



Mastering AI Agents: Building Production- Ready Applications



Getting started with Amazon
Bedrock AgentCore

▼ Prerequisites

At an AWS Event (Setup)

Self paced (Setup)

Sagemaker AI Studio

Amazon Bedrock AgentCore
Fundamentals (Optional)

Lab 1: Create the Agent Prototype

Lab 2: Enhance your Agent with
Memory

Lab 3: Scale with Gateway and
Identity

**Lab 4: Deploy the Agent to
production with Observability**

(Optional) Lab 5: Build a Customer-
Facing Frontend Application

Lab 6: Clean up

AgentCore

AgentCore Documentation

▼ AWS account access

Open AWS console
(us-west-2)

Get AWS CLI credentials

Exit event

[Event dashboard](#) > Lab 4: Deploy the Agent to production with Obse...

Lab 4: Deploy the Agent to production with Observability

Overview

In prior labs, we made from our Customer Support Agent a tool-powered memory-aware intelligent assistant for the end users. But how to move it from a laptop based prototype to a scalable production endpoint all while securely granting our users access to it?

[AgentCore Runtime](#) is a secure, serverless runtime designed for deploying and scaling AI agents and tools. It supports any frameworks, models, and protocols, enabling developers to transform local prototypes into production-ready solutions with minimal code changes. This will transform our prototype into a production-ready system that can handle real-world traffic with full monitoring and automatic scaling. Additionally, AgentCore Runtime integrates seamlessly with [AgentCore Observability](#) to provide full visibility into your agent's behavior in production.



Prerequisite for AgentCore Observability

To view metrics, spans, and traces generated by the AgentCore Runtime using AgentCore Observability, you first need to complete a one-time setup to turn on Amazon CloudWatch Transaction Search. More specifically, [following this documentation](#) , you have to:

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/> .
2. From the navigation pane, under **Application Signals**, choose **Transaction Search**.
3. Choose **Enable Transaction Search**.
4. Select the box to ingest spans as structured logs, and enter a percentage of spans to be indexed. You can index spans at 1% for free and change the percentage later based on your requirements.

Key benefits:

- **Framework Agnostic:** Works with any Python-based agentic framework (Strands Agents, LangGraph, CrewAI)
- **Model Flexible:** Support for LLMs in Amazon Bedrock, OpenAI, and other LLM providers
- **Production Ready:** Built-in health checks and monitoring
- **Easy Integration:** Easy management through AWS console and APIs and support for real-world production workloads.
- **Scalable:** Designed for enterprise workloads

What we are building

In this lab, by means of the Amazon BedrockAgentCore Python SDK, you will wrap your AgentCore Memory-enhanced (Lab 2) Strands Agent (Lab 1) with a BedrockAgentCoreApp and define a dedicated `invoke` entrypoint. Then, you will use the `configure` and `launch` capabilities of the SDK to deploy your agent to AgentCore Runtime. Your application is then able to invoke this agent via the same SDK.

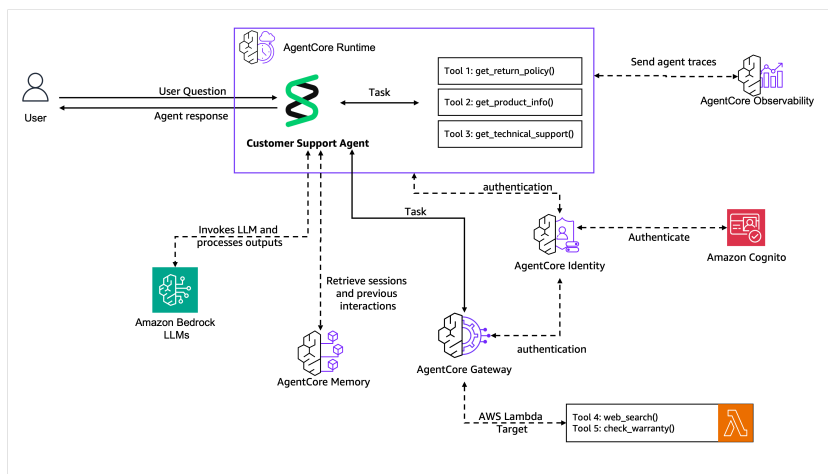
By the end of this lab, you'll understand:

- How to initialize the AgentCore Runtime App and define the entrypoint for runtime invocation
- How to configure, build and deploy your AgentCore Runtime instance
- How to programmatically invoke your agent from within the AgentCore Runtime
- How to monitor and analyze your agent's traces with AgentCore Observability

Architecture diagram

© 2008 - 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy policy](#) [Terms of use](#) [Cookie preferences](#)

Below is the updated Solution Architecture diagram including the Agent deployment within the AgentCore Runtime:



Incremental Workshop Journey: Lab 4 of 5-part progressive workshop.

