

AWS Bedrock Agentcore

Architectural Paradigms and Implementation Strategies in AWS
Bedrock Agent Core: An Enterprise-Grade Analysis





Executive Summary

This documentation presents a comprehensive implementation of advanced agent capabilities using Amazon Bedrock AgentCore to design, build, and operate secure, scalable, and production-ready AI agents.

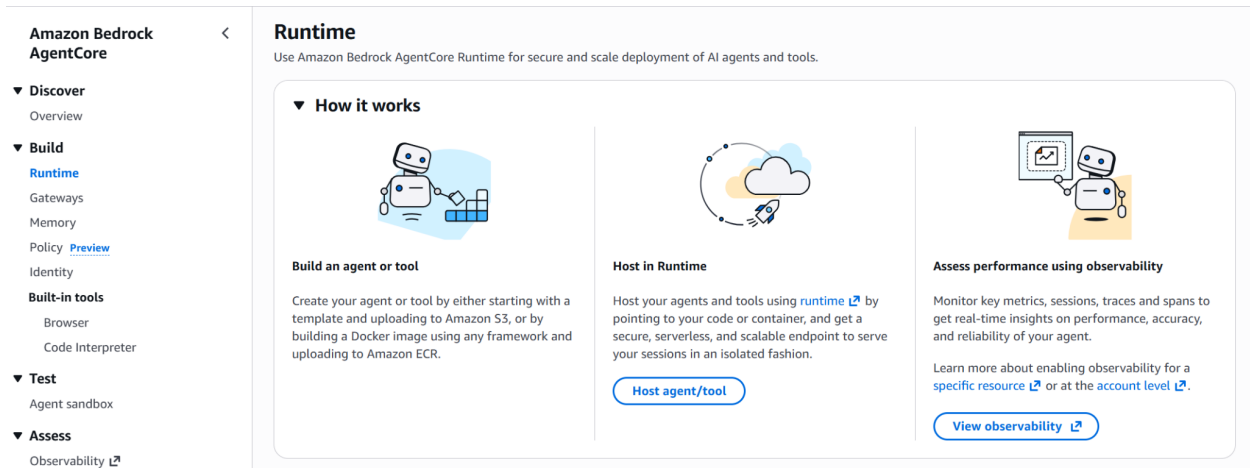
- Agent Core simplifies agent development compared to direct model API usage

1. What is Bedrock Agent Core

Amazon Bedrock AgentCore is an advanced agent orchestration framework that enables AI agents to securely interact with tools and data using fine-grained permissions and built-in governance

1.1 AgentCore Runtime

Serverless environment that provides complete session isolation and automatically handles Docker packaging, ECR registry management, and scaling.



The screenshot shows the Amazon Bedrock AgentCore Runtime console. On the left is a navigation menu with sections: Discover (Overview), Build (Runtime, Gateways, Memory, Policy [Preview](#), Identity), Built-in tools (Browser, Code Interpreter), Test (Agent sandbox), and Assess (Observability [↗](#)). The main content area is titled 'Runtime' with the subtitle 'Use Amazon Bedrock AgentCore Runtime for secure and scale deployment of AI agents and tools.' Below this is a 'How it works' section with three columns: 1. 'Build an agent or tool' with an icon of a robot and blocks, text about creating agents from templates or Docker images, and an upload icon. 2. 'Host in Runtime' with an icon of a cloud and a rocket, text about hosting agents using runtime [↗](#), and a 'Host agent/tool' button. 3. 'Assess performance using observability' with an icon of a robot and a chart, text about monitoring metrics and enabling observability, and a 'View observability [↗](#)' button.

- Secure state-**decoupled runtime** with IAM encryption networking persistent resumable workflows
- Built-in observability with traces metrics logs auditing for scalable agents

Endpoints (1) Info

Delete
Edit
Test endpoint
Create endpoint

An endpoint maps to a specific version, making it easy to update versions without changing client code.

