AWS Open Source Blog

Introducing Strands Agents, an Open Source AI Agents SDK

by Clare Liguori | on 16 MAY 2025 | in Announcements, Artificial Intelligence, Generative AI, Open Source |
Permalink | Comments | Share

Today I am happy to announce we are releasing Strands Agents. Strands Agents is an open source SDK that takes a model-driven approach to building and running AI agents in just a few lines of code. Strands scales from simple to complex agent use cases, and from local development to deployment in production. Multiple teams at AWS already use Strands for their AI agents in production, including Amazon Q Developer, AWS Glue, and VPC Reachability Analyzer. Now, I'm thrilled to share Strands with you for building your own AI agents.

Compared with frameworks that require developers to define complex workflows for their agents, Strands simplifies agent development by embracing the capabilities of state-of-the-art models to plan, chain thoughts, call tools, and reflect. With Strands, developers can simply define a prompt and a list of tools in code to build an agent, then test it locally and deploy it to the cloud. Like the two strands of DNA, Strands connects two core pieces of the agent together: the model and the tools. Strands plans the agent's next steps and executes tools using the advanced reasoning capabilities of models. For more complex agent use cases, developers can customize their agent's behavior in Strands. For example, you can specify how tools are selected, customize how context is managed, choose where session state and memory are stored, and build multi-agent applications. Strands can run anywhere and can support any model with reasoning and tool use capabilities, including models in Amazon Bedrock, Anthropic, Ollama, Meta, and other providers through LiteLLM.

Strands Agents is an open community, and we're excited that several companies are joining us with support and contributions including Accenture, Anthropic, Langfuse, mem0.ai, Meta, PwC, Ragas.io, and Tavily. For instance, Anthropic has already contributed support in Strands for using models through the Anthropic API, and Meta contributed support for Llama models through Llama API. Join us on GitHub to get started with Strands Agents!

Our journey building agents

I primarily work on Amazon Q Developer, a generative AI-powered assistant for software development. My team and I started building AI agents in early 2023, around when the original ReAct (Reasoning and Acting) scientific paper was published. This paper showed that large language models could reason, plan, and take actions in their environment. For example, LLMs could reason that they needed to make an API call to complete a task and then generate the inputs needed for that API call. We then realized that large language models could be used as agents to complete many types of tasks, including complex software development and operational troubleshooting.

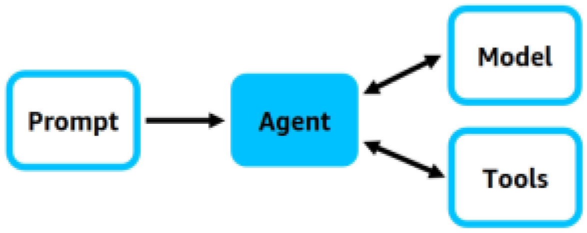
At that time, LLMs weren't typically trained to act like agents. They were often trained primarily for natural language conversation. Successfully using an LLM to reason and act required complex prompt instructions on how to use tools, parsers for the model's responses, and orchestration logic. Simply getting LLMs to reliably produce syntactically correct JSON was a challenge at the time! To prototype and deploy agents, my team and I relied on a variety of complex agent framework libraries that handled the scaffolding and orchestration needed for the agents to reliably succeed at their tasks with these earlier models. Even with these frameworks, it would take us months of tuning and tweaking to get an agent ready for production.

Since then, we've seen a dramatic improvement in large language models' abilities to reason and use tools to complete tasks. We realized that we no longer needed such complex orchestration to build agents, because models now have native tool-use and reasoning capabilities. In fact, some of the agent framework libraries we had been using to build our agents started to get in our way of fully leveraging the capabilities of newer LLMs. Even though LLMs were getting dramatically better, those improvements didn't mean we could build and iterate on agents any faster with the frameworks we were using. It still took us months to make an agent production-ready.

We started building Strands Agents to remove this complexity for our teams in Q Developer. We found that relying on the latest models' capabilities to drive agents significantly reduced our time to market and improved the end user experience, compared to building agents with complex orchestration logic. Where it used to take months for Q Developer teams to go from prototype to production with a new agent, we're now able to ship new agents in days and weeks with Strands.