Steps to follow

Note: The code snippet in the following steps are shown for reference purposes only. The complete implementation is provided in the accompanying Jupyter notebook for this lab section.

Step 1: Prepare Your Agent for AgentCore Runtime

Let's first define the necessary AgentCore Runtime components via Python SDK within our previous local agent implementation. Expand the following snippet to see the additional required commands:

► Code Snippet

Step 2: Deploy to AgentCore Runtime

[AgentCore Starter Toolkit](#) [\[↗\]](#) allows us to configure, build and deploy our BedrockAgentCoreApp to the AgentCore Runtime.

► Code Snippet

Note: You might get a warning mentioning "Platform Mismatch". It's only for information, and safe to ignore for the scope of this workshop.


Step 3: Invoke Your Deployed Agent

Now that our agent is deployed and ready, we can invoke it with the right Cognito authorization token using the same AgentCore Starter Toolkit:

► Code Snippet

And it is all it takes to have a secure and scalable endpoint for our Agent with no need to manage all the underlying infrastructure!

Step 4: AgentCore Observability

[AgentCore Observability](#)  delivers powerful monitoring, tracing, and debugging capabilities for AI agents. It automatically emits telemetry in the standardized **OpenTelemetry (OTEL)** format, enabling seamless integration with existing observability platforms. Through **Amazon CloudWatch** dashboards, teams gain real-time visibility into agent workflows with filtering, metadata tagging, and step-by-step visualizations of execution paths and intermediate outputs—making it easier to detect and resolve performance bottlenecks or failures.

All telemetry data such as metrics, logs, and spans are centrally stored in CloudWatch, including a dedicated **Generative AI observability dashboard** that provides trace visualizations, span metrics, and error breakdowns. In addition, **AWS CloudTrail** captures and records all API-level interactions across AgentCore components, creating a complete audit trail that supports troubleshooting, security, and compliance.

This foundation establishes multiple layers of production-grade observability:

1. **Generative AI Observability Dashboard** – A CloudWatch console view purpose-built for AI agents, offering real-time metrics, session analytics, trace visualizations, and error breakdowns for rapid diagnosis and optimization.
2. **CloudWatch Metrics** – Automatically published indicators such as invocations, latency, duration, sessions, and error rates for Runtime, Memory, and Gateway, powering dashboards, alerts, and trend analysis.
3. **CloudWatch Logs** – Structured, time-stamped logs (including span and event records) that support detailed troubleshooting and forensic analysis.

4. AWS CloudTrail – An immutable record of control-plane and data-plane API activity—capturing who did what, when, and from where—to enhance security monitoring and compliance.

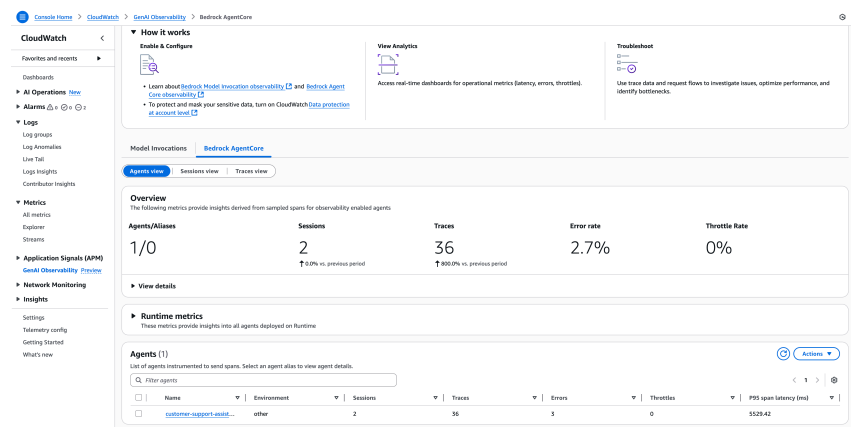
Now that your agent is deployed to the **AgentCore Runtime** and integrated with **Memory** and **Gateway**, let's explore each observability layer in detail.

4.1 AgentCore's Built-in Observability Dashboard

The CloudWatch GenAI Observability dashboard is the primary view for day-to-day monitoring, offering immediate insights into agent activity, sessions, and traces without requiring additional setup.

