

Amazon Bedrock AgentCore

A tour of every service, from harness to registry

Deck 00 of the AgentCore training series

Chiwai Chan

Who am I?

Tinkerer · Cloud · IoT · Robotics · Generative AI

<https://chiwaichan.co.nz>

<https://nz.linkedin.com/in/chiwaichan>

<https://github.com/chiwaichan>

<https://x.com/chiwaichanconz>

PART 1

The big picture

What is Amazon Bedrock AgentCore?

- An agentic platform to build, deploy and operate agents securely at scale
- Works with any framework and any foundation model
- No infrastructure to manage
- Modular: use services together or independently
- Frameworks named: CrewAI, LangGraph, LlamaIndex, Strands Agents

Open-source flexibility **and** enterprise-grade security — you don't have to choose.

Problems it solves

- **Run** agents securely at scale — isolated sessions, serverless hosting
- **Act** across tools and data with the right permissions and governance
- **Monitor** performance and quality in production
- **Remember** context across turns and sessions (agents are otherwise stateless)
- **Improve** continuously with evaluations, recommendations and A/B tests

What can you build?

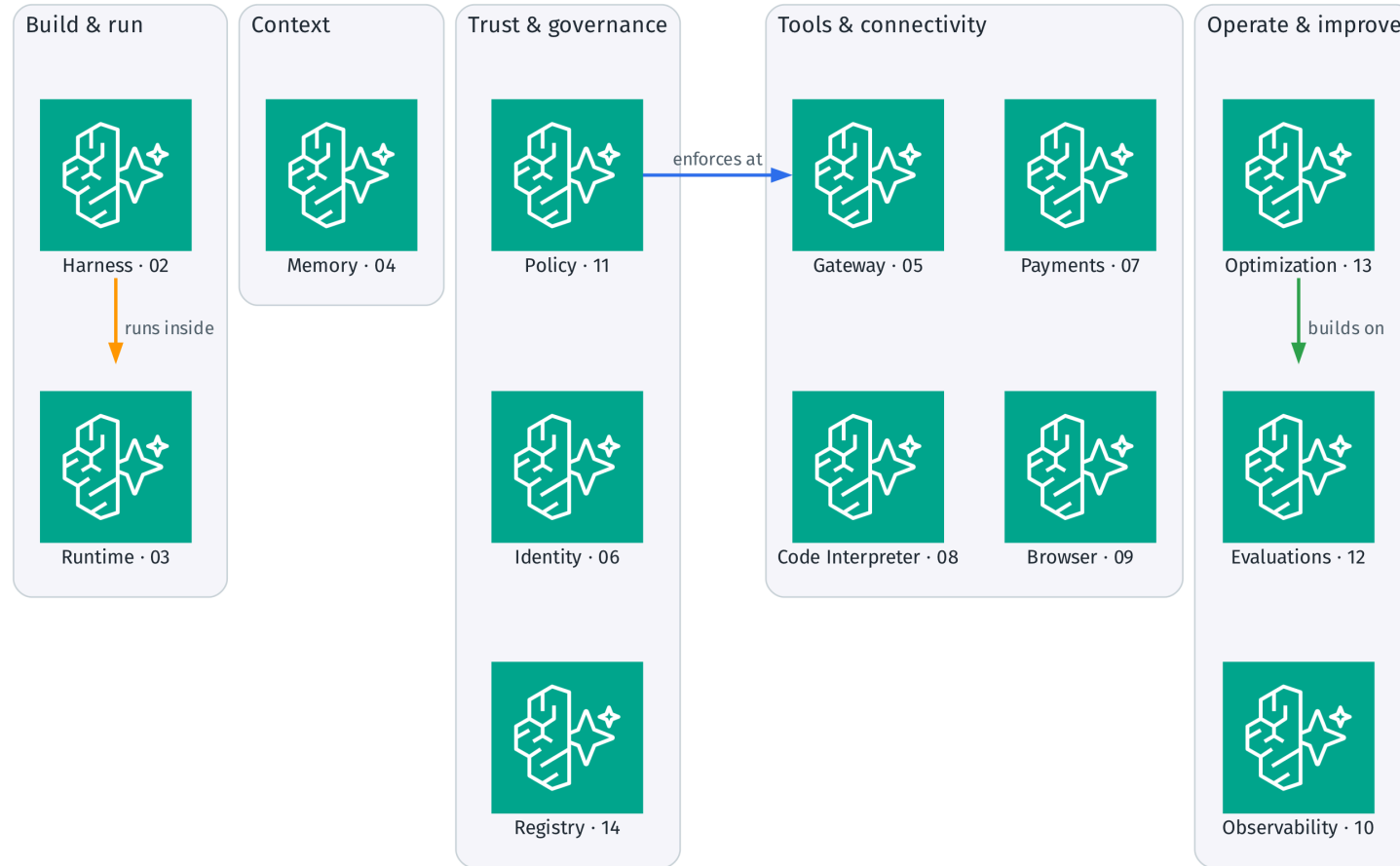
Use case	What it looks like
Agents	Autonomous apps that reason, use tools, keep context
Tools & MCP servers	Turn APIs, Lambda or OpenAPI specs into MCP tools
Agent platforms	A paved path: approved tools, shared memory, governed access