< 1 > ⚙️

	Name	Status	Descrip...	Endpoi...	Associa...	Cloudw...	Observ...	Last updated (UTC+05:30)
☐	DEFAULT	Ready	-	arn:a...	Version 2	Logs i	Dashboard... i	January 31, 2026, 21:50

Versions (2) Info

A snapshot of the agent created automatically with each update, allowing you to track changes, manage rollbacks, and reference specific states.

< 1 > ⚙️

Version	Status	Descrip...	Created at (UTC+05:30)
Version 2	Ready	-	January 31, 2026, 21:50
Version 1	Ready	-	January 31, 2026, 21:05

- Endpoints act as fixed access URLs that map to a specific agent version, allowing seamless upgrades without modifying client applications.
- Each version captures agent state, enabling rollbacks, reproducibility, and precise change tracking.

1.2 AgentCore Gateway

Centralized control layer for agent tool integration, standardizing access to AWS Lambda functions, REST APIs, and OpenAPI endpoints by converting them into agent-ready tools using the **Model Context Protocol (MCP)**

Amazon Bedrock AgentCore > Gateways

Amazon Bedrock AgentCore

Discover
Overview
Build
Runtime
Gateways
Memory
Policy [Preview](#)
Identity
Built-in tools
Browser
Code Interpreter
Test
Agent sandbox
Assess
Observability [i](#)
Evaluations [Preview](#)

Gateways

How it works



Create a gateway
To use external tools within your agents, create a [gateway](#) to manage a collection of targets that point to tools through OpenAPI schemas, REST API schemas, or as pre-configured integrations. You can configure [inbound Auth](#) and a [policy engine](#) for the gateway, and [outbound Auth](#) for each target to securely manage access to downstream resources.

Create gateway



Add Target
Copy the gateway resource ARN into your runtime agent code to invoke it as part of your session.

Gateways (0) Info

Edit
Delete
Create gateway

Gateways associate targets to third party tools to connect to agents.

< 1 > ⚙️

Gateway name	Description	Status	Creation time
--------------	-------------	--------	---------------



To use external tools within your agents

Target type

☒ **MCP server**
Define this target with pre-configured MCP server.

☐ **Lambda ARN**
Define this target with a Lambda function and S3 resource.

☐ **REST API**
Define this target with an OpenAPI or Smithy schema.

☐ **API Gateway**
Define this target with an API Gateway REST API deployed to a stage.

☐ **Integrations**
Define this target with pre-configured templates from integration providers.

1.3 AgentCore Memory

AgentCore provides a native, fully managed memory system that is deeply embedded into the agent execution lifecycle,

Amazon Bedrock AgentCore > Memory

Amazon Bedrock AgentCore

Discover

Overview

Build

Runtime

Gateways

Memory

Policy

Preview

Identity

Built-in tools

Browser

Code Interpreter

Test

Agent sandbox

Assess

Observability

Evaluations

Preview

Docs and resources

Amazon Bedrock

User guide

Bedrock Service Terms

Memory

How it works

Create a memory resource

Create [Memory](#) to help agents learn over time. Memory stores short-term [events](#) and long-term [memory records](#) (if configured), allowing agents to remember conversations, improve task execution, and provide personalized responses across sessions. Choose from four pre-built strategies - summarization, semantic, user preferences, and episodes (new) - or customize them for fine-grain control.

Create memory

Integrate it within an agent to invoke

Integrate the memory within your agent code using the memory ARN to retain context, knowledge, and user preferences across sessions.

Assess performance using observability

Monitor key metrics, sessions, traces and spans to get real-time insights on performance, accuracy, retrieval effectiveness and reliability of your memory resource.