By default, AgentCore forwards agent traces to **CloudWatch GenAI Observability**. To view them, open CloudWatch → GenAI Observability → Bedrock AgentCore. From here you can review agent activity, drill into sessions, and open traces for detailed analysis.

Agents view



The **Agents view** gives a high-level snapshot of all deployed agents. It's the first stop when you want to understand how your agents are behaving in production.

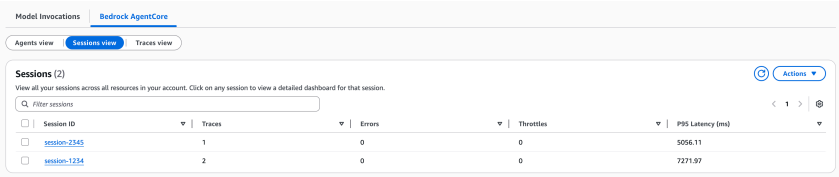
- **Summary metrics:** total sessions, traces, errors, and throttles across agents.
- **Runtime metrics:** aggregate KPIs for all agents, including session counts, invocations, errors, throttles, and latency trends, displayed with time-series graphs to highlight spikes or anomalies.

- **Per-agent breakdown:** a detailed table showing sessions, traces, errors, throttles, and P95 span latency for each agent, enabling comparisons across multi-agent deployments.

These views let you track **traffic patterns, error trends, and performance at both the account and individual agent level**, making the Agents view the starting point for monitoring overall agent health.

Sessions

The Sessions view shows the list of all the sessions associated with all agents in your account, with filters and drill-downs that make it easy to investigate behavior for a specific session.



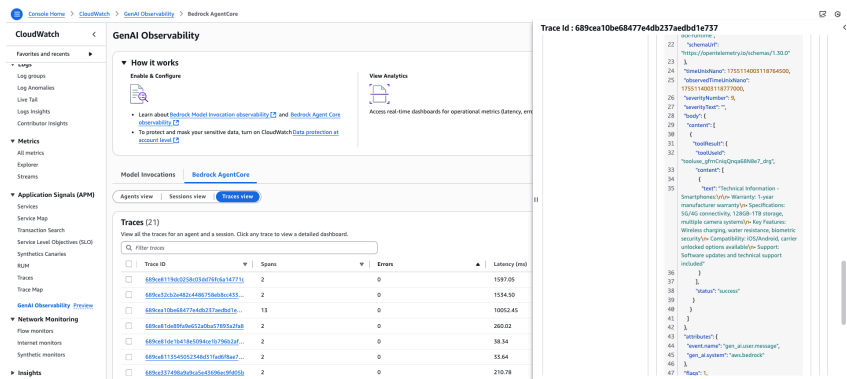
Session ID	Traces	Errors	Throttles	P95 Latency (ms)
session-2545	1	0	0	5056.11
session-1234	2	0	0	7271.97

- **Session listing:** Displays all agent conversation sessions with sortable columns for session ID, trace count, errors, throttles, and P95 latency.
- **Session detail navigation:** Click any session ID to open a dedicated dashboard that shows per-session metrics (latency, errors, span count)
- **Trace a conversation end-to-end** by jumping from the session to its associated traces and spans
- **Session comparison:** Easily spot anomalies by comparing sessions—such as high latency, repeated tool failures, or unusually long conversations.

In this view, you gain insights into the full **lifecycle and outcome** of each session, **engagement patterns, context usage** (e.g., memory interactions like RetrieveMemoryRecords), and **tool utilization** (which Gateway tools were invoked). It serves as your workspace for diagnosing conversation-specific performance and behavior.

Traces

The **Traces view** offers a focused lens into the internal execution path of each request processed by your agents. It's the go-to place for understanding how individual requests unfold operationally.



- **Trace listing:** Displays all recorded agent traces for your account, with sortable columns for trace ID, span count, errors, and latency—letting you quickly identify slow or failing requests.
- **Filtering & sorting:** Easily narrow down traces using filters (e.g., latency, status, attributes) or sort by columns to prioritize analysis.
- **Drill-in analytics:** Use the **Actions** menu to open **Logs Insights** for contextual logs.
- **Execution detail:** Each trace reveals the full execution flow—spans representing discrete operations, tool invocations, and internal decision points—helping visualize the end-to-end request journey.

These features enable you to **trace every step of an agent's request**, pinpoint latency bottlenecks, uncover error patterns, and gain a granular understanding of runtime behavior.

