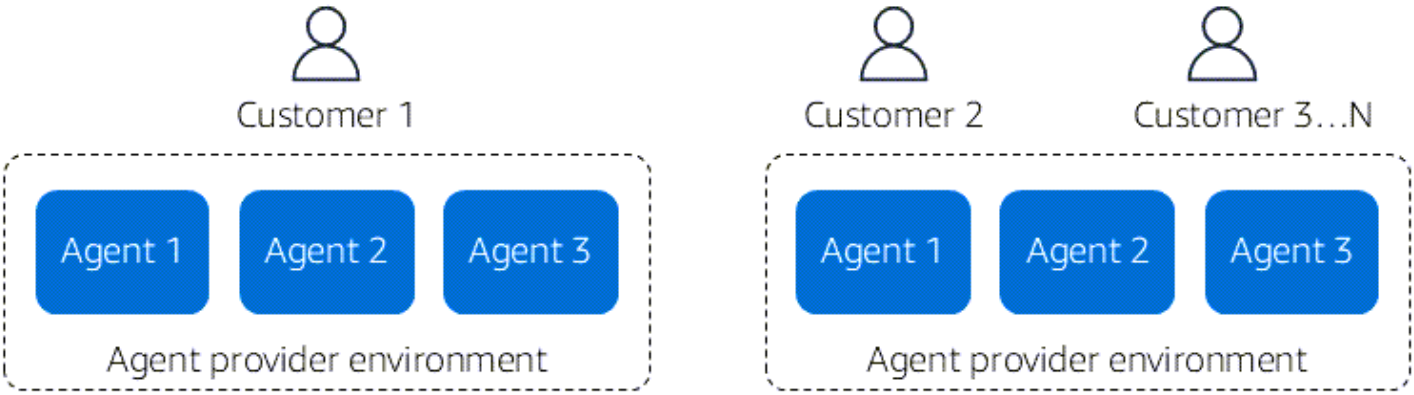
This division of responsibility also applies to agentic models. A multi-tenant agent requires a degree of centralized management, operation, and insights, and it makes sense to continually address these needs through a control plane. The following diagram shows a conceptual view of how these planes are divided within an agent as a service (AaaS) environment.




This diagram shows the traditional separation of control and application planes. What's new is that the control plane now manages the agents that make up an AaaS environment. The control plane interacts with all agents because we presume that the agents are built, managed, and deployed by one provider.

This model introduces additional layers of complexity, especially in agent lifecycle and third-party coordination, but retains the foundational separation of concerns. The control plane still provides the same core capabilities by orchestrating the configuration of agents, providing tenant and agent observability, collecting consumption and metering data for billing, and managing tenant policies.

This scenario becomes more complex if you consider a multi-agent system that incorporates agents from various providers. The following diagram shows an example of such a model.




This diagram depicts four agents from different providers that are part of multi-agentic system. Third-party providers still operate and deploy each agent, which are configured to enable authorized access from one or more providers. The agents, however, remain under the provider's control, so each agent maintains its own control plane.

