Model Context Protocol strategies on AWS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Model Context Protocol strategies on AWS

Table of Contents

Model Context Protocol strategies on AWS

Amazon Web Services (contributors)

March 2026 (document history)

This guide can help you develop and implement Model Context Protocol (MCP) strategies across your organization to support your agentic AI journey. As agents and language models become increasingly central to business operations, establishing an MCP strategy is critical for successful agentic solutions.

This guide explores three foundational pillars for building an MCP strategy: MCP tool design, MCP server hosting, and MCP governance. By addressing these interconnected components, organizations can create scalable, secure, and effective systems for managing model context across their AI implementations. This guidance provides actionable insights and strategic guidance for organizations at any stage of an organization's AI journey, from initial experimentation to full-scale production deployments. This helps them develop tailored MCP solutions that align with their specific needs and objectives.

These best practices are derived from real-world implementations of organizations deploying MCP at enterprise scale, an analysis of current MCP specification standards, and lessons learned from custom Large Language Model (LLM) applications in production.

AI systems use increasingly sophisticated and robust LLMs in a wide variety of use cases. LLMs excel at understanding natural language, generating human-like responses, and reasoning over complex information. However, to transform LLMs from conversational interfaces into systems that can autonomously accomplish complex tasks, organizations are adopting *agentic AI architectures*, AI systems that can perceive their environment, reason about goals, make autonomous decisions, orchestrate across multiple steps, and take actions to achieve objectives on behalf of users. This agentic approach helps organizations build AI systems that can understand user intent through natural language, autonomously coordinate across multiple data sources and tools, and deliver personalized experiences at a scale that was not possible with traditional request-response patterns. To make these agents more capable, organizations need to provide access to their existing tools and data to enrich the agent's contextual understanding and allow it to act on a user's behalf.

MCP provides a standardized protocol for AI-tool integration, enabling consistent communication between agents and external resources. While MCP itself defines the communication standard,

implementing it effectively requires careful consideration of architectural patterns, security models, operational practices, and performance optimization strategies to achieve scalable, secure, and maintainable solutions.

This guide synthesizes lessons learned from enterprise MCP deployments, providing actionable recommendations that align with the AWS Well-Architected Framework. It covers strategies for MCP tool design, MCP server hosting, and MCP governance, which are essential in building your own MCP solutions. The recommendations in this guide map to the following five pillars of the AWS Well-Architected Framework:

- **Security** – Token isolation, scoped-down credentials, separate read/write authorization
- **Operational Excellence** – Tool selection accuracy metrics, golden datasets for regression testing
- **Reliability** – Per-user and per-tool rate limiting, load shedding
- **Performance efficiency** – Workflow-scoped tools, tool filtering, semantic search to reduce context window usage
- **Cost optimization** – Reusable MCP servers across teams, reduced per-request token costs through tool filtering

Intended audience

This guide is intended for architects, developers, and technology leaders who are implementing agentic AI solutions in their organizations. To understand the concepts in this guide, you should understand how LLMs work and have foundational knowledge about MCP, tools, and prompt engineering.

Objectives

Building Agentic AI systems that are production-ready means solving for governance, optimization, and security together to support your organization's policies. The below explains how this guide addresses these objectives:

- **Governance** – Without centralized governance, you cannot answer audit questions about your AI workloads, including which agents accessed which data, with what permissions, and when. You also cannot enforce versioning. The MCP hosting strategy section of this guide explains how users could be running outdated local MCP servers with known vulnerabilities due to lack of systematic enforcements.

For regulated industries, governance is critical. Auditors want to see policy enforcement and tool usage tracking across all agents from a single pane. MCP governance provides that.

By following the recommendations in this guide, you can improve task accuracy by 28-32% in peer-reviewed benchmarks. For more information, see MARCO: Multi-Agent Real-time Chat Orchestration (ACL Anthology website). Governance is not just about compliance; it also improves your agentic AI system performance.

- **Optimization** – Your teams might build the same integrations more than once. For example, when five different teams write their own database query script for their AI application to communicate with their databases, that's five times the development cost and five sets of bugs list to maintain. MCP lets you build it once and share it across the entire engineering community. The savings compound as your agent count grows.

  There is also a per-request cost problem that most teams don't notice at first. Every tool definition consumes context window tokens. At 20 tools, you're spending 5,000-10,000 tokens per invocation on descriptions alone, alongside the user inquiries. This increases latency and LLM inference costs and degrades accuracy as the model struggles to pick the right tool from the list of available tools.

  Agents that use structured tool wrappers are approximately three times more accurate on database tasks than agents that access APIs directly (for more information, see Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex Environments). How you design and present tools to an AI model is important. This guide recommends giving tools clear schemas, scoping them to actual workflows instead of raw endpoints, and limiting information in the context window. The MCP tool design strategy section of this guide dives deep into these aspects.
- **Security and compliance** – Imagine an agentic AI system that hallucinates a cleanup step and tries to delete a production database. If the agent inherited the user's full admin credentials, the deletion might go through. With token isolation and scoped-down credentials that only grant read and create access, it fails safely.

  Regulated workflows sharpen this further. The guide provides examples (healthcare pipelines that require HIPAA validation and personally identifiable information anonymization before processing patient data). Embedding such logic in MCP tools means compliance happens deterministically every time.

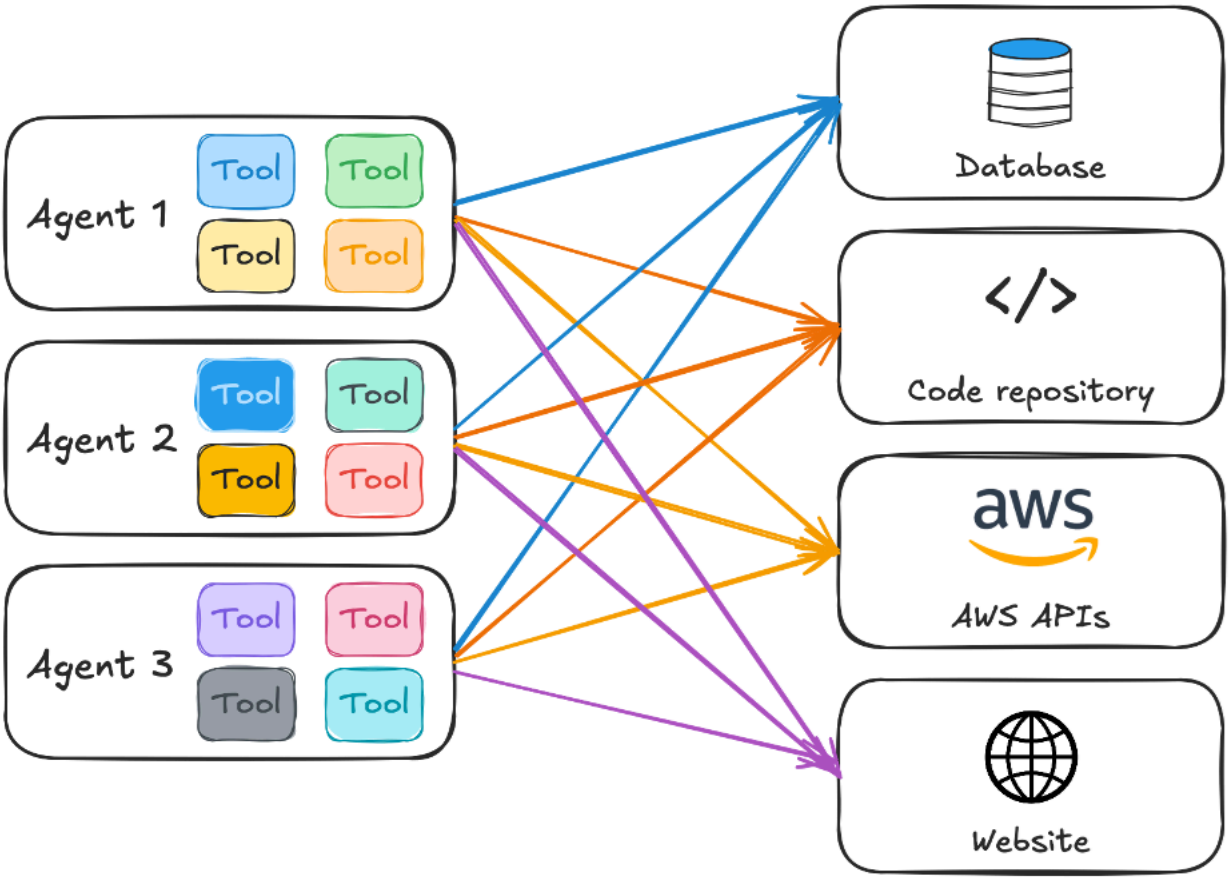
What is MCP?