Core concepts of Strands Agents

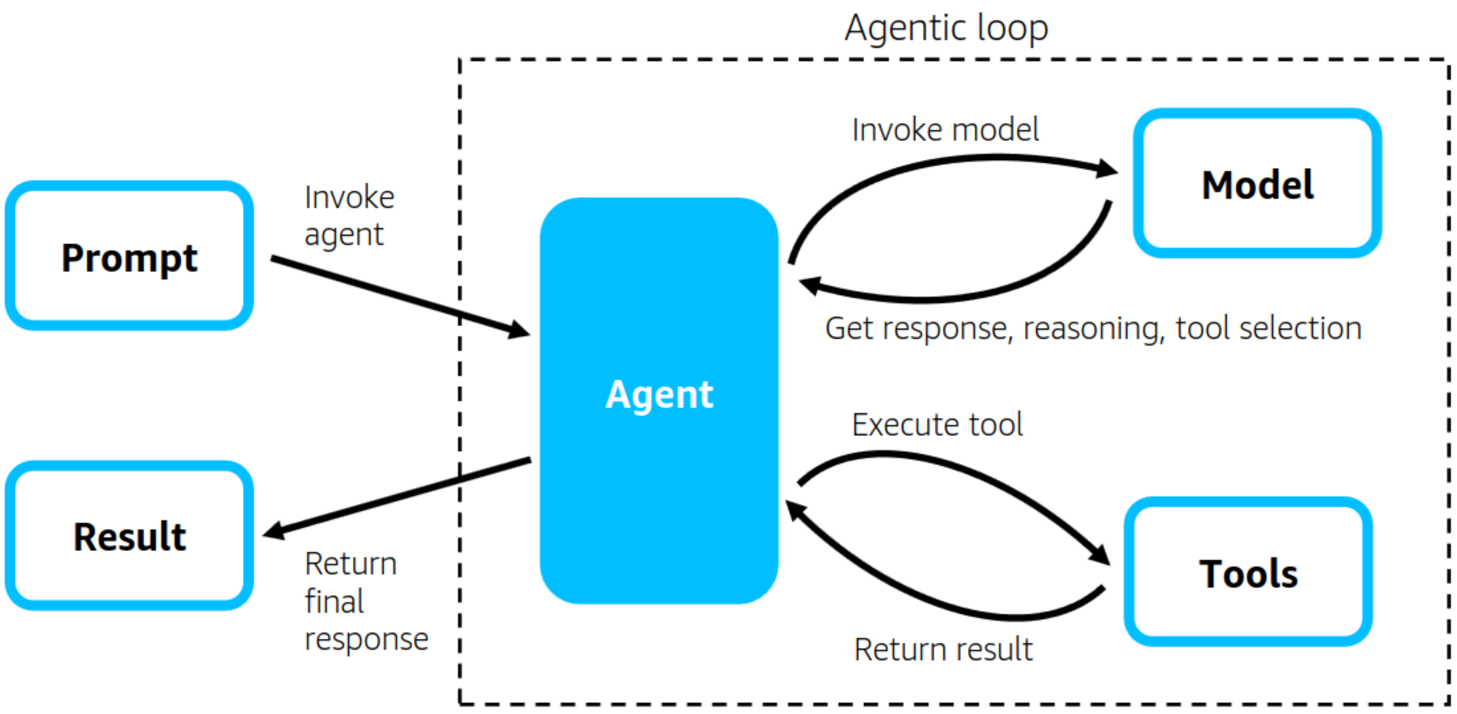
The simplest definition of an agent is a combination of three things: 1) a model, 2) tools, and 3) a prompt. The agent uses these three components to complete a task, often autonomously. The agent's task could be to answer a question, generate code, plan a vacation, or optimize your financial portfolio. In a model-driven approach, the agent uses the model to dynamically direct its own steps and to use tools in order to accomplish the specified task.



To define an agent with the Strands Agents SDK, you define these three components in code:

1. **Model:** Strands offers flexible model support. You can use any model in Amazon Bedrock that supports tool use and streaming, a model from Anthropic's Claude model family through the Anthropic API, a model from the Llama model family via Llama API, Ollama for local development, and many other model providers such as OpenAI through LiteLLM. You can additionally define your own custom model provider with Strands.
2. **Tools:** You can choose from thousands of published Model Context Protocol (MCP) servers to use as tools for your agent. Strands also provides 20+ pre-built example tools, including tools for manipulating files, making API requests, and interacting with AWS APIs. You can easily use any Python function as a tool, by simply using the Strands `@tool` decorator.
3. **Prompt:** You provide a natural language prompt that defines the task for your agent, such as answering a question from an end user. You can also provide a system prompt that provides general instructions and desired behavior for the agent.