4.2 AgentCore Observability Metrics

AgentCore automatically publishes **built-in observability metrics** for all primitives Runtime, Memory, and Gateway to **Amazon CloudWatch** under the namespace **AWS/Bedrock-AgentCore**. These metrics give you real-time visibility into request activity, latency, error rates, session usage, and performance. You can use them to build dashboards, configure alarms, and analyze trends for production operations.

Navigating to AgentCore Metrics in CloudWatch

1. Open CloudWatch Console

- Navigate to the **CloudWatch** service in the AWS Console.
- Select **Metrics** from the left navigation panel.

- Click **All metrics**.

2. Locate AgentCore Metrics

- In the metrics browser, select the namespace **AWS/Bedrock-AgentCore**.
- Explore the available dimension combinations for each resource type.

3. Filter by Resource

- Use the search bar to enter the ARN or name of your resource.
- Examples: `customer_support_agent`, `CustomerSupportMemory`, or `customersupport-gw`.

Runtime Metrics

Runtime metrics capture how your agent handles requests in production and how users experience responsiveness.

Key Metrics to Monitor

- **Invocations** – Total requests received by the agent, showing usage and peak load periods.
- **Latency** – End-to-end response time (request to final token), including model inference, memory lookups, and tool execution.
- **Sessions** – Number of active agent sessions, useful for capacity planning and engagement analysis.
- **User Errors** – Invalid requests (400), missing resources (404), or permission errors (403).
- **System Errors** – Internal service errors (500) that may indicate infrastructure issues.
- **Throttles** – Requests rejected due to TPS or quota limits (429).

Memory Metrics

Memory metrics measure how efficiently the agent stores and retrieves conversation context, and how background processing maintains long-term memory quality over time.

Key Metrics to Monitor

- **Latency** – End-to-end processing time for memory operations.
- **Invocations** – Total number of API requests to the Memory service.

- **System Errors** – Memory API calls that failed with AWS server-side errors (5xx).
- **User Errors** – Memory API calls that failed with client-side errors (4xx).
- **Errors** – Total errors across control-plane and data-plane operations, including ingestion failures.
- **Throttles** – Requests throttled (429), not counted as invocations or errors.
- **Creation Count** – Number of new memory events and records created.

Gateway Metrics

Gateway metrics provide visibility into tool execution, MCP operations, and request distribution across targets.

Key Metrics to Monitor

- **Invocations** – Total requests made to Gateway data plane APIs.
- **Latency** – Time from receiving a request until the first response token is sent.
- **Duration** – Full end-to-end request time until the final response token is sent.
- **TargetExecutionTime** – Time taken by the target (Lambda, API) to execute, excluding Gateway overhead.
- **TargetType** – Distribution of requests served by target type (MCP, Lambda, OpenAPI).
- **System Errors (5xx)** – Requests that failed due to server-side issues.
- **User Errors (4xx)** – Requests that failed due to client errors (excluding throttles).
- **Throttles (429)** – Requests throttled due to exceeded limits.

Best Practices for Metrics

- **Dashboards** – Combine Runtime, Memory, and Gateway metrics in a single CloudWatch dashboard.
- **Alarms** – Configure CloudWatch Alarms for high latency, error rates, or throttling.
- **Trends** – Monitor growth in CreationCount and Sessions to anticipate scaling needs.
- **Correlation** – Combine metrics with spans and logs for deep troubleshooting.

With these metrics in CloudWatch, you have baseline observability across AgentCore resources, enabling real-time monitoring and proactive alerting.

4.4 AgentCore Observability Logs

CloudWatch Logs capture detailed, time-stamped events across AgentCore components. These structured JSON logs provide visibility into agent operations, background processing, and system behavior for deep troubleshooting and performance analysis. Let's explore the logs of Agentcore Gateway as an example.

Gateway Logs - Setup and Navigation

Gateway logs capture the MCP (Model Context Protocol) operations that occur when your customer support agent (from Lab 1) uses the centralized tools you configured in Lab 3. These logs provide detailed insight into how AgentCore Gateway manages tool discovery, execution, and performance for the `check_warranty_status`, `web_search`, and `get_product_info` tools.