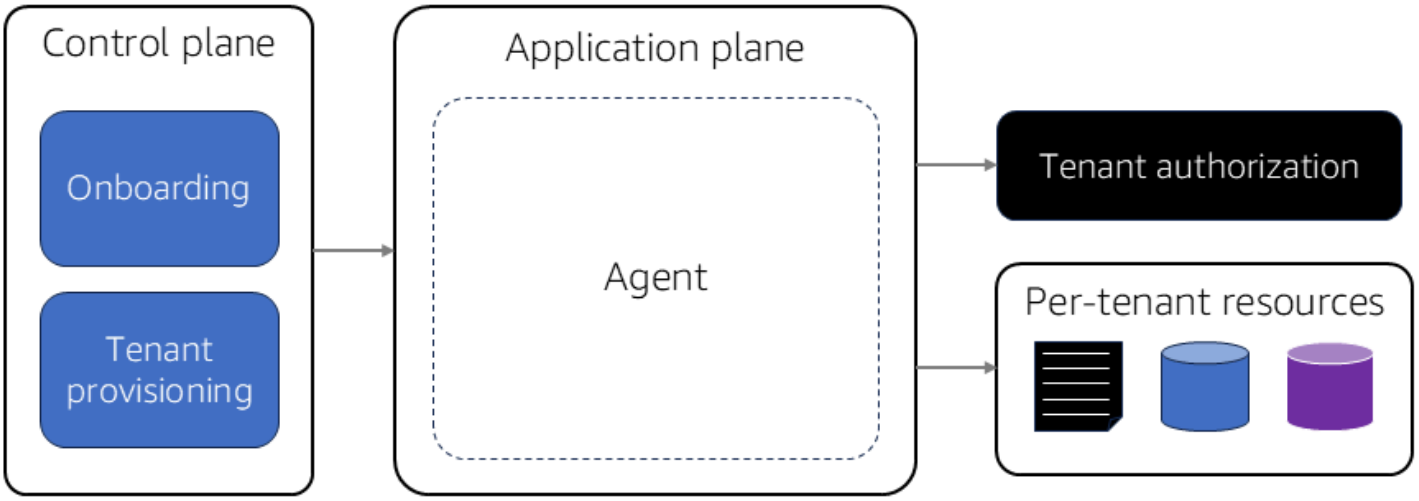
Essentially, these multi-tenant agents behave as third-party services that integrate with other agents. As such, they must have their own control plane to provide the centralized operation, configuration, and management of an agent's capabilities.

We assume that agents are independent services that run in a provider-hosted experience. But this may be unclear in a scenario where an agent consumer imposes more constraints on how and where to host an agent.

## Onboarding tenants to agents

Onboarding is typically a vital part of any AaaS environment. How you create, configure, and provision tenants often involves many moving parts, integrations, and tools. The agent onboarding experience may require the same services that are found in an AaaS control plane, which includes tenant identity, tiering, provisioning per-tenant resources, and configuring tenant policies.

Your approach to agent onboarding is influenced by the footprint and tenancy model of your agentic environment. Siloed and pooled agents each have their own nuances, and the choice of using either a single agent or multiple agents also affects the onboarding process. The following diagram shows a conceptual view of how onboarding affects an agent's configuration.