An agent interacts with its model and tools in a loop until it completes the task provided by the prompt. This agentic loop is at the core of Strands' capabilities. The Strands agentic loop takes full advantage of how powerful LLMs have become and how well they can natively reason, plan, and select tools. In each loop, Strands invokes the LLM with the prompt and agent context, along with a description of your agent's tools. The LLM can choose to respond in natural language for the agent's end user, plan out a series of steps, reflect on the agent's previous steps, and/or select one or more tools to use. When the LLM selects a tool, Strands takes care of executing the tool and providing the result back to the LLM. When the LLM completes its task, Strands returns the agent's final result.



In Strands' model-driven approach, tools are key to how you customize the behavior of your agents. For example, tools can retrieve relevant documents from a knowledge base, call APIs, run Python logic, or just simply return a static string that contains additional model instructions. Tools also help you achieve complex use cases in a model-driven approach, such as with these Strands Agents example pre-built tools:

- **Retrieve tool**: This tool implements semantic search using Amazon Bedrock Knowledge Bases. Beyond retrieving documents, the retrieve tool can also help the model plan and reason by retrieving other tools using semantic search. For example, one internal agent at AWS has over 6,000 tools to select from! Models today aren't capable of accurately selecting from quite that many tools. Instead of describing all 6,000 tools to the model, the agent uses semantic search to find the most relevant tools for the current task and describes only those tools to the model. You can implement this pattern by storing many tool descriptions in a knowledge base and letting the model use the retrieve tool to retrieve a subset of relevant tools for the current task.
- **Thinking tool**: This tool prompts the model to do deep analytical thinking through multiple cycles, enabling sophisticated thought processing and self-reflection as part of the agent. In the model-driven approach, modeling thinking as a tool enables the model to reason about if and when a task needs deep analysis.
- **Multi-agent tools** like the workflow, graph, and swarm tools: For complex tasks, Strands can orchestrate across multiple agents in a variety of multi-agent collaboration patterns. By modeling sub-agents and multi-agent collaboration as tools, the model-driven approach enables the model to reason about if and when a task requires a defined workflow, graph, or swarm of sub-agents. Strands support for the Agent2Agent (A2A) protocol for multi-agent applications is coming soon.

Get started with Strands Agents

Let's walk through an example of building an agent with the Strands Agents SDK. As has long been said, naming things is one of the hardest problems in computer science. Naming an open source project is no exception! To help us brainstorm potential names for the Strands Agents project, I built a naming AI assistant using Strands. In this example, you will use Strands to build a naming agent using a default model in Amazon Bedrock, an MCP server, and a pre-built Strands tool.

Create a file named `agent.py` with this code:

```
from strands import Agent
from strands.tools.mcp import MCPClient
from strands_tools import http_request
from mcp import stdio_client, StdioServerParameters

# Define a naming-focused system prompt
NAMING_SYSTEM_PROMPT = """
You are an assistant that helps to name open source projects.

When providing open source project name suggestions, always provide
one or more available domain names and one or more available GitHub
organization names that could be used for the project.

Before providing your suggestions, use your tools to validate
that the domain names are not already registered and that the GitHub
organization names are not already used.
"""


# Load an MCP server that can determine if a domain name is available
```

You will need a GitHub personal access token to run the agent. Set the environment variable `GITHUB_TOKEN` with the value of your GitHub token. You will also need Bedrock model access for Anthropic Claude 3.7 Sonnet in us-west-2, and AWS credentials configured locally.

Now run your agent:

```
pip install strands-agents strands-agents-tools
python -u agent.py
```

You should see output from the agent similar to this snippet:

```
Based on my checks, here are some name suggestions for your
open source AI agent building project:

## Project Name Suggestions:

1. **Strands Agents**
- Available domain: strandsagents.com
- Available GitHub organization: strands-agents
```