How the services fit together

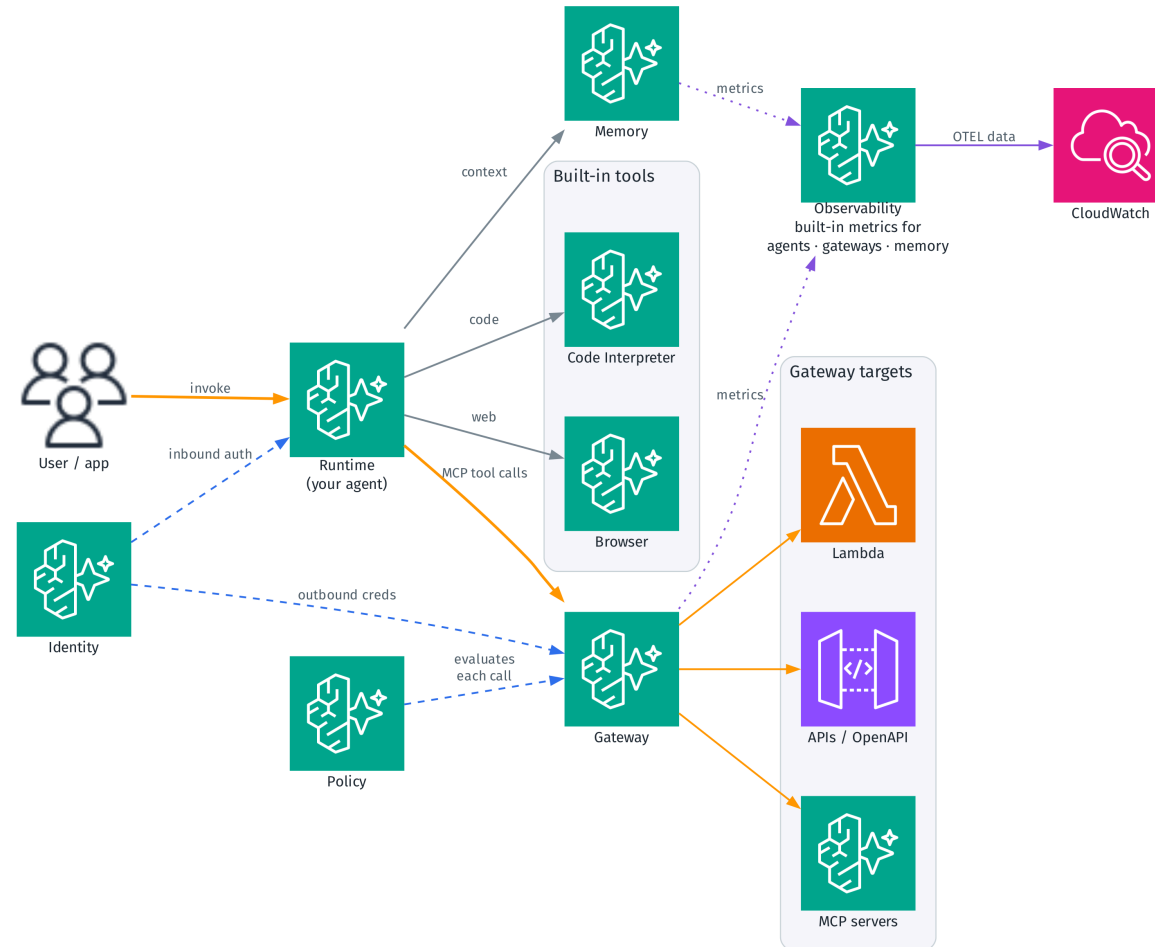
Layer	Services
Build & run	Harness (managed loop) · Runtime (host your code)
Context	Memory (short-term + long-term)
Tools & connectivity	Gateway · Code Interpreter · Browser · Payments
Trust & governance	Identity · Policy · Registry
Operate & improve	Observability · Evaluations · Optimization

Harness runs **inside** Runtime; Policy enforces **at** Gateway; Optimization **builds on** Evaluations.