[View observability dashboard](#)

Memory (1)

Edit

Delete

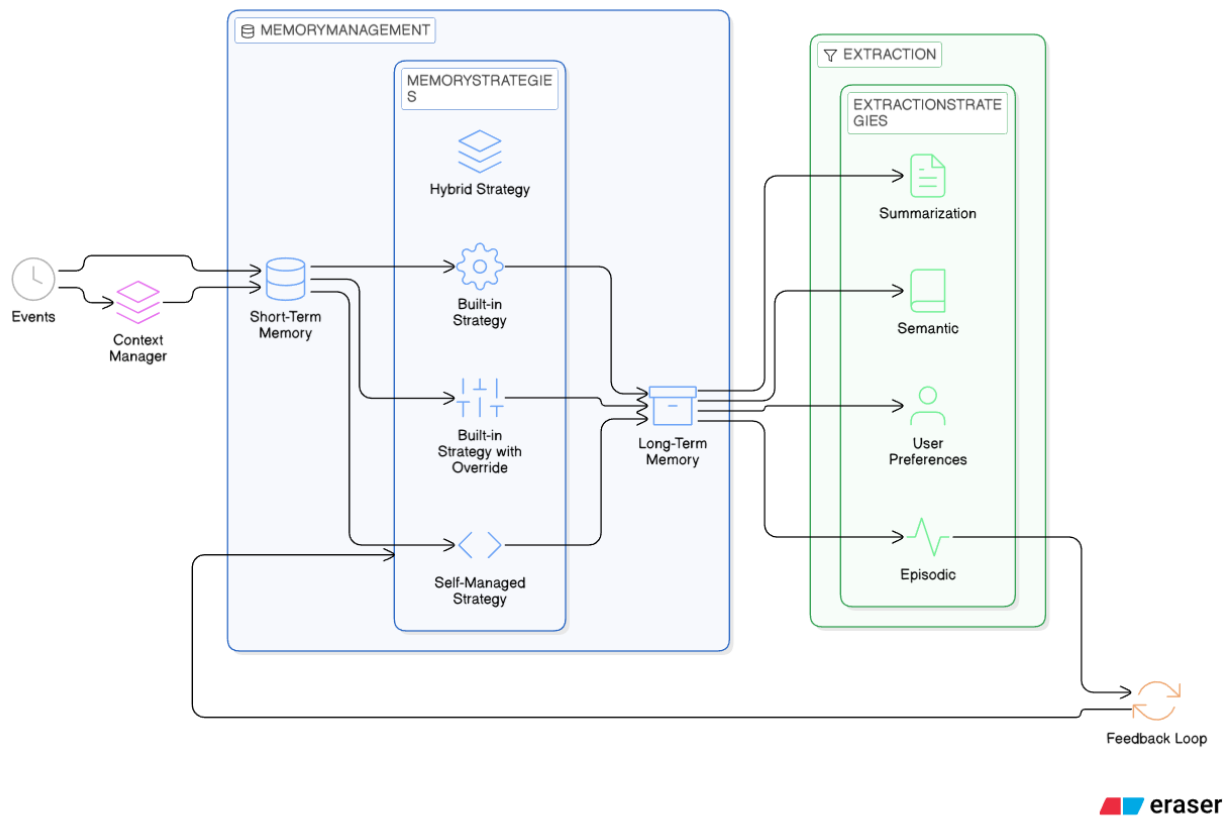
Create memory

Find resources

Memory ID	Memory status	Extrac...	Event ...	Description	Creation time (U...	Last updated (UTC-07:00)
agent_long_term_memo...	Active	3	90	-	January 28, 2026, 2...	January 28, 2026, 23:58 (UTC+05:30)



Built-in AgentCore memory strategies



Semantic Memory Strategy

- Stores important facts automatically for reuse across future agent interactions.

Summary Memory Strategy

- Shortens long conversations into compact context for efficient reasoning.

User Preference Strategy

- Remembers user choices to personalize agent behavior without repeated instructions.

1.3 AgentCore Policy

AgentCore policies are mandatory and enforced at runtime, meaning an agent cannot bypass or override them—even if the model attempts unsafe or unauthorized actions.