Enable Gateway Log Delivery:

1. Navigate to your Gateway resource from Lab 3:

- Go to Amazon Bedrock AgentCore → Gateways → **customersupport-gw**

2. Configure Log Delivery:

- In the **Observability** section, find **Log delivery**
- Click **Add** → Select **Log type**: APPLICATION_LOGS
- **Destination log group**: `/aws/vendedlogs/bedrock-agentcore/gateway/APPLICATION_LOGS/customersupport-gw-xxxxxxxxxxxx`

The log group name includes an automatically generated ID suffix (like `customersupport-gw-dcbgswzb5p`) unique to your Gateway resource.

Add delivery to Amazon CloudWatch Logs



Log type

APPLICATION_LOGS

Destination log group

Log groups prefixed with '/aws/vendedlogs/' will be created automatically. Other log groups must be created prior to setting up a log delivery. [Create a new log group](#)

/aws/vendedlogs/bedrock-agentcore/gateway/APPLICATION_LOGS/



Format: arn:aws:logs:region:account-id:log-group:log-group-name or /aws/vendedlogs/log-group-name

► Additional settings - optional

Cancel

Add

3. Access Gateway Logs in CloudWatch:

- Navigate to **CloudWatch** → **Logs** → **Log groups**
- Find: /aws/vendedlogs/bedrock-agentcore/gateway/APPLICATION_LOGS/customer-support-gw-[auto-generated-ID]
- Click to open:
BedrockAgentCoreGateway_ApplicationLogs

Gateway Processing

When customers interact with your agent, AgentCore Gateway processes tool requests through a **four-phase MCP protocol**:

- 1. Initialization** → Establish MCP connection with the gateway
- 2. Discovery** → Enumerate available tools for the agent
- 3. Execution** → Invoke specific tools (warranty check, web search, etc.)
- 4. Correlation** → Track multiple tool calls within the same session

Each phase generates detailed logs that help you monitor tool performance, debug MCP issues, and understand tool usage patterns.

Gateway Log Structure and Analysis

Sample Gateway Log Entry:

```

1  {
2    "resource_arn": "arn:aws:bedrock-agentcore:<region>",
3    "event_timestamp": 1756958992250,
4    "body": {
5      "id": "2",
6      "log": "Received request for tools/call method",

```



```
7      "isError": false
8    },
9    "account_id": "<account-id>",
10   "request_id": "b2474cdc-b5dd-44b0-876d-8de900414ee3"
11 }
```

Key Fields:

Field	Description
resource_arn	ARN of the Gateway resource—useful for filtering logs across multiple gateway instances.
event_timestamp	Timestamp (in milliseconds) indicating when the gateway operation occurred.
body.id	Sequential request identifier within the gateway session for tracking operation order.
body.log	Human-readable operation description detailing MCP protocol interactions.
body.isError	Boolean flag (true or false) indicating whether the operation encountered an error.
account_id	AWS account identifier for multi-account deployments.
request_id	Unique correlation ID linking gateway operations to agent requests—enables end-to-end tracing.

Complete Gateway Processing Example

Let's trace a real tool execution workflow. When a customer asks your agent "Can you check the warranty on my laptop?", you'll see this complete MCP sequence in CloudWatch Logs:

```
1  {"log": "Started processing request with requestId: 0"}
2  {"log": "Received request for initialize method"}
3  {"log": "Successfully processed request with requestId: 0"}
4  {"log": "Started processing request with requestId: 1"}
5  {"log": "Received request for tools/list method"}
6  {"log": "Successfully processed request with requestId: 1"}
7  {"log": "Started processing request with requestId: 2"}
8  {"log": "Received request for tools/call method"}
```

```

9    {"log": "Executing tool LambdaUsingSDK___check_warrant
10   {"log": "Successfully processed request with requestId

```

What This Sequence Reveals:

1. MCP Initialization

- Received request for initialize method confirms proper gateway connection
- Request ID 0 shows this is the first operation in the session
- Successful completion enables subsequent tool operations

2. Tool Discovery

- Received request for tools/list method shows agent querying available tools
- Gateway provides the list of Lab 3 tools:
check_warranty_status, web_search, get_product_info
- Request ID 1 maintains proper sequencing

3. Tool Execution

- Received request for tools/call method indicates specific tool invocation
- Executing tool
LambdaUsingSDK___check_warranty_status shows which Lab 3 tool is running
- from target ME2UI4BINR identifies the specific Lambda function you deployed

4. Performance Tracking

- Request IDs (0→1→2) show proper sequential processing
- Timestamp analysis reveals tool execution latency
- Successfully processed confirms no errors in the MCP protocol

What This Observation Tells You

- **Protocol Validation:** MCP initialization and discovery working correctly for your Lab 3 setup
- **Tool Availability:** All workshop tools are properly registered and discoverable
- **Execution Success:** Customer warranty requests are successfully routed to your Lambda function

4.5 AWS CloudTrail Audit Logging

AWS CloudTrail provides comprehensive audit logging for AgentCore API operations, creating an immutable record of who performed what actions, when, and from where. CloudTrail automatically captures the last 90 days of management events for your AWS account, including all AgentCore resource creation, updates, and configuration changes from your workshop activities.