Each time you onboard an agent, the control plane must take the necessary steps to enable the tenant to access the agent. How to introduce tenants varies based on the agent authorization model, but assume that you'll create a tenant identity that associates agent requests with individual tenants. This tenant context dictates the agent experience by applying it to routes, scopes, and control access.

Onboarding may also require you to configure any per-tenant resources that an agent uses. It's here that the control plane's tenant provisioning service connects your agent to tenant-specific data and resources that the agent consults.

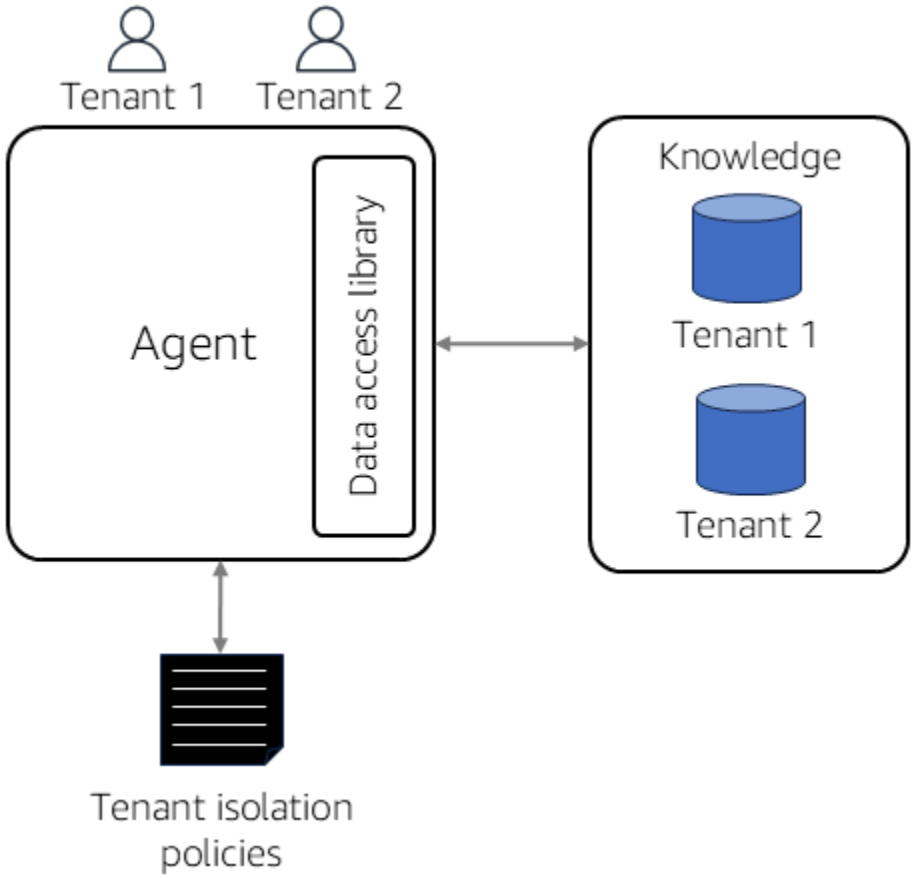
If your system relies on integrating third-party agents, you must also address the needs of those agents during the onboarding process. How this works depends on the security and integration mechanisms for authorizing access between agents. Ideally, the required steps to orchestrate and configure agent-to-agent authentication and authorization are addressed through automated onboarding.