The screenshot displays the 'Create policies for PolicyEngine_2ur86 (1)' interface. On the left, the 'Prompt' tab is active, showing a 'Resource scope' dropdown set to 'All gateways'. Below this is a text area for a natural language prompt, with a sample prompt: 'In your own words, describe what you'd like the policies to do. You can add 1 or more policy statements in natural language here.' A 'Generate Cedar' button is at the bottom. On the right, the 'Cedar preview (1)' section shows a preview of the generated Cedar policy for 'policy_q0g7b'. The policy code is as follows:

```
1 permit (  
2   principal is AgentCore::OAuthUser,  
3   action == AgentCore::Action::"InsuranceAPI_file_claim",  
4   resource ==  
5     AgentCore::Gateway::"arn:aws:bedrock-agentcore:us-west-2:01234567890"  
6 )  
7 when  
8 {  
9   principal.hasTag("scope") &&  
10  principal.getTag("scope") like "*insurance:claim*"  
11 };
```

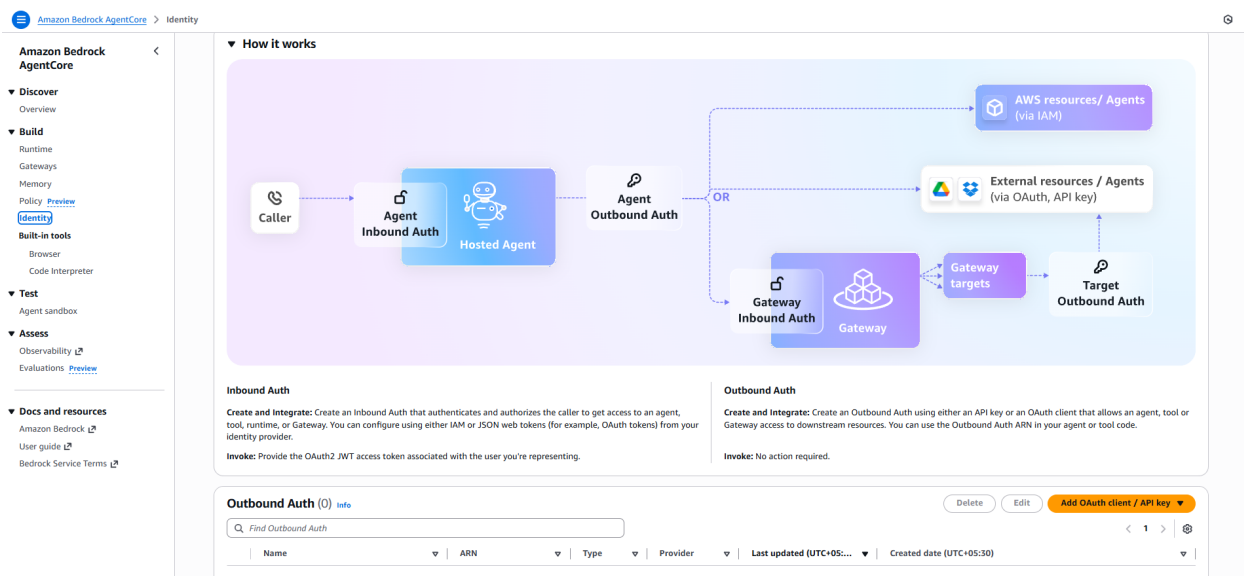
The interface also includes a 'Create policy(s)' button at the top right and a 'Valid' status indicator for the previewed policy.

- Uses **Cedar language** to define natural language boundaries for secure access.

```
permit (
  principal is AgentCore::OAuthUser,
  action == AgentCore::Action::"InsuranceAPI__file_claim",
  resource ==
```

```
AgentCore::Gateway::"arn:aws:bedrock-agentcore:us-west-2:012345678901:gateway/in
surance"
)
when
{
  context.input has claimType &&
  (context.input.claimType == "health" ||
  context.input.claimType == "property" ||
  context.input.claimType == "auto")
};
```

1.4 Agent Identity:



AgentCore Identity defines who an agent is and what it is allowed to do.

Each agent is given its own secure identity, just like a user or service account. This identity controls which tools, APIs, and AWS resources the agent can access.