CloudTrail complements the existing observability layers by adding the security and compliance dimension—tracking not just how your agents perform, but who manages them and how they're configured. This audit trail is essential for production environments where you need to demonstrate compliance, investigate security incidents, or troubleshoot configuration issues.

Accessing CloudTrail Event History

CloudTrail Event History is available immediately with no setup required. It provides a console-based view of recent API activity across your AWS account.

Navigate to AgentCore Events:

1. Open the CloudTrail Console

- Navigate to **CloudTrail** in the AWS Console
- Select **Event history** from the left navigation panel

2. Filter for AgentCore Activity

- Clear the default "Read only = false" filter by clicking the **X**
- Click **Add filter** → Select **Event source**
- Enter: `bedrock-agentcore.amazonaws.com`
- Click **Apply**

3. Review Your Workshop Activities

- You will see AgentCore API calls from Labs 2-4
- Events are listed with the most recent first
- Each row shows the event name, time, user, and affected resources

Tracking Your Workshop Activities

The Event History will show a complete record of your workshop progress. Here's what events correspond to each lab:

Lab	AgentCore Events	What Happened	Resources
Lab 2	CreateMemory	Created AgentCore Memory for customer preferences	CustomerSupp xxxxx
Lab 3	CreateGateway, CreateGatewayTarget	Set up shared tools via AgentCore Gateway	customersupp xxxxx
Lab 4	CreateAgentRuntime, UpdateAgentRuntime	Deployed agent to AgentCore Runtime	customer_sup xxxxx

Additional Events You Might See:

- GetMemory, ListMemories - Memory resource queries
- GetGateway, ListGateways - Gateway resource queries
- GetAgentRuntime - Runtime resource queries

Understanding Event Details

Click any event to open the detailed view and examine the complete API call record:

Key Fields Explained:

- **Event Time:** When the operation occurred (UTC or local timezone)
- **User Identity:** Your IAM user or role that performed the action
 - Shows the exact AWS principal responsible
 - Includes session context for assumed roles
- **Event Source:** Confirms bedrock-agentcore.amazonaws.com
- **Event Name:** Specific API operation (e.g., CreateGateway)
- **AWS Region:** Where the operation was performed
- **Source IP Address:** Origin of the API call
- **Request Parameters:** Input data for the API call

- Resource names, configurations, and settings
- Sensitive data is automatically redacted
- **Response Elements:** Output from the API call
 - Resource ARNs, status codes, creation timestamps
 - Error messages if the operation failed
- **Resources:** List of AWS resources affected by the operation
 - Shows the full ARN of created or modified resources

Example Event Analysis:

When you created your Memory resource in Lab 2, CloudTrail recorded:



- **Who:** Your AWS identity
- **What:** CreateMemory operation
- **When:** Exact timestamp
- **Where:** Your AWS region and source IP
- **Result:** Memory ARN and configuration details

Congratulations - you successfully completed Lab 4: Hosting and monitoring agents securely and at scale with AgentCore Runtime and Observability!

You've configured, built and deployed your customer support agent to the AgentCore Runtime for further invocations. Now you have:

- a deployed production-ready agent, that the end users can access securely
- a serverless solution which will scale automatically to meet the ever-changing demand
- a built-in monitoring and traceability of all of the agent's steps

Resources


- [Amazon Bedrock AgentCore Runtime Documentation](#) 
- [AgentCore Runtime Code Samples](#) 
- [Amazon Bedrock AgentCore Observability Documentation](#) 
- [AgentCore Observability Code Samples](#) 

Try it out

At an AWS event

If you are following the workshop via workshop studio, now go to JupyterLab in SageMaker Studio. In the JupyterLab UI navigate to `lab-04-agentcore-runtime.ipynb`

Self paced

For the complete working implementation and examples: [Deploy to Production - Use AgentCore Runtime with Observability](#) .

Ready to Continue? → Go to [Lab 5: Build User Interface - Create a customer-facing application](#)

[Previous](#)[Next](#)