# Enforcing tenant isolation

Tenant isolation is a concept that applies to all multi-tenant settings. It means that your policies and strategies ensure that one tenant can't access other tenant resources. For multi-tenant agents, you may need to introduce constructs and mechanisms that help enforce and agent's tenant isolation requirements.

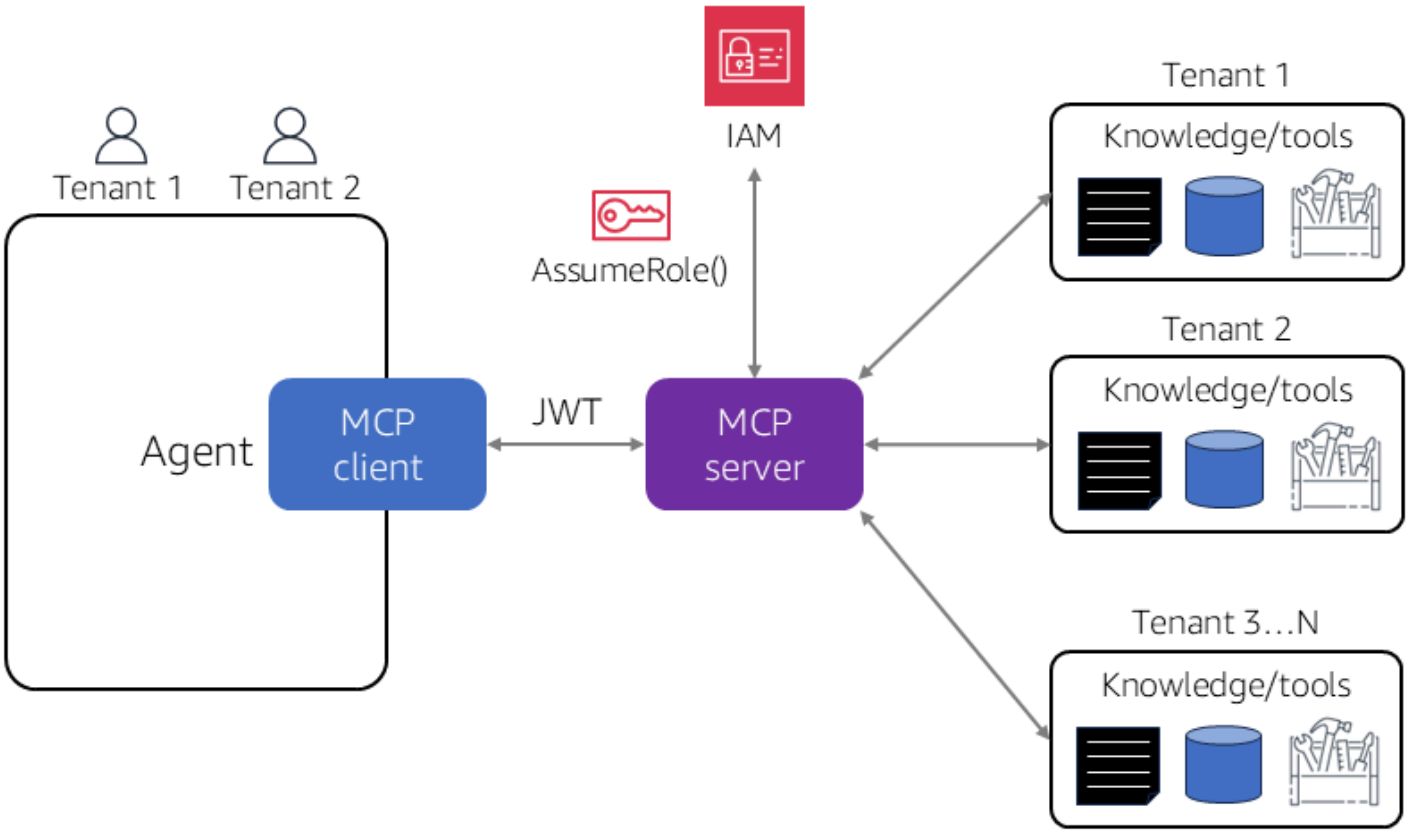
Applying tenant isolation is like other strategies that use traditional multi-tenant systems. Generally, when you construct an AaaS architecture, identify any area in your system where a request or action can access resources to determine if the request crosses any tenant boundaries. For example, microservices may have dependencies on per-tenant, dedicated Amazon DynamoDB tables. This requires you to introduce policies that ensure that one tenant's table can't be accessed by another tenant.