Service map



A request, end to end



PART 2

Build & run

Harness — a managed agent loop

- Declare model, system prompt, tools, skills as **configuration**; AgentCore runs the loop
- Each session: isolated **microVM** with its own filesystem and shell
- Models from Amazon Bedrock, OpenAI, Google Gemini, or LiteLLM-compatible providers
- Switch model providers mid-session without losing context
- Tools via AgentCore Gateway, MCP servers, built-in Browser and Code Interpreter
- Powered by Strands Agents; no separate harness charge

Harness — lifecycle and when to graduate

- Built-in short-term and long-term memory; bring your own container
- Mount S3 Files or EFS to share data across sessions
- Immutable versions + named endpoints for safe rollout and rollback
- `InvokeHarness` state in AWS Step Functions
- Export to Strands code and run on Runtime when config isn't enough
- **GA** in all regions where AgentCore is supported

Harness vs. Runtime

	Harness	Runtime
Orchestration loop	Provided (Strands)	Yours
How you add features	Config field	Code (usually AgentCore SDK)
Choice of framework	No	Yes
Bidirectional streaming	No	Yes
Graph / workflow patterns	No	Yes
Session isolation, VPC, versioning	Yes	Yes

Runtime — serverless hosting for agents and tools

- Framework agnostic: LangGraph, Strands, CrewAI or custom code
- Any model: Amazon Bedrock, Anthropic Claude, Google Gemini, OpenAI
- Protocols: MCP and Agent to Agent (A2A)
- Each session in a dedicated microVM; memory sanitized when it ends
- Up to **8 hours** per session on microVMs, **14 days** on Instances
- **100MB** payloads; HTTP plus WebSocket bidirectional streaming

Runtime — key building blocks

- **Runtime**: hosts your agent or tool code; has a unique identity
- **Versions**: immutable snapshots; each config update creates a new one
- **Endpoints**: addressable pointers to a version; **DEFAULT** tracks latest
- **Sessions**: identified by **runtimeSessionId**, isolated microVM each
- Consumption-based pricing: CPU billing aligned to active processing
- Built-in inbound auth via AgentCore Identity (Okta, Entra ID, Cognito)

PART 3

Context & connectivity

Memory – context-aware agents

Type	What it holds	Example
Short-term	Turn-by-turn events in one session	"What about tomorrow?" after a Seattle weather question
Long-term	Insights extracted across sessions	Remembering a window-seat preference

- Long-term records come from **strategies** added to the memory resource
- Built-in strategies: semantic, user preference, summary, episodic
- Also built-in overrides and self-managed strategies

Gateway — one secure entry point for agentic traffic

- Converts APIs, Lambda functions and services into **MCP** tools
- Fronts other agents and HTTP services (incl. A2A) via passthrough
- Routes inference across model providers by the requested model
- Inbound **and** outbound authentication in one managed service
- Semantic tool search across thousands of tools
- 1-click integrations: Salesforce, Slack, Jira, Asana, Zendesk

Gateway – target categories

Category	Behaviour	Examples
MCP	Aggregated into one virtual MCP server	Lambda, API Gateway, OpenAPI, Smithy, MCP servers, integrations, connectors
HTTP	Direct, no aggregation or translation	AgentCore Runtime agents, A2A agents, passthrough
Inference	Model-based routing to providers	Amazon Bedrock, OpenAI, Anthropic

- Inbound auth: OAuth (JWT), IAM (SigV4), authenticate-only, or none

Identity — identity for agents and workloads

- Agent identities are **workload identities** in a central directory
- **Token vault**: stores OAuth tokens, client credentials and API keys, KMS-encrypted
- OAuth 2.0 client credentials (2LO) and authorization code (3LO) flows
- Built-in providers: Google, GitHub, Slack, Salesforce, Atlassian (Jira)
- SDK decorators: `@requires_access_token`, `@requires_api_key`
- Native integration with Runtime and Gateway

Payments – microtransactions for agents

- Lets agents pay for APIs, MCP servers and content
- Protocols: **x402** and **Machine Payments Protocol (MPP)**, both on HTTP 402
- Wallet providers: Coinbase CDP and Stripe (Privy)
- Per-session budget (`maxSpendAmount`, currency) plus an expiry time
- Coinbase x402 Bazaar MCP server exposed through AgentCore Gateway
- Launched in **Preview** (May 2026)