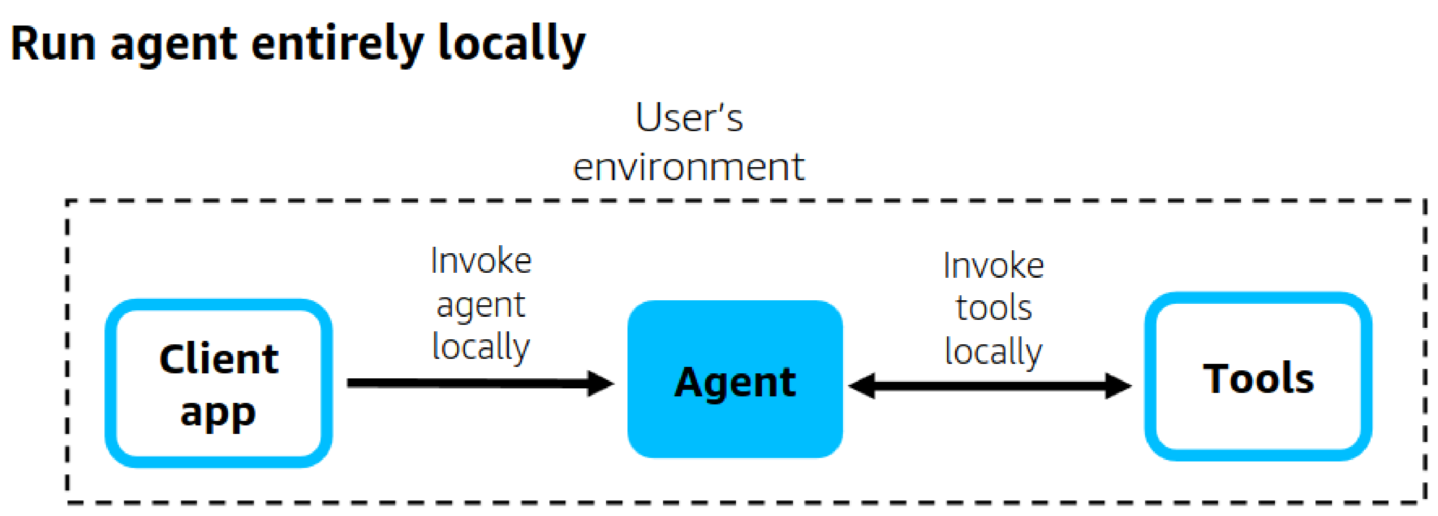
You can easily start building new agents today with the Strands Agents SDK in your favorite AI-assisted development tool. To help you quickly get started, we published a Strands MCP server to use with any MCP-enabled development tool, such as the Q Developer CLI or Cline. For the Q Developer CLI, use the following example to add the Strands MCP server to the CLI's MCP configuration. You can see more configuration examples on GitHub.

```
{
  "mcpServers": {
    "strands": {
      "command": "uvx",
      "args": ["strands-agents-mcp-server"]
    }
  }
}
```