In this instance, consider tenant isolation through an agent lens and its interactions with any of its per-tenant resources. The following diagram shows a conceptual example of how agents apply tenant isolation policies to control access to tenant resources.



On the right-hand side of this diagram, the agent has per-tenant knowledge that's stored in separate vector databases. As the agent processes a request, it examines the context of the tenant making the request. Based on this, the agent applies an appropriate isolation policy to ensure that tenants are restricted from accessing data or resources outside of their designated boundaries.

If your agent uses a Model Context Protocol (MCP), it can also implement your tenant isolation model. The following diagram shows an example of how to introduce MCP and apply isolation policies.



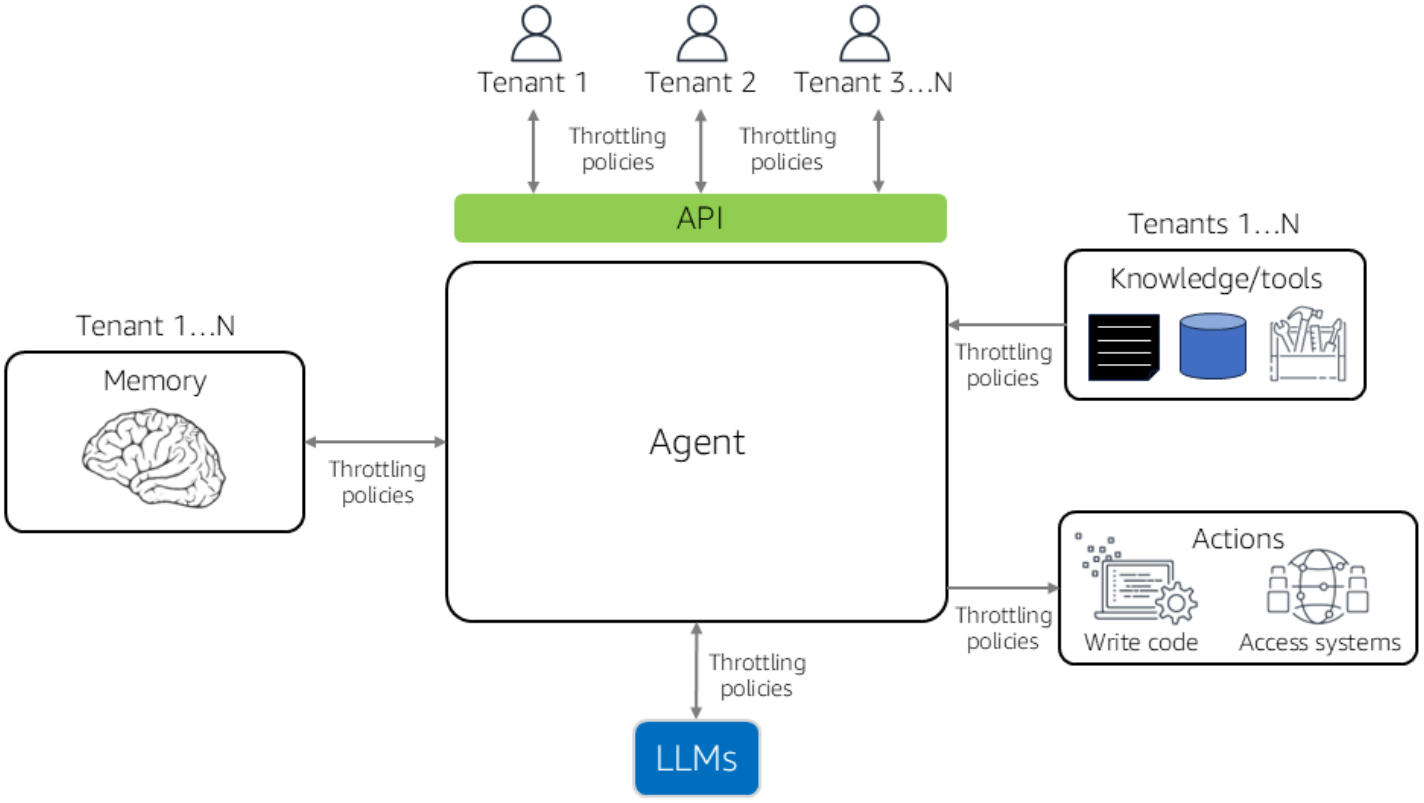

MCP is a standardized protocol that an agent uses to integrate with any tools, data, and resources. In this example, an MCP client and MCP server interact with the tenant-specific knowledge and tools shown on the right side of the diagram. The tenant context flows from client to server, and the server uses this context to acquire tenant-scoped credentials from the AWS Identity and Access Management (IAM) service. The credentials control access to each tenant's resources, ensuring that one tenant can access another tenant's resources.

As agents incorporate multi-tenancy, they must introduce mechanisms that apply tenant isolation policies as they process requests. In some instances, IAM can help limit access to tenant resources. In other instances, you may need to introduce other tools or frameworks to apply tenant isolation policies.

# Noisy neighbor and agents

In a multi-tenant AaaS environment where multiple tenants share an agent, think about where and how to introduce policies that prevent noisy neighbor conditions. Policies can introduce general-purpose throttling that applies to all consumption, or you can have tenant or tier-based policies that apply throttling based on a given persona. You might place greater consumption restrictions on basic-tier tenants than you would on premium-tier tenants.

This notion of throttling can be applied at multiple architecture points. The following diagram shows an example of some possible areas to introduce noisy neighbor policies.




In our prior review of multi-agent implementation, we examined different resources that your agent can use, highlighting the potential for per-tenant resources within an agent. Each touchpoint is a potential area to introduce throttling policies, which helps to ensure that tenants don't exceed the consumption limits of your system or a tenant's tiering policies.

The best places to introduce noisy neighbor protections are at points in the architecture where tenants share resources. These shared or pooled components, such as compute, memory, APIs, and large language models, are the most susceptible to performance degradation if a single tenant consumes disproportionately.

One natural place to apply throttling is at the agent's entry point, sometimes called the "outer edge." Here, you can introduce global or tenant-tier-based rate limits before the agent begins processing the request. Throttling can also be applied deeper in the execution path, such as when the agent calls an LLM, accesses memory, or invokes shared tools.

These policies help you enforce fair usage, maintain agent resilience under load, and preserve a consistent experience across tenants. Depending on your goals, you might focus on general system protection (resilience) or on granularly managing the tenant experience (for example, with tier-based entitlements).