LLMs work by predicting an answer to a prompt based on their training data. This means that the LLM can only provide answers about data and events it has already seen. Methods like Retrieval Augmented Generation (RAG) and knowledge bases allow you to include contextual data. However, if you were to ask an LLM what tomorrow's weather forecast is going to be or how many customers are in your database, it would likely hallucinate or not be able to provide an answer because these fall outside of the LLM's pre-trained knowledge. To be able to answer these kinds of questions, an agent needs access to external capabilities, data, and APIs outside of the LLM's native context.

Understanding tools

We can give the LLM access to additional systems and context through tools**.** *Tools* are functions given to the LLM to achieve a clear objective. A tool could call an API, query a database, perform calculator operations, operate a code sandbox, perform a web search, and even invoke another AI system or agent-as-a-tool. Each tool should include a description that tells the LLM what the tool does, when to use it, and what parameters it accepts. This enables the LLM to make nuanced decisions about which tool or combination of tools to invoke based on the user's input. The LLM is told about what tools are available to the agent, allowing it to generate responses that instruct the agent to invoke the tool. For example, when you ask the LLM how many customers are in your database, the LLM will send a response back to the agent requesting to run the `query_database` tool with specific input parameters. The LLM determines which tool to invoke and the inputs for the tool call. The agent then executes the tool, which converts the natural language input into a syntactically correct function call and runs the query. The agent invokes the tool or tools based on the instruction from the LLM, and those results are passed back to the LLM. This takes advantage of the LLM's ability to reason over text-based input and select the appropriate tools for the job.

The following image shows how each agent manages its own tool set for each target.



Scaling tool access can present challenges for agentic AI solutions:

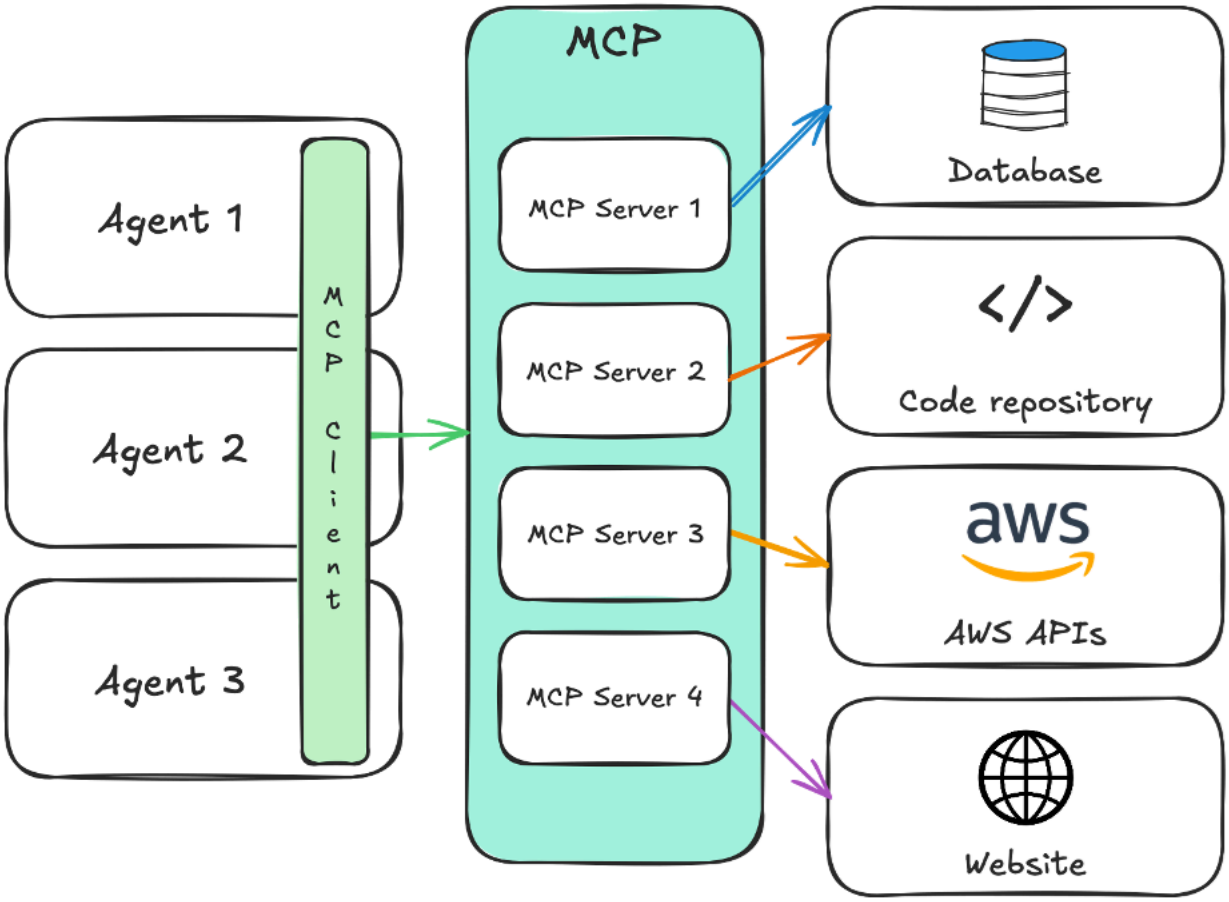
- If every developer is creating their own tool for the same external capabilities, there is a lot of duplicated effort and non-standardized ways of interacting with these external capabilities. This produces inconsistent implementations across your agents. While you could solve that problem by developing standard tools in libraries and distributing them, this lacks centralized governance. This makes it difficult to enforce security policies, track tool usage, manage versioning across teams, or ensure compliance with organizational standards. Additionally, when you embed tools directly with the agent, you must redeploy your agent every time a new tool is created or an existing one is updated.
- Providing tools to an LLM consumes its context window. *The context window* is the number of *tokens* (units of text that LLMs process—typically representing words, parts of words, or punctuation) that a model can consider at any one time. LLMs have a context window limits. Tools and their documentation consume that finite context window along with system prompts and user prompts. As the context window fills, LLMs can experience degraded performance due to multiple factors: difficulty in identifying relevant information, increased processing complexity, and reduced capacity for reasoning. The challenge is compounded when tool

definitions, system prompts, and conversation history compete for limited context window space because they are provided on each LLM invocation.

Thus, the number of tools and how they are documented have direct impact on the LLM's performance, such as response time, and accuracy.