AgentCore uses IAM-based permissions, the agent can only perform actions that are explicitly allowed. Even if the model tries to do something unsafe, the action is blocked by policy.

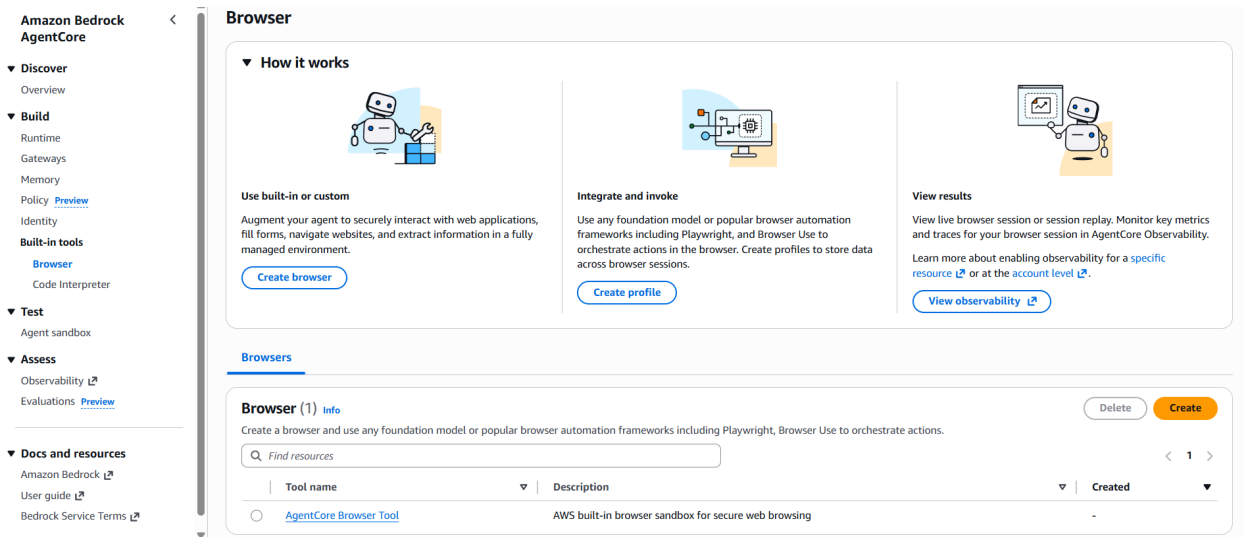
Every action taken by the agent is logged and traceable, making it easy to audit, monitor, and meet security or compliance requirements.

1.4 Agent Built-in tools

Browser Tool

Ready-to-use, managed capabilities that can be integrated into any agent without additional infrastructure.

- Safely browses web information with controlled, auditable, and restricted agent access.