Code Interpreter — sandboxed code execution

- Agents write and run code in an isolated sandbox
- Languages: Python, JavaScript, TypeScript
- Pre-built runtimes with common libraries pre-installed
- Files: up to 100 MB inline; up to 5 GB via S3 with terminal commands
- Default 15-minute execution, extendable up to 8 hours
- Custom session properties and network modes; CloudTrail logging

Browser — a managed cloud browser

- Isolated, containerized browser for agents to use web apps
- AWS managed `aws.browser.v1`, or a custom browser (recording, network, IAM role)
- Sessions: default 15 minutes, maximum 8 hours; run many at once
- **Automation** endpoint (WebSocket) — Strands, Nova Act, Playwright
- **Live View** lets an end user watch and interact in real time
- Session recording to your S3 bucket, replay in the console

PART 4

Operate & govern

Observability — trace, debug, monitor

- Step-by-step visualisation of the agent's execution path
- CloudWatch dashboards: sessions, latency, duration, tokens, errors
- Telemetry in **OpenTelemetry (OTEL)**-compatible format
- Built-in metrics for agents, gateways and memory by default
- Instrument your code for extra spans, metrics and logs
- All data stored in Amazon CloudWatch

Policy – deterministic control over tool calls

- Policy engines associated with gateways; **every** tool call evaluated first
- Enforced at the gateway boundary, outside agent code
- Author in **Cedar**, natural language, or **Dogwood**
- Principals: `AgentCore::OAuthUser` (JWT) or `AgentCore::IamEntity` (IAM)
- Temporal policies reason over a session's action history
- Decisions logged via CloudWatch for audit

Evaluations — measure agent quality

Type	When
Online	Continuously, sampling live production sessions
On-demand	Targeted, on chosen span or trace IDs
Batch	Async job over many sessions; aggregate scores

- Evaluators: built-in (LLM-as-a-judge), third-party (DeepEval, AutoEval), custom
- Custom: your own LLM judge or code-based via Lambda
- Frameworks: Strands, LangGraph with OpenTelemetry / OpenInference

Optimization — the improvement loop

1. **Recommendations**: optimised system prompt or tool descriptions from traces
2. **Configuration bundles (optional)**: versioned, immutable config snapshots
3. **A/B testing**: split traffic via Gateway; online evals report significance
4. Promote the winner, then repeat from the new baseline

Builds on AgentCore Evaluations — you pick the target evaluator to optimise for.

AWS Agent Registry – discover and govern

- Central catalog: MCP servers, agents, tools, skills, custom resources
- **Registries** hold **records**; approval workflow gates discoverability
- Hybrid search: semantic + keyword
- Native **MCP endpoint** for MCP-compatible clients
- Authorization: IAM or JWT from your corporate IdP
- **GA** (August 2026) with AWS Organizations auto-detection

Security — shared responsibility

- AWS secures the cloud; you secure what you run in it
- Data at rest encrypted by default (AWS owned KMS keys); TLS 1.2+ in transit
- Gateways can use a customer-managed KMS key
- Resource-based policies for Runtime, Gateway and Memory
- Use `aws:SourceArn` / `aws:SourceAccount` against confused deputy
- VPC and AWS PrivateLink options