MCP establishes a universal standard for connecting agents to external capabilities. It is commonly referred to as the "USB-C for AI applications." Instead of registering tools directly with agents, MCP servers act as an intermediary for hosting tools that are discovered and invoked over JSON-RPC 2.0. Instead of adding tens or hundreds of different tools to your agent and maintaining them over time, MCP allows you to register MCP servers that encapsulate the tools that your agent can access. This approach standardizes how tools are packaged, presented, and invoked. This can help address the scale and governance challenges of tool usage inside your agents. It also decouples agent development and operations from the tools that it uses for external capabilities.

The following figure shows agents using MCP to access external resources.



However, the MCP standard does not solve all scaling and governance challenges. Implementation of MCP servers must be combined with effective tool design, hosting, and enterprise governance strategies. This guide provides best practices for each strategy to help you build and use MCP as part of your agentic AI solutions.

When to use MCP

MCP provides strategic infrastructure for scaling your agentic AI initiatives. By centralizing tool management and governance, MCP servers reduce the cumulative cost of building and maintaining custom integrations across multiple agents. This delivers increasing returns as your agent ecosystem expands.

MCP likely becomes part of your strategy when:

- You need centralized governance for how agents access enterprise systems and services, such as databases, APIs, internal tools, and third-party integrations.
- Developers spend too much time writing custom integrations that are not consistent across implementations.
- You have duplicated tools that could serve common capabilities.
- You want to offer your proprietary tools or data to external consumers or third-party agentic systems through standardized, governed MCP interfaces, unlocking new revenue streams while maintaining security and control.

After you decide that MCP servers are going to be part of your strategy, evaluate whether existing open-source MCP server implementations meet your needs, whether they require enhancement, or whether you need to build custom servers. Many pre-built MCP server implementations are available in public repositories, and they cover common capabilities such as file system access, web browsing, code sandboxes, database access, and API integrations.

In many cases, pre-existing MCP servers are sufficient. For example, AWS provides the AWS MCP Server, a managed remote MCP server that provides AI assistants and agents with secure, authenticated access to AWS services through natural language interactions. You can use the AWS MCP Server to perform complex, multi-step AWS tasks by combining real-time access to AWS documentation, syntactically correct API calls, and pre-built workflows called Agent SOPs that follow AWS best practices. AWS continuously tests the AWS MCP Server to make sure that customer agents can use them successfully.

You should test these existing MCP servers with your agents to determine if they fulfill your use cases. If an agent fails to complete workflows, generates incorrect or suboptimal responses, fails to navigate complex multi-step processes, or misses important domain-specific best practices or security considerations, you should consider enhancements in several dimensions.

When existing MCP servers do not fully meet your needs and they struggle to use the existing tools correctly or produce accurate responses, consider these enhancement approaches before building custom servers:

- **Enrich agent context** – If your agent struggles to correctly or efficiently use the tools in an existing MCP server, consider supplementing those tool definitions with additional documentation or examples. This helps provide additional context to the LLM.
- **Add complementary tools** – Extend existing MCP servers with tools that access additional organizational data or context that agents need to complete workflows successfully.
- **Improve underlying APIs** – Simplify your service APIs to make them more LLM-friendly by reducing parameter complexity, providing clearer error messages, and offering sensible defaults that agents can use.

While using existing MCP server implementations accelerates development for common capabilities, building custom MCP servers is a necessity when your use case requires specialized functionality. Custom MCP servers help you encapsulate domain expertise, enforce organizational standards, improve agent reliability for complex workflows, and support compliance with security requirements. Consider building a custom MCP server in the following situations:

- **Domain-specific workflows** – Multi-step workflows requiring domain expertise should be encapsulated in custom MCP tools when the necessary knowledge is not captured in API documentation. For example, instead of letting agents orchestrate complex healthcare data pipelines that must validate Health Insurance Portability and Accountability Act (HIPAA) compliance, anonymize PII, and transform to HL7 FHIR format, provide a `process_patient_data` tool that embeds the domain expertise directly. This removes the dependency on the LLM to correctly orchestrate and run the workflow steps, which improves consistency and compliance.
- **Golden path abstractions** – Agents might struggle to implement optimal approaches because they lack organizational context and default to basic patterns rather than organizational best practices. In these scenarios, you can enforce prescriptive standards for cost, performance, or security by encapsulating these golden paths in custom MCP tools. For example, instead of letting agents deploy infrastructure with default settings that might be unsecure or inefficient,

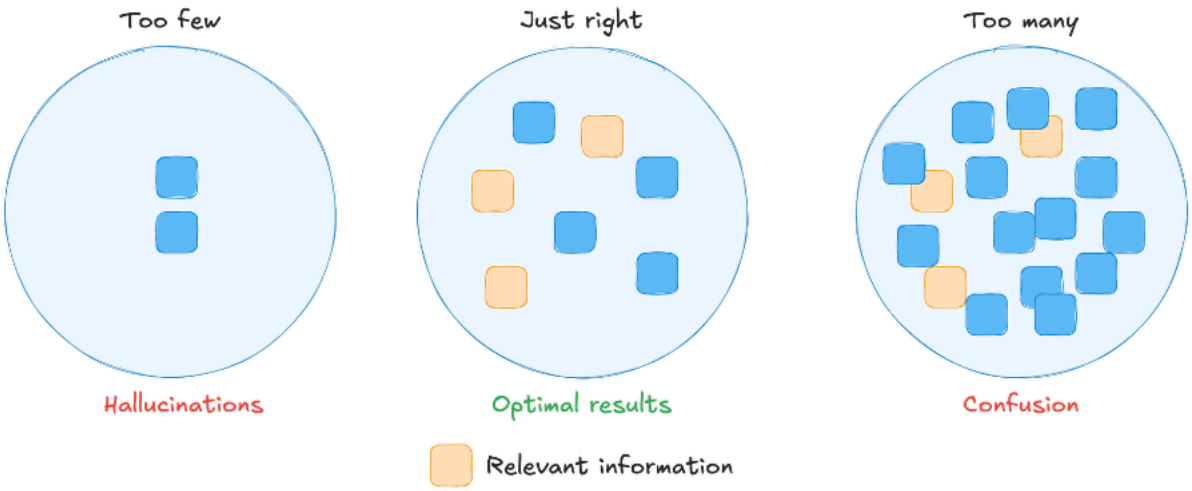
provide a `deploy_secure_infrastructure` tool that directly embeds your organization's standards.

- **Complex multi-service orchestration** – Instead of making the agent orchestrate complex workflows by trying to infer the correct sequence and set of services to use in each step, you can deterministically build that logic inside an MCP tool. You may also want to provide expertise about optimal service integration patterns that the agent might be unaware of. This can also improve the accuracy and efficiency of your agents.
- **Service-specific best practices** – This is common for security-focused tools that help agents implement encryption policies, access controls, and compliance patterns specific to the service being accessed through the agent tool. Additionally, if there are service-specific operational best practices that are not obvious, using an MCP server can help you make sure that they are implemented and not left up to an agent to reason about.

MCP tool design strategy

The main job of the MCP client and server is to discover and present tools to the LLM so that it can use them to improve its responses. This makes MCP tool design one of the most important strategies for building effective MCP solutions. From the model's perspective, tools are a function that they can invoke as needed to provide more accurate and complete responses. The function interface abstracts the underlying implementation of a tool, which can range from a wrapper around a single API call to complex workflow logic.

However, you must strike a balance with the quantity of tools provided to the LLM. If there are too few tools, the LLM might not be able to collect the right context and information, so it will take the best guess with the information available within the model. If there are too many tools, the LLM may get confused about the right tool selection and sequence, leading to hallucinations. Your goal is to get the number of tools just right. The following image shows the challenges of too few and too many tools.