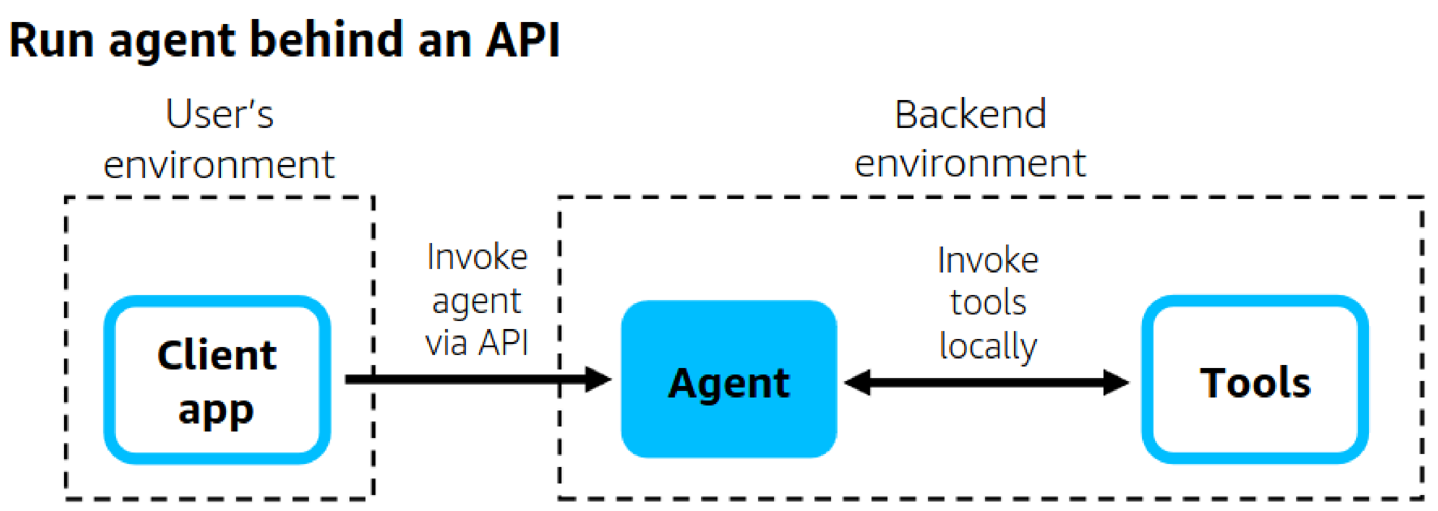
Deploy Strands Agents in production

Running agents in production is a key tenet for the design of Strands. The Strands Agents project includes a deployment toolkit with a set of reference implementations to help you take your agents to production. Strands is flexible enough to support a variety of architectures in production. You can use Strands to build conversational agents as well as agents that are triggered by events, run on a schedule, or run continuously. You can deploy an agent built with the Strands Agents SDK as a monolith, where both the agentic loop and the tool execution run in the same environment, or as a set of microservices. I will describe four agent architectures that we use internally at AWS with Strands Agents.

The following diagram shows an agent architecture with Strands running entirely locally in a user's environment through a client application. The example command line tool on GitHub follows this architecture for a CLI-based AI assistant for building agents.

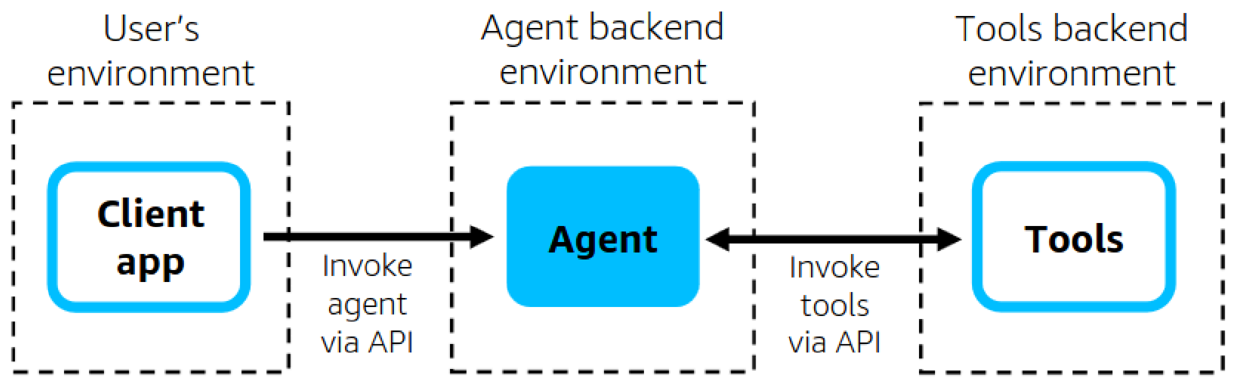


The next diagram shows an architecture where the agent and its tools are deployed behind an API in production. We have provided reference implementations on GitHub for how to deploy agents built with the Strands Agents SDK behind an API on AWS, using AWS Lambda, AWS Fargate, or Amazon Elastic Compute Cloud (Amazon EC2).