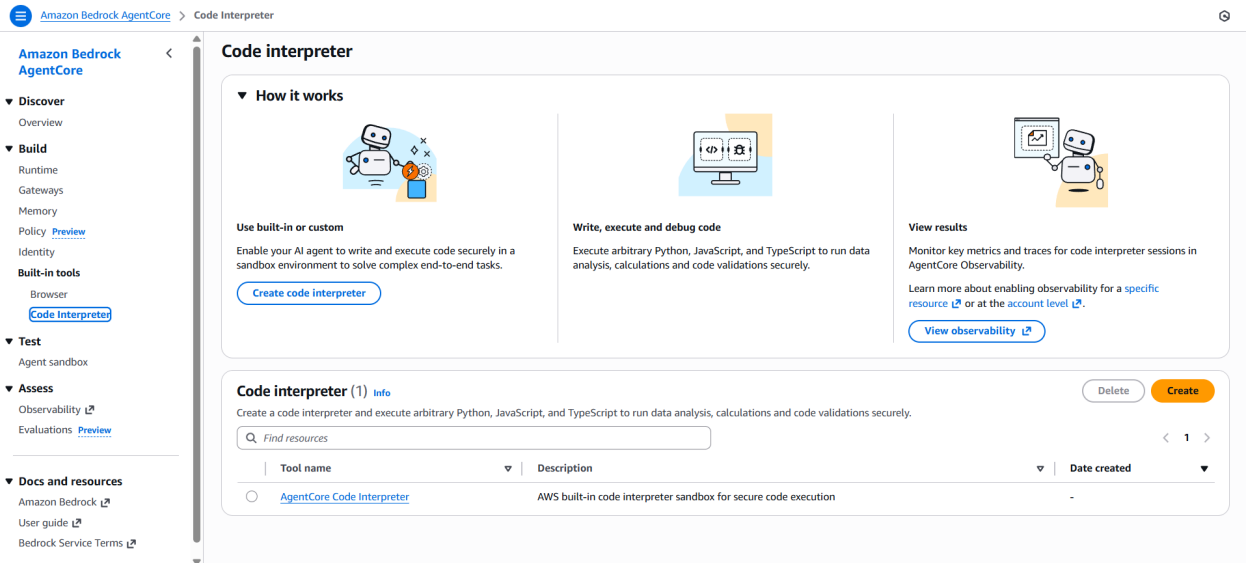
The screenshot shows the Amazon Bedrock AgentCore interface for the Browser tool. On the left is a navigation sidebar with sections: Discover (Overview), Build (Runtime, Gateways, Memory, Policy, Identity, Built-in tools), Test (Agent sandbox), Assess (Observability, Evaluations), and Docs and resources (Amazon Bedrock, User guide, Bedrock Service Terms). The main content area is titled 'Browser' and includes a 'How it works' section with three cards: 'Use built-in or custom' (with a 'Create browser' button), 'Integrate and invoke' (with a 'Create profile' button), and 'View results' (with a 'View observability' link). Below this is a 'Browsers' section with a search bar and a table listing browser tools. The table has columns for Tool name, Description, and Created. One tool is listed: 'AgentCore Browser Tool' with the description 'AWS built-in browser sandbox for secure web browsing'.

Tool name	Description	Created
AgentCore Browser Tool	AWS built-in browser sandbox for secure web browsing	-

- Agents securely browse web sources, validate information, gather external data, supporting accurate, auditable decision making.
- Agents analyze data, perform calculations, generate insights, combining browsing and computation for autonomous reasoning workflows.

Code Interpreter Tool

- Enables agents to safely run code inside a sandboxed environment
- Perform calculations, analyze data, transform inputs, and validate logic in real time.



The screenshot shows the Amazon Bedrock AgentCore Code Interpreter interface. On the left is a navigation sidebar with sections: Discover (Overview), Build (Runtime, Gateways, Memory, Policy, Identity), Built-in tools (Browser, Code Interpreter), Test (Agent sandbox), Assess (Observability, Evaluations), and Docs and resources (Amazon Bedrock, User guide, Bedrock Service Terms). The main content area is titled 'Code interpreter' and includes a 'How it works' section with three steps: 'Use built-in or custom' (with a 'Create code interpreter' button), 'Write, execute and debug code' (describing execution of Python, JavaScript, and TypeScript), and 'View results' (with a 'View observability' button). Below this is a 'Code interpreter (1)' section with a search bar and a table listing the 'AgentCore Code Interpreter' tool, described as an 'AWS built-in code interpreter sandbox for secure code execution'.

Features of AgentCore Agent

Human-in-the-Loop Native Support

- Tool execution can be paused for explicit human approval, then resumed without losing agent state.

Built-In Observability & Auditing

- Tool calls are automatically logged with execution context, outcomes, and policy decisions for compliance and debugging.

What's the difference between using Agent Core versus directly calling the Bedrock API?

AgentCore manages orchestration, retries, state, and tools automatically within a unified framework. Direct APIs require manual coordination, custom error handling, and explicit state management. As a result, AgentCore reduces development effort, complexity, and long-term maintenance.