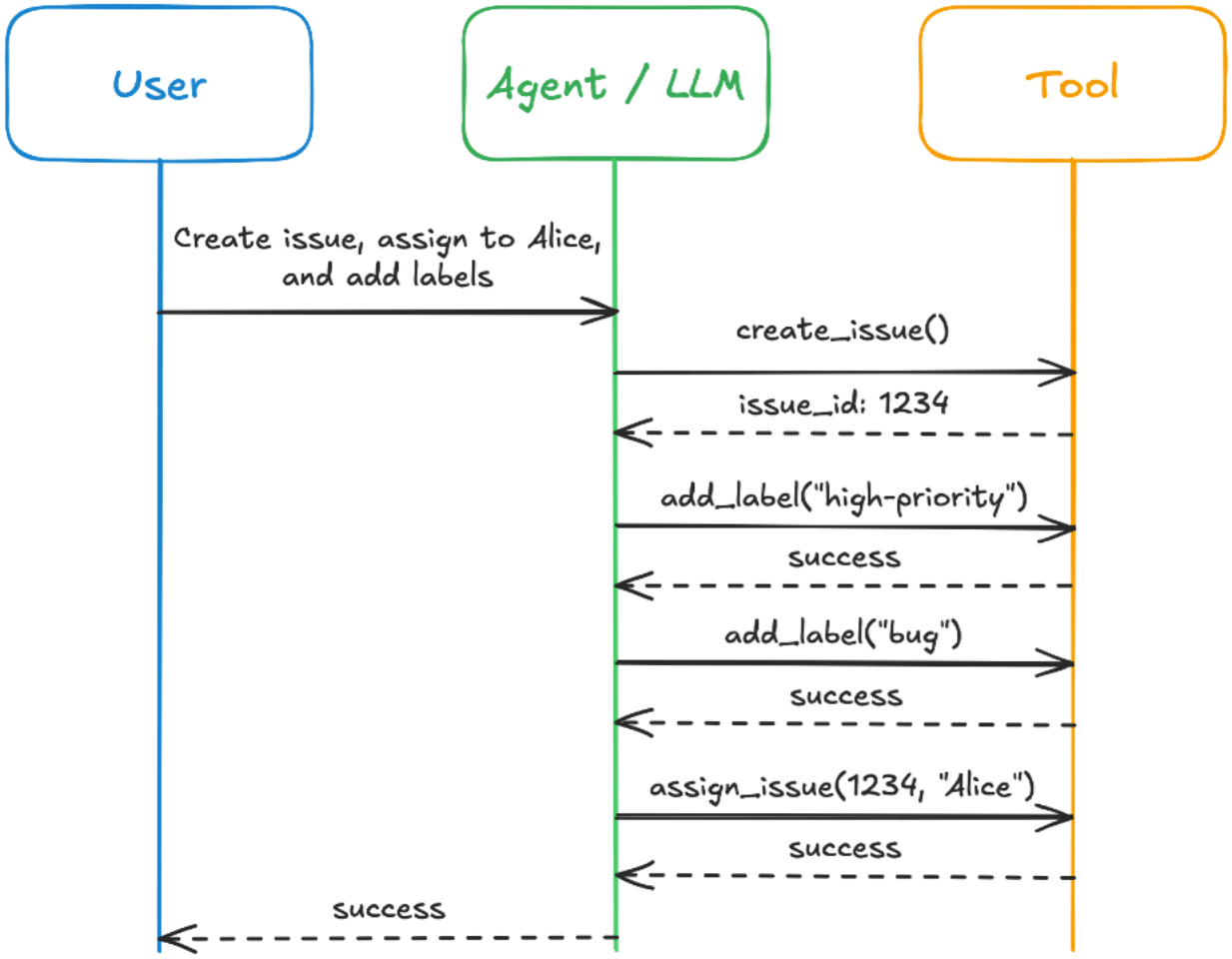
The solution requires understanding how many tools to provide and how to scope each tool. The granularity of your tools, whether they map to individual API calls or complete workflows, directly impacts the total number of tools that agents need and how effectively they can use them. This section provides best practices for scoping MCP tools, creating tool definitions, discovering tools, and organizing them.

Tool scope

There are two approaches for developing tools: granular and coarse-grained.

Granular

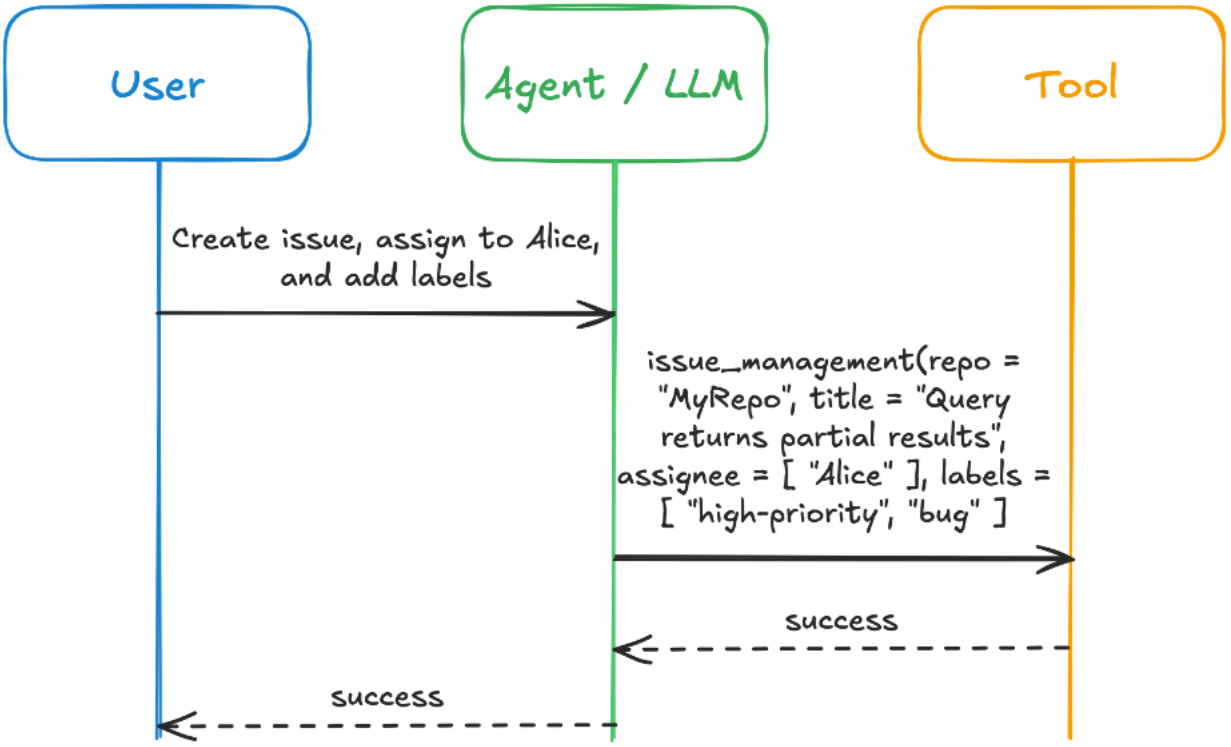
In a granular approach, you would create a tool per API, action, or query. For example, you could create `create_issue`, `get_issue`, `add_label`, `assign_issue`, and `close_issue` tools for your Git repository. This would allow the LLM to make granular calls to each API and orchestrate each one as necessary. Consider the following prompt: "Create an issue for the product service called 'Query only returns partial results', label it as a bug and high-priority, and assign it to Alice." The following image shows how a tool-per-API approach would respond to this prompt.