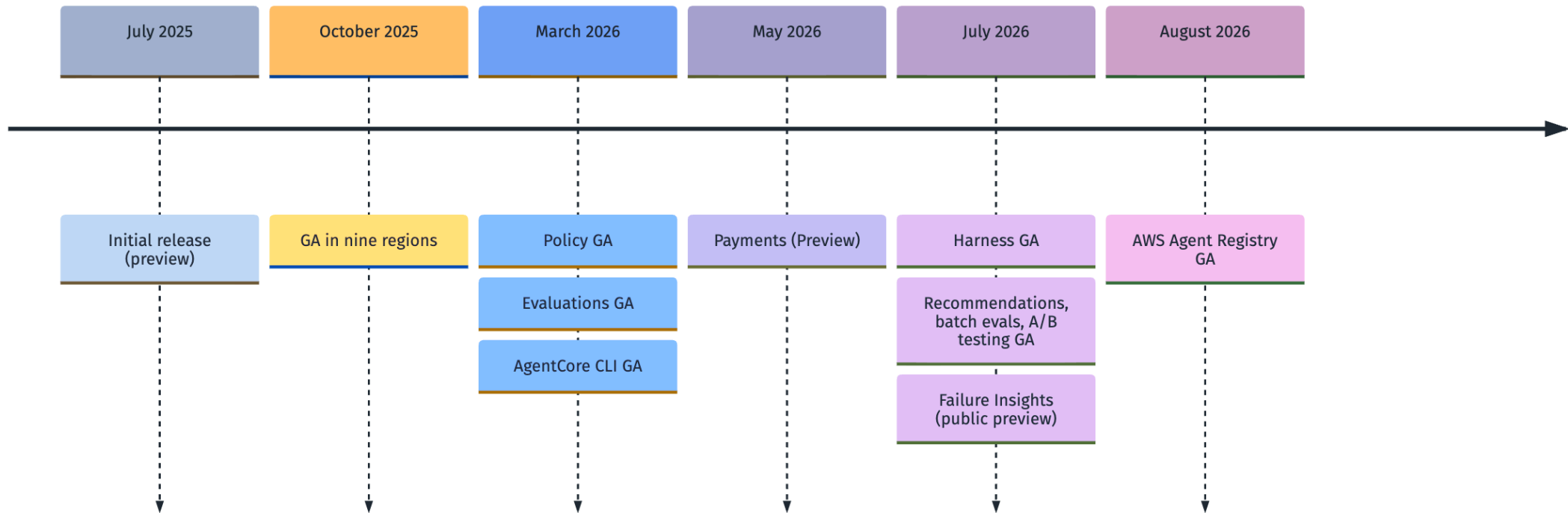
PART 5

Getting started

Availability at a glance

Milestone	When
Initial release (preview)	July 2025
AgentCore GA in nine regions	October 2025
Policy, Evaluations, AgentCore CLI GA	March 2026
Payments in Preview	May 2026
Harness, Recommendations, Batch evals, A/B testing GA	July 2026
Failure Insights public preview	July 2026
AWS Agent Registry GA	August 2026

Availability timeline



Getting started – the AgentCore CLI

```
npm install -g @aws/agentcore  
agentcore create      # Harness, Agent, or Skip  
agentcore dev         # local server + agent inspector  
agentcore deploy      # CDK under the hood  
agentcore invoke --prompt "Hello, what can you do?"
```

- Needs Node.js 20+; Python 3.10+ for agent code
- Build types: **CodeZip** (default, no Docker) or **Container**
- Add capabilities: **agentcore add memory | gateway | credential | evaluator**

Interfaces for AgentCore

Interface	Use it for
AgentCore CLI	Create, dev, deploy, invoke, status
AgentCore Python SDK	Runtime, Memory, Tools, Identity, Evaluations primitives
AgentCore MCP server	Transform, deploy, test from Kiro, Cursor, Claude Code, Amazon Q CLI
AWS SDK / AWS CLI	Full control plane + data plane APIs
AgentCore console	Manage services, agent sandbox testing

CLI and Python SDK don't cover every operation — fall back to the AWS SDK.

Where to go next

#	Deck	#	Deck
01	Getting Started & Tooling	09	Browser
02	Harness	10	Observability
03	Runtime	11	Policy
04	Memory	12	Evaluations
05	Gateway	13	Optimization
06	Identity	14	Agent Registry
07	Payments	15	Security
08	Code Interpreter		

UP NEXT

01 · Getting Started & Tooling

Open the next deck → · All decks in the series

References

AgentCore Developer Guide — 35 pages (<https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/<page>.html>)

- [what-is-bedrock-agentcore](#)
- [agentcore-get-started-cli](#)
- [develop-agents](#)
- [harness](#)
- [harness-vs-runtime](#)
- [agents-tools-runtime](#)
- [runtime-how-it-works](#)
- [memory](#)
- [memory-strategies](#)
- [built-in-strategies](#)
- [gateway](#)
- [gateway-core-concepts](#)
- [gateway-supported-targets](#)
- [identity](#)
- [identity-overview](#)
- [key-features-and-benefits](#)
- [payments](#)
- [code-interpreter-tool](#)
- [browser-tool](#)
- [observability](#)
- [policy](#)
- [policy-core-concepts](#)
- [evaluations](#)
- [evaluations-types](#)
- [evaluators](#)
- [optimization](#)
- [optimization-how-it-works](#)
- [insights](#)
- [registry](#)
- [security](#)
- [data-protection](#)
- [data-encryption](#)
- [resource-based-policies](#)
- [cross-service-confused-deputy-prevention](#)
- [release-notes](#)