Orchestration

- **Agent Core:** Automatically manages multi-step workflows, tool chaining, and decision flow
- **Direct API:** You manually coordinate every step and tool call

Retry & Reliability

- **Agent Core:** Built-in retries, exponential backoff, and timeout handling
- **Direct API:** Custom retry loops and error handling must be written and maintained

State Management

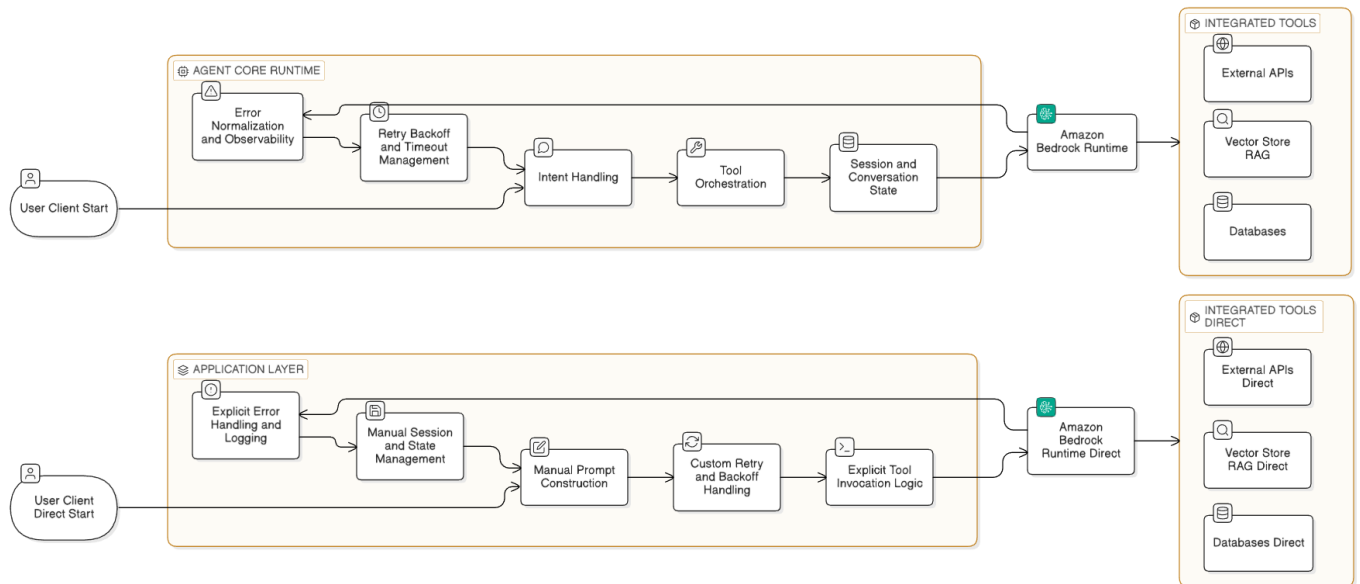
- **Agent Core:** Session and conversation context handled automatically
- **Direct API:** Manual tracking of conversation history and memory

Tool Integration

- **Agent Core:** Declarative tool registration and automatic invocation
- **Direct API:** Manual tool schema definition, invocation logic, and response parsing

Development & Maintenance Effort

- **Agent Core:** Faster development with significantly less code and lower maintenance
- **Direct API:** More boilerplate, higher complexity, and increased maintenance cost



How can Amazon Bedrock AgentCore be integrated with AWS services such as Lambda, Step Functions, API Gateway, EventBridge, or SQS to build serverless, event-driven architectures?

AgentCore agents are versioned, exposed via **secure endpoints**, and invoked by **Lambda with IAM permissions**, enabling governed, scalable, event-driven orchestration.

- AgentCore exposes versioned, secure agent endpoints invoked by Lambda, enabling scalable event-driven execution through API Gateway, EventBridge, and SQS.
- Step Functions orchestrate multi-step agent workflows with retries, approvals, and observability, while IAM enforces strict access control and governance.