In this approach the system prompt and every registered tool definition are provided to the LLM on each call. This consumes additional context and incurs a latency penalty because each tool call represents an individual call to the LLM. It also increases the complexity of handling errors within the workflow.

Coarse-grained

A coarse-grained, or *workflow-driven*, approach would be tools that are workflow oriented. The tool focuses on end-to-end user intent over API structure. Instead of a tool-per-API, you have one tool that deterministically calls many APIs. Using the previous Git repository example, you could create a `create_and_setup_issue` tool that is called once by the agent. The tool implementation creates the issue, adds labels, and assigns it to a user, based on the parameters provided to the tool. The following image shows how a coarse-grained approach would process the same prompt.



This approach shows how all complexity remains hidden from the LLM layer. When the orchestration logic is embedded within the tool implementation, all of the sequential steps, logging, retry logic, circuit breakers, and rate limiting are performed deterministically in the tool. The workflow-driven approach makes it simpler for the LLM to invoke the correct tool with the right parameters. It is important to note that some APIs might already provide workflow intent, such as the Amazon EC2 `RunInstances` API. In these cases, a tool-per-API might provide the workflow-oriented design you desire.

However, tools can become too coarse-grained as well. If your single workflow tool attempts to do too many things and has many possible parameters, the LLM can struggle to reason about how to correctly use the tool. It can also create challenges with parameter selection and error handling. Thus, tool development must strike a balance that aligns with user intent and avoids too little or

too much functionality in a single tool. We recommend that you design tools around complete user workflows, bundling operations that commonly occur together (such as three or more API calls). We also recommend that you decompose tools that exceed eight or more parameters or handle multiple distinct user intents. Test with real prompts to verify that agents can correctly use each tool.

If you have complex and dynamic workflows that cannot easily be encapsulated as a deterministic tool, you might consider using the *agent-as-tool pattern*. Instead of your primary agent trying to orchestrate complex tasks in a workflow, a specialized agent can act as a tool. These types of tools can implement advanced decision-making and branching, and they can handle errors and retry logic that cannot be easily managed in deterministic code. This is similar to, but distinct from, the Agent2Agent (A2A) protocol. The A2A protocol is complementary, providing interoperability and collaboration between agents in any agentic framework.

We recommend that you start with your workflow analysis by mapping your most common user workflows to identify the core capabilities each agent needs. This establishes your minimum viable toolset. Based on our experience developing MCP servers at scale, we recommend the following practices. When these practices conflict, prioritize user intent and workflow**.**

Best practices for MCP tool scoping

- **Think in user stories and bundle common operations** – Tools should map directly to complete user interactions rather than requiring orchestration of multiple operations. If workflows commonly require three or more separate calls, combine them into a single tool. This reduces the cognitive load on the LLM, minimizes the number of tool calls, reduces context consumption and latency required to complete tasks, and improves accuracy and latency.
- **Limit parameters to eight or fewer –** If a tool exceeds eight parameters, decompose it into multiple tools. LLMs struggle with parameter selection as complexity increases.

> **Note**
>
> If bundling operations requires more than eight parameters, prioritize bundling over parameter count because simplifying the workflow is more valuable than strict parameter limits.

- **Separate read and write operations –** Provide different tools for reading data and modifying it. This separation makes it explicit when agents are performing potentially destructive operations,

enables different authorization policies, and reduces the risk of unintended modifications during information gathering.

- **Provide sensible defaults** – Design tools so that the LLM needs to specify only the parameters that are specific to the individual request. Defaults reduce parameter complexity and improve tool selection accuracy by minimizing the information that the LLM must reason about.
- **Prefer deterministic execution –** Make tool execution and output deterministic when possible. Deterministic tools are more reliable and easier to test. For complex workflows that require intelligent orchestration, branching logic, or advanced error handling that cannot be easily managed in deterministic code, consider using specialized agents as tools. However, use this pattern selectively because it adds complexity.

Tool definitions

When an LLM receives a request that it cannot handle directly, it will review available tools to help it complete the request. The LLM selects tools based on its semantic understanding of the provided tools' names and descriptions and any instructions provided in the prompt. It will then create input based on the defined input schema and expect output based on the output schema. Therefore, creating descriptive tool definitions and validated input and output schemas is critical in helping the LLM to select tools effectively. There are generally two approaches for creating this documentation: the tool specification approach and the docstring approach.

Tool specification approach

The recommended approach is to directly follow the MCP tool specification when defining the tool. The following example is shown using the Strands Agent tool decorator:

```
@tool(
  name = "search_website",
  description = "This tool searches the provided website for semantic matches to the
 query provided",
  inputSchema = {
    "json": {
      "type": "object",
      "properties": {
        "url": {
          "type": "string",
          "description": "The url of the website to load and search."
        },
        "query": {
```

```
          "type": "string",
          "description": "The content you want to try and match in the website."
      }
    },
    "required": ["url", "query"]
  },
  outputSchema = {
    "json": {
      "type": "object",
      "properties": {
      "results": {
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      }
    }
  }
)
def search_website:
  …
```

Using standard fields, such as `name`, `description`, `inputSchema`, and `outputSchema` makes sure that every tool has consistent documentation that both the LLM and humans can understand. Every tool should define these fields at a minimum and optionally provide a title and annotations, which are optional hints about tool behavior. When possible, use enums for parameter values to make it easy for the LLM to select the correct options. Enums work best for finite sets, such as status or priority values, but are not suitable for free-form text, dynamic values, arbitrary numbers, or resource identifiers. In those cases, provide clear descriptions and examples instead. Also include a default value when possible so the LLM does not have to guess what the correct option is. Keep in mind that tool definitions are included in the LLM prompt on each invocation, consuming context window space alongside system instructions and conversation history.

Docstring approach

Another approach, if you are writing your tools in Python, is using docstrings to provide the tool's description, usage, and output. The following is an example of this approach:

```
def search_website(url: str, query: str) -> list:
```

```
    """
    This tool loads the specified website and then attempts to find content that
matches the provided query through semantic search. It provides back a list of strings
that are the sentences that match the query.
    Args:
      url: the website url to load
      query: the content you want to semantically match in the website
    """
```

Docstrings do not enforce a schema or standardized format. Using this approach might yield inconsistent results based on how tool developers choose to document each tool. Defining and enforcing an organization-wide standard is essential if you follow this approach.

Best practices for MCP tool definitions

- **Follow MCP tool specification** – Provide `name`, `description`, `inputSchema`, and `outputSchema` fields for every tool. For Python implementations, use Pydantic models to provide inline documentation through field descriptions, automatic type validation, and constrained values through enums. This makes schemas self-documenting and improves LLM understanding of valid parameter options.
- **Write descriptions as prompts** – Tool descriptions are instructions that guide LLM decision-making. Include the essential components of the tool's purpose (what the tool does), when to use it (user intent patterns or scenarios), the context of the output (what the output is used for), parameters, and error conditions.
- **Provide concrete examples** – Including workflow examples with actual values is the most effective way to guide LLMs about correct tool usage.
- **Document dependencies explicitly** – Include prerequisites, numbered sequences, state changes, and follow-up actions.

Tool discovery

There are three approaches for discovering and registering tools in your agent with MCP servers: static definition, dynamic discovery, and search function.