# Data, operations, and testing

## Agents and data ownership

A review of agent implementation highlights scenarios where an agent relies on a given tenant's data. In this instance, consider the data lifecycle and, more importantly, where it's stored. This is especially important for industries and use cases where the data's nature influences how an agent accesses it.

AaaS providers must evaluate how to resolve data issues in a multi-tenant environment, which can affect an agent's onboarding, isolation, and operations. Applicable nuances and strategies vary according to the tools, technologies, and data that you consume. You can approach this in many ways, which is something to be aware of as you create any AaaS offering.

## Multi-tenant agent operations

As you construct agent environments, think about how to operate and manage your agents. As a provider, you need metrics, data, insights, and logs that allow you to monitor an agent's health, scale, and activity. This is more pronounced in a multi-tenant agentic environment where you'll want to understand how individual tenants consume agent resources.

This is even more significant in multi-agent settings when you need insights into agent interactions. Being able to profile and track activities between agents may be essential to troubleshooting issues that affect your system's scale, accuracy, and efficacy.

Operations teams may also profile LLM interactions to derive a better sense of the loads that agents place on LLMs. This data is essential for refining agent implementation. It can also give operational teams a view of how agents and tenancy affect the overall cost profile of a system.

## Training and testing multi-tenant agents

One challenge associated with building agents is that they're expected to learn and evolve. It also means that we must test our agent, refine it, and improve its accuracy in advance of moving it into production. There are many areas where you can inspect and assess if your agent is correctly assessing and categorizing intent or choosing and invoking appropriate tools and actions. The list of variables is extensive, but this is ultimately about ensuring that your agent finds outcomes that achieve your goals.

Examining all the moving parts and principles associated with testing agents is beyond this document's scope but note that testing strategies add complexity to multi-tenant AaaS environments. For example, if an agent has data, memory, and other constructs that are contextually applied to each tenant, then an agent's outcomes can be shaped by per-tenant resources.

If you use an agent to simulate a scenario, you may need to expand your simulation for tenant-specific use cases. Correspondingly, you must refine validation procedures to allow for instances where validation criteria differ for each tenant.

# Considerations and discussion

## Where does SaaS fit?

Industry experts actively debate on how agents influence the software as a service (SaaS) landscape. While it's true that agents are changing software for many systems, it's a stretch to suggest that agents make delivery models obsolete. Some SaaS providers will likely be disrupted by adopting agents, and some may entirely rethink their value proposition, by leaning into an agent as a service (AaaS) model. Others may strike a balance by selectively introducing agents to address specific needs.

This topic is interesting because adopting the best SaaS principles may represent the next evolution of SaaS. This might mean that SaaS is going way, or it might mean that the foundational principles of SaaS are being packaged and realized in an agent-based model. It's probably less important to decide where the terminology ultimately lands, but it seems unlikely that SaaS as a concept will disappear. It's more likely that agents will shape the SaaS footprint.

Ultimately, we must decide which strategies can be applied to AaaS, which means enabling organizations to adopt agentic architectures and business strategies so that providers can maximize the efficiency, value, and impact of their agentic systems. Agents aren't black boxes. Agents consume resources, scale operations, depend on data, and generate costs—all factors that providers must address. Agent providers must evaluate how multi-tenant principles can shape service offerings and optimize operational models.

## Discussion

The agentic landscape continues to evolve with designs varying based on domains, intended use cases, and target industries. Part of this evolution includes further refining our view of strategies, patterns, and tradeoffs that architects consider as they design and build agents.

A comprehensive agent strategy must align with both business and technical objectives. This includes defining target markets and personas, establishing pricing and resource management strategies, and determining how agents fit into larger systems. These considerations are particularly important when delivering AaaS, where scale, cost efficiency, and innovation are primary goals.

Operational capabilities are equally important. The environment must support monitoring of agent activity, health metrics, and usage patterns. This becomes more complex in multi-agent systems, where operations must be coordinated across independent agents.

Overall, this discussion of agents only scratches the surface of the various architectural considerations that could be part of agentic systems. Beyond selecting appropriate tools, frameworks, and LLMs, success depends on creating an architecture that meets business requirements for scalability, efficiency, deployment, and multi-tenancy.

# 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 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 CSPM, 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.

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.

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.