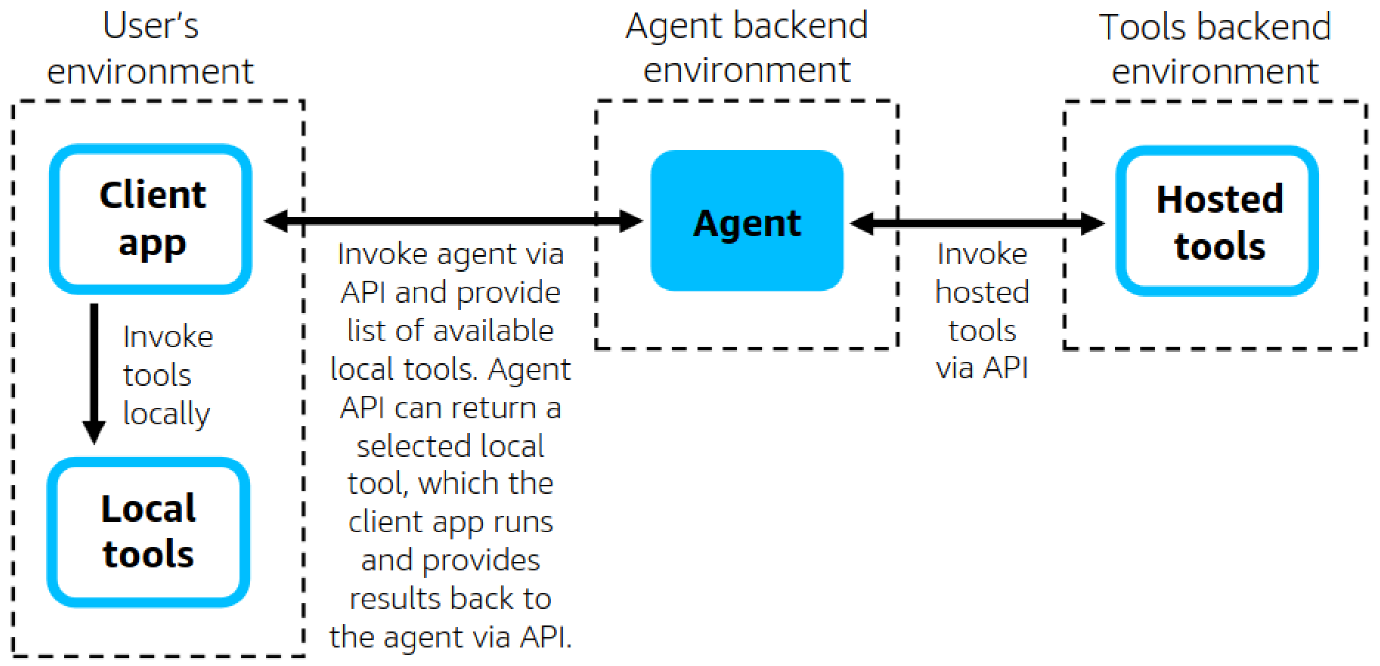
You can separate concerns between the Strands agentic loop and tool execution by running them in separate environments. The following diagram shows an agent architecture with Strands where the agent invokes its tools via API, and the tools run in an isolated backend environment separate from the agent's environment. For example, you could run your agent's tools in Lambda functions, while running the agent itself in a Fargate container.

Run tools in an isolated backend environment



You can also implement a return-of-control pattern with Strands, where the client is responsible for running tools. This diagram shows an agent architecture where an agent built with the Strands Agents SDK can use a mix of tools that are hosted in a backend environment and tools that run locally through a client application that invokes the agent.

Run a mix of tools in the user's environment and in the backend


User's environment
Agent backend environment
Tools backend environment
Client app
Invoke agent via API and provide list of available local tools. Agent API can return a selected local tool, which the client app runs and provides results back to the agent via API.
Agent
Invoke hosted tools via API
Hosted tools
Invoke tools locally
Local tools

Regardless of your exact architecture, observability of your agents is important for understanding how your agents are performing in production. Strands provides instrumentation for collecting agent trajectories and metrics from production agents. Strands uses OpenTelemetry (OTEL) to emit telemetry data to any OTEL-compatible backend for visualization, troubleshooting, and evaluation. Strands' support for distributed tracing enables you to track requests through different components in your architecture, in order to paint a complete picture of agent sessions.

Join the Strands Agents community

Strands Agents is an open source project licensed under the Apache License 2.0. We are excited to now build Strands in the open with you. We welcome contributions to the project, including adding support for additional providers' models and tools, collaborating on new features, or expanding the documentation. If you find a bug, have a suggestion, or have something to contribute, join us [on GitHub].

To learn more about Strands Agents and to start building your first AI agent with Strands, visit [our documentation.]

Clare Liguori

Clare Liguori is a Senior Principal Software Engineer for AWS Agentic AI. She focuses on re-imagining how applications are built and how productive developers can be when their tools are powered by generative AI and AI agents, as part of Amazon Q Developer.

Like (29)   Share

104 Comments

Log in to comment

Sort by: **Top**

F **Fahmiimimed** Aug 27, 2025

Very good

Like (1)   Reply   Share   ...

Z **zilong** Jul 20, 2025

Completed

Like (1)   Reply   Share   ...

SR **Steven Romero** Jun 3, 2025

A lot of new offerings, when do we use Strands? When do we use Bedrock Agents? When do we use Bedrock Flows? What makes Strands different from PydanticAI?

Like (1) Reply (2) Share ...

O **Omar** Nov 6, 2025

Bedrock Flows is how you coordinates your Bedrock Flows. Whether you use Strands or other technologies will depend on your architecture and needs. For example, some services like Bedrock Knowledge bases are not available in certain regions. Industries that don't want to take their workloads outside their region can benefit from having alternatives to Bedrock.

But to answer generally, Bedrock and Bedrock Agentcore are great for having a centralized, streamlined development process with minimized integration overhead. But if these services aren't available to your particular use case, alternatives are great!

Like (1) Reply Share ...