Static definition

First, you can statically define the available tools directly in the agent code. In this approach you define a remote tool (a client-side reference object in a framework like Strands Agent SDK) for each

tool provided by the MCP server that is accessed by an MCP client. The following example uses streamable HTTP transport:

```
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp import MCPClient

streamable_http_mcp_client = MCPClient(
    lambda: streamablehttp_client("https://mcp1:8000/mcp")
)

reverse_text = RemoteTool(
    name="reverseText",
    client=streamable_http_mcp_client
)

agent = Agent(tools=[reverse_text])
```

Individual tool registration helps you to be very selective about the tools you make available to the LLM, which minimizes the amount of context window used. The tradeoff is that it requires knowing the names of the available tools and can be brittle if the available tools change in the MCP server.

Dynamic discovery

The next approach is using dynamic discovery and registering all the available tools with the agent. This approach consumes context linearly as more tools are added to the MCP server. The following is an example of this approach:

```
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp import MCPClient

streamable_http_mcp_client = MCPClient(
    lambda: streamablehttp_client("https://mcp1:8000/mcp")
)

with streamable_http_mcp_client:
    tools = streamable_http_mcp_client.list_tools_sync()
    agent = Agent(tools=tools)
```

Consider a scenario where a typical tool definition consumes approximately 250–500 tokens (including name, description, and schema). Registering 20 tools would consume 5,000–10,000 tokens of your context window. When you have a small number of MCP servers and you have control over the number of tools, this option is the simplest to implement. However, if the list of tools is expected to grow, it can create silent context management issues in your agents. An alternative variation of this approach is to use a tool filter parameter when calling `list_tools`, such as what the Strands Agents SDK provides, to reduce the number of tools that are registered with the agent.

Search function

The third option is to use a search function to find relevant tools during runtime. You list all available tools from your MCP server and then perform a semantic search over those tools based on the user prompt. Then, the resulting tools are registered with your agent. Amazon Bedrock AgentCore Gateway provides a native semantic search capability that can make this kind of solution easier to implement.

Best practices for MCP tool discovery

- **Context window preservation** – Pick a tool discovery and registration approach that conserves as much of your context window as possible.
- **Use tool filtering or semantic search capabilities** – Dynamically provide the LLM with a scoped-down set of tools to choose from, which improves its accuracy and effectiveness at choosing the right tool. Tool filtering can operate on tool names (exact matching or patterns), tool descriptions (semantic matching), or domain or category tags. Semantic search is particularly effective for matching user intent against tool descriptions. Both approaches reduce context window use.

Tool organization

Discovering the right tools and ensuring the LLM can use them effectively is one of the most critical parts of effective tool development. As you start to develop MCP servers, you need a strategy that determines:

- How many tools go into one MCP server
- What tools should not be put into the same MCP server

- How to name tools to make them searchable and prevent name collisions (different tools with the same name)
- How to document the tools and MCP server to make them easy to use by the LLM

Namespace organization is a design pattern that prevents tool name collisions, groups related functionality, and facilitates efficient tool identification by LLMs. The pattern establishes structured categorization that is analogous to organized storage systems rather than unstructured accumulation. We recommend the *domain-noun-verb* pattern for tool naming. For example, `github_issue_create`, `github_issue_list`, `github_issue_update`, `github_pullrequest_create`, `github_pullrequest_list`, or `github_pullrequest_merge`. The advantage of this pattern is evident when examining alphabetical sorting behavior. When tools are listed alphabetically, all issue-related operations cluster together (`create`, `list`, `update`), followed by pull request operations (`create`, `list`, `merge`). The noun (resource type) functions as an organizational boundary. This structure facilitates both LLM tool scanning and human documentation navigation because related functionality naturally groups together.

The MCP server should be bounded at the domain level but may be sub-divided based on the separation of duties for the capabilities that it provides. For example, you might have separate MCP servers for write operations and read operations to a database. To enforce this separation, it is recommended that you implement guardrails at the agent level that restrict which MCP servers can be accessed based on user intent and permissions. This can be achieved through a combination of the following:

- **Conditional server loading** – Load the read-only MCP server only when the agent detects read operations in the user input.
- **Permission-based filtering** – Use user authorization to grant access to only appropriate MCP servers.

Finally, you will want to create an upper bound on the number of tools provided by an MCP server. Do not make assumptions about how agents will use your MCP server. They may naively list all the available tools and provide them all to the LLM. If you have more than 50 tools in a single server, you should consider splitting it into multiple servers.

Best practices for MPC tool organization

- **Use the domain-noun-verb naming standard for tools** – Implement strategies to prevent name collisions in both MCP servers and in agents.
- **Set an upper bound** – Restrict the number of tools in a single MCP server.
- **Divide MCP servers** – Use separation of duties to divide MCP servers into logical groups.

MCP hosting strategy

Abstracting the available tools into MCP servers decouples your agent development from the available tools. This introduces the challenges of where you host your MCP server and how tools are organized inside those servers.

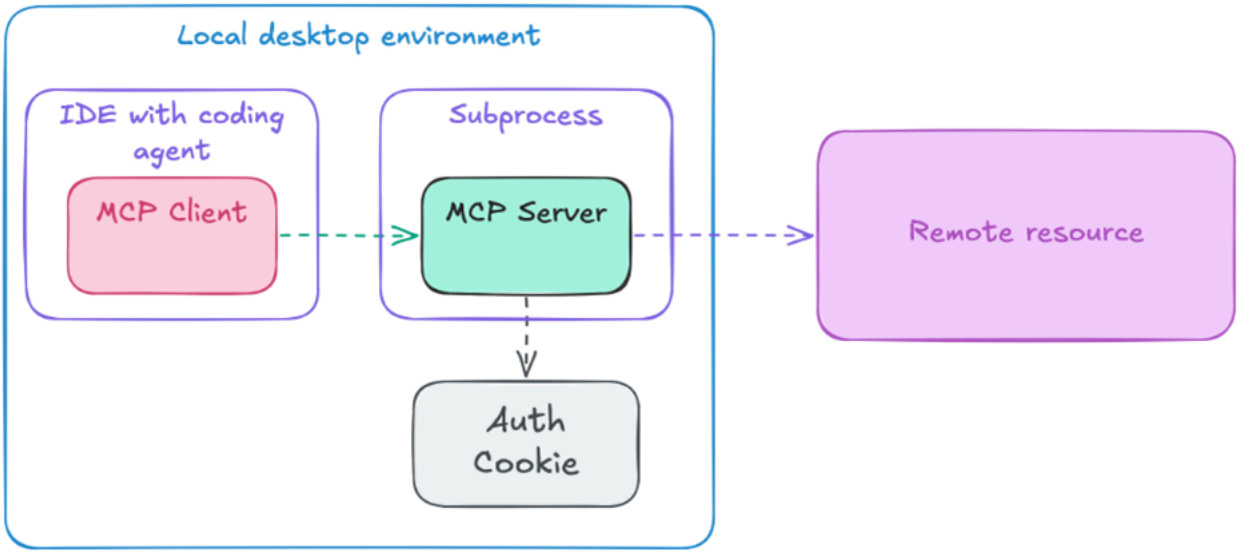
Hosting approaches

There are three options for hosting your MCP servers: running them locally on an end-user machine, hosting them remotely, or hosting them through an MCP gateway. Each option has advantages and tradeoffs.

Local hosting

Local hosting runs the MCP server as a subprocess on your local machine along with the agent that communicates with the server by using JSON-RPC over standard input and output streams. This approach does not require authentication between the client and the server. Tools can interact with local applications and files, use locally stored credentials, and they inherit the network access of the user's local machine. This is the simplest hosting pattern and has several benefits.

Many customers get started with MCP using local servers. They allow engineers to rapidly iterate and solve a variety of problems from their local environment. Consider an MCP server that connects to a Git repository that an engineer's coding assistant us using. Keeping the MCP server local makes a lot of sense because it can use the engineer's unique credentials to access the repository, and it does not add an extra network call to a remote MCP server. The following image shows a locally hosted MCP server being used with a coding agent in an IDE.



For these types of deployments, you must consider how the MCP servers are developed and distributed. Most customers develop an MCP registry where servers can be registered and downloaded by end users. It's very similar to a container registry where a user can search for specific capabilities and find the MCP servers suited to their needs.

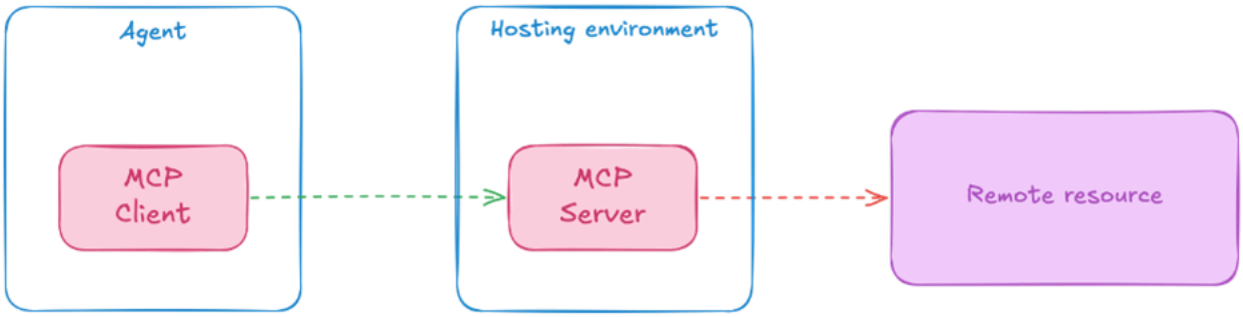
There are public MCP registries, such as Official MCP Registry, and there are privately hosted registries. Organizations typically align their MCP registry strategy with existing policies around open source software distribution, container registries, and internal package management. You should consider factors like security scanning, approval workflows, and compliance requirements.

However, local hosting introduces operational challenges that organizations should consider. First, end-users must discover, download, and configure MCP servers independently. This can add complexity to get started with each individual MCP server they use locally. Second, you cannot control the MCP server lifecycle, meaning that users may continue running outdated versions locally with security vulnerabilities or missing features. This can complicate meeting compliance requirements. Some IDEs and CLI tools, such as Kiro, allow organizations to manage and control which MCP tools are available, ensuring consistency and security across teams.

Remote hosting

The second option is to host remote MCP servers that are accessed over HTTP or HTTPS. This provides access to any network-connected client. Using remote hosting allows you to centrally control access to MCP resources and capabilities, implement authentication and authorization, and control the versioning and updates of the MCP server logic. Remote hosting still requires the use

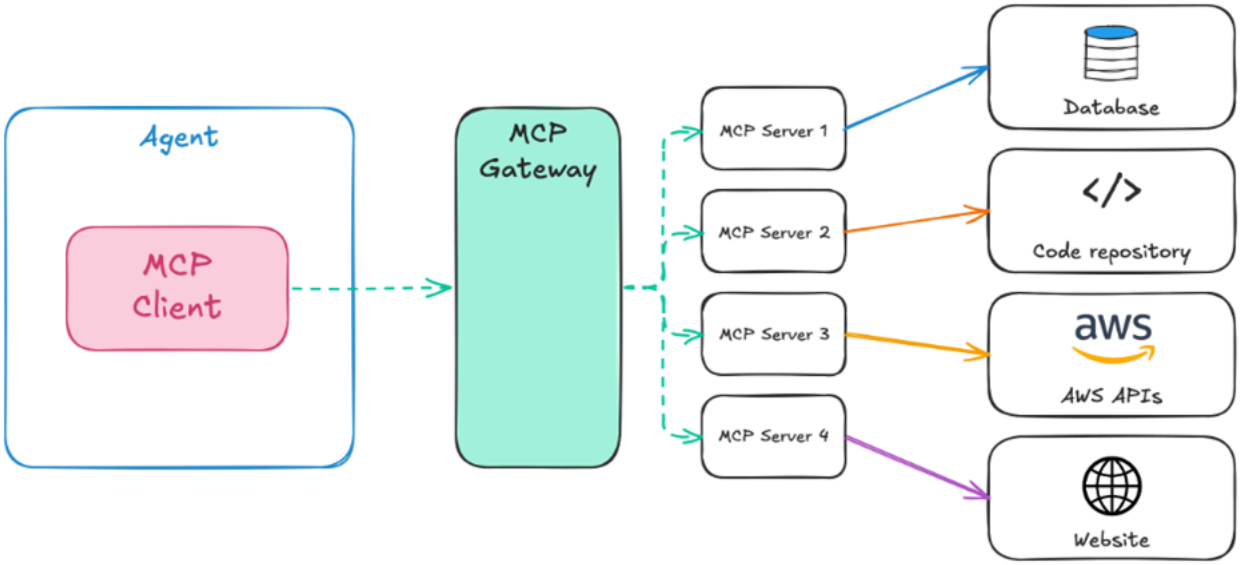
of an MCP registry so that end-users can discover the MCP servers that they want to use with their agent. The following image shows the remote hosting approach.



From an agent development perspective, the experience is similar whether the MCP server is local or remote. The most significant change is implementing authentication and authorization, including both the agent's access to the MCP server and the server's access to external resources. Remote MCP server implementations must be carefully planned to consider multi-tenant access and privilege management. The MCP governance strategy chapter contains more information about authentication and authorization considerations.

MCP gateway

The final option is using an MCP gateway. MCP gateways act as a centralized proxy between MCP clients and servers, and they orchestrate access to the registered MCP servers. Without a gateway, each agent needs to register every remote MCP server that it might want to use. A gateway allows the agent to connect to a single endpoint that manages the authentication, authorization, routing, and protocol translation. New MCP servers and tools can be added dynamically and made immediately available to the agent. The following image shows the MCP gateway approach.



Some gateway solutions, such as Docker MCP Gateway, also manage the lifecycle of the MCP servers, launching servers on-demand as needed. MCP gateways, such as Amazon Bedrock AgentCore Gateway, can also help manage tool discovery by providing native semantic search capabilities. This provides agents with a single endpoint to connect with an MCP client and helps optimize their context window usage. The result is simple agents that can choose and use MCP tools effectively. However, it has similar identity-related challenges as the remote MCP server approach.

Best practices for hosting MCP servers

- The spectrum of hosting options are not a one size fits all. Much of the usage of MCP servers today is local.
- As you start to use remote MCP servers, your main consideration is consistent authentication and authorization to the MCP server and how the MCP server performs authentication and authorization to downstream resources.
- MCP gateways simplify connectivity and authentication and authorization for hosting multiple remote MCP servers. They also provide capabilities to improve context window management by searching for applicable tools.

MCP governance strategy

The other critical capability that MCP offers organizations is support for centralized governance. Your MCP governance strategy should address authentication and authorization to both the MCP servers as well as the resources they access. It should also address rate limiting to protect downstream resources, operational metrics for monitoring tool usage and performance, and managing deployments and distribution of MCP servers.

Authentication and authorization

One of the most important parts of your authentication and authorization strategy is managing downstream resource access from MCP servers. When a user calls an agent, authentication and authorization is performed to ensure the user has permissions to call the agent. Then, the agent orchestrates calling specific tools in MCP servers. You need to decide how to authorize access on a per-tool basis.

One option is *machine-to-machine authorization*, where user consent or interaction is not required. For example, a time-based agent invocation uses an MCP server to collect logs from an application and analyze them. In this scenario, the agent is pre-authorized to access the specified data. The second option is *user-delegated access*, where a user provides their consent to access user-specific data and resources.

The following table shows authentication and authorization patterns.

Factor	User-delegated access	Machine-to-machine
Data ownership	User-specific authorization to data	System or organization-wide data
User interaction	User is present and can consent	No user interaction
Operation timing	Interactive or real-time	Background, scheduled, or batch
Permission scope	Permissions vary by user	Consistent permissions at the agent level

User-delegated access requires careful implementation and should be developed with your security team. Agents must be able to evaluate which tools an LLM has selected and whether they require additional authorization. MCP tools must include descriptions to indicate their authentication and authorization requirements and where to retrieve access tokens. Client applications must support intermediate authentication requests, and the MCP client must provide the retrieved credentials back to the agent for each tool call.

You should ensure that MCP tools always have their own tokens to access external capabilities and that the access is logged and audited. User credentials should not be propagated through your agentic system. For example, your MCP servers should not use the same token to access data that was used to invoke the agent. Downstream calls should use explicitly scoped, purpose-generated tokens. This helps provide additional guardrails to prevent unintended data access on-behalf of actions. It can also help prevent hallucinations from producing unintended results. Imagine that a user with full admin permissions asks an agent to clone a production database for use in pre-production. To do so, the user only needs `READ` and `CREATE` permissions. Let's say the LLM hallucinates and believes it needs to clean up the old database as part of this request. If it reuses the user's credentials, it would likely succeed because the user's original credentials have `DELETE` permissions. Instead, if the MCP server uses an intentionally scoped-down token for the request with just `READ` and `CREATE` permissions, the attempt to delete the production database would fail.

You can use Amazon Bedrock AgentCore Identity to help implement these patterns. Make sure that you make an intentional choice about whether the permissions to list and invoke tools hosted by an MCP server implies permission to the external capabilities that the MCP server exposes. This identity flow from the MCP server to the resource and back to the user is dependent on the type of authentication and authorization service being used. You must decide how this is handled at scale for your MCP servers.

When designing your authentication and authorization patterns, implement token isolation mechanisms that retrieve different access tokens for each tool being accessed. Do not reuse tokens between tools and servers. AgentCore Identity provides this token isolation capability. It automatically manages both workload tokens (for machine-to-machine authentication) and user tokens (for user-delegated access) to ensure proper separation and prevent permission escalation. This is especially critical when incorporating remote MCP servers or MCP gateways.

Best practices for MCP authentication and authorization

- **Token separation** – Do not pass bearer tokens from callers to downstream services. Validate the aud (audience) field matches the server receiving the token. The audience claim specifies

which service the token is intended for, preventing unauthorized token reuse across different MCP servers.

- **Select an access approach** – Choose between machine-to-machine and user-delegated access for each tool your MCP servers provide. Consider grouping tools together in the same MCP server that use the same authentication pattern.

Controlling load

As with any distributed system, you must consider how to control load in your MCP server fleet. First, you consider whether to implement rate limiting in your MCP servers and where to implement the limits. If you choose not to implement rate limiting, you pass on any rate limiting performed by downstream resources. Many systems choose to rate limit based on request attributes, such as a user or account ID. Validate that the requests sent to downstream services carry on those same attributes so that multiple users are not affected by load being driven by another user.

If you do choose to implement rate limiting, the recommended approach is to implement primary rate limiting at the MCP server level, with backend services providing secondary protection and agents adapting their behavior based on rate limit feedback. Consider whether the rate limits are per-MCP server or per-tool. Per-MCP server rate limits help protect your MCP server fleet and services in a multi-tenant environment. However, that can be very coarse-grained. Per-tool rate limits are designed to prevent overwhelming downstream resources that might not sufficiently rate limit themselves. If a tool calls multiple APIs, you should set the rate limit to align to the lowest rate allowed by those APIs.

Passing rate limit information in HTTP headers can also be a useful metric for users and automated systems to help manage their own request rate and retry strategy. For example, you might send these headers back to the agent from your MCP server, as shown in the following example:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 45
X-RateLimit-Reset: 1640995200
```

Additionally, consider load shedding to protect the overall service when no single customer is exceeding a rate limit but the load is impacting the system's performance.

Best practices for controlling load

- **Choose a rate-limiting approach** – Plan to rate limit individual users based on either their use of downstream resources or through their use of your MCP server and tools.
- **Consider load shedding** – Protect your MCP server fleet from general overload that is not driven by a single or handful of customers.

Operational metrics

Key metrics to capture for MCP implementations should focus on the customer experience they deliver. These metrics commonly include token usage, tool selection accuracy, number of tools registered with the agent, and tool latency. For example, monitoring output tokens returned by each tool enables you to set alarms when tools exceed a threshold for context window usage. When a tool exceeds that threshold, you might want to review the tool's behavior. This ties into the MCP tool design strategy as well. Tool selection accuracy metrics indicate how well agents choose appropriate tools for given tasks, while execution speed and success rates highlight performance bottlenecks and reliability issues.

For example, to evaluate the tool-selection and tool-use accuracy metrics, AWS teams created golden datasets for regression testing. The datasets were generated synthetically by using LLMs from historical API invocation logs upon user queries. Using the pre-defined tool-selection and tool-use metrics (such as tool selection accuracy, tool parameter accuracy, and multi-turn function call accuracy), the AWS teams could objectively evaluate the AI agent's ability to correctly identify the appropriate tools, populate their parameters with accurate values, and maintain coherent tool invocation sequences across conversational turns.

Measuring metrics about the number of tools registered with an agent can help you identify potential context window management challenges as well as changes in the available tools presented by MCP servers. You should regularly review operational metrics that indicate the user experience with your MCP server and tools.

Contributors

Authoring

- Alex Torres, Sr. Solutions Architect, AWS
- Saikat Gomes, Sr. Customer Solutions Manager, AWS
- Mike Haken, Sr. Principal Solutions Architect, AWS
- Sreeja Das, Principal Engineer, AWS

Reviewing

- Ted Swinyar, Solutions Architect Manager, AWS
- Raju Patil, Senior Data Scientist, AWS

Technical writing

- Lilly AbouHarb, Senior Technical Writer, AWS

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	—	March 16, 2026

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

A2A (Agent-to-Agent)

A stateful protocol for agent-to-agent collaboration supporting task delegation and state transfer.

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.

Agent

An AI system that can autonomously reason, plan, and take actions using tools to achieve goals.

Agent Ops

Operational practices for building, testing, deploying, and running AI agents in production at scale.

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.

Citizen Developer

A business user who creates AI applications using no-code/low-code platforms without specialized technical skills.

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.

FM gateway

A centralized intermediary that controls and normalizes access to foundation models. Also known as an *LLM gateway*.

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.

guardrails (AI)

Safety mechanisms that filter, validate, and constrain agent inputs and outputs to help ensure responsible and safe AI behavior.

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.

human-in-the-loop (HitL)

A workflow pattern where agent execution pauses for human review and approval at critical decision points.

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.

MCP

See Model Context Protocol.

Model Context Protocol (MCP)

A stateless protocol for agent-to-tool communication.

MCP server

A service that exposes one or more tools through the Model Context Protocol.

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.

Shadow AI

Unauthorized AI applications built or used outside of governed channels within an organization.

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.

tool

A function or API that an agent can invoke to perform operations in external systems.

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.