



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 1: Create the Agent Prototype

Lab 1: Create the Agent Prototype

Overview

Amazon Bedrock [AgentCore](#) helps you deploying and operating AI agents securely at scale - using any framework and model. It provides you with the capability to move from prototype to production faster. In this lab, we'll create a Customer Support Agent prototype using the [Strands Agents](#) framework. This prototype will serve as your starting point for exploring the complete journey from agent prototype to production-ready solutions.

What we are building

In this lab, we'll create a **Customer Support Agent prototype** with basic capabilities. This agent will have the following local tools available:


Tool Function	Description
<code>get_return_policy()</code>	Get return policy for specific products
<code>get_product_info()</code>	Get product information
<code>web_search()</code>	Search web for updated product information

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

- How to create tools using the `@tool` decorator
- How to initialize a Strands agent with model and tools
- How to test your agent locally in a Jupyter notebook

The Workshop Journey

In subsequent labs, we'll build up from this simple agent to show a typical journey faced by agent developers when moving from

agent prototype to real business value of production agents. While doing that, we'll show how [AgentCore](#)  accelerates that journey.

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

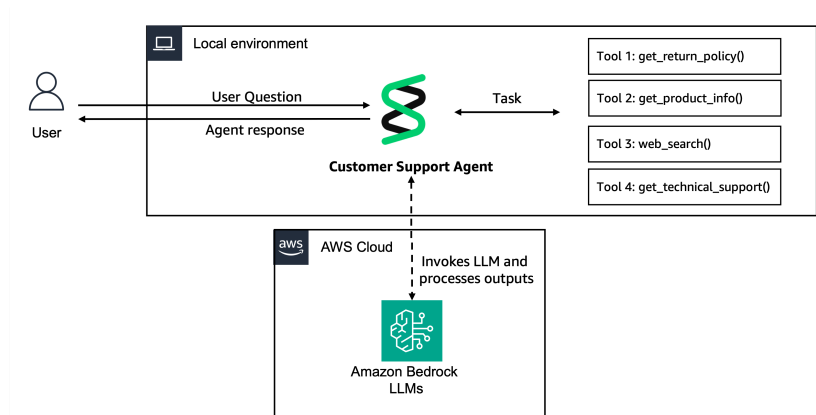
This initial prototype has several limitations that we'll address in subsequent labs:

- **No persistent memory** - Agent forgets previous conversations
- **No access to existing enterprise APIs** for real customer data
- **Local tools only** - No shared or enterprise-grade tool integration
- **No production observability** - Limited visibility into agent behavior
- **No identity management** - Cannot act on behalf of specific users
- **No scalability** - Runs only in local development environment

i Production Vision: In subsequent labs, we'll transform this prototype into a production-ready solution with AgentCore's long-term memory (LTM) that maintains rich customer context across months of interactions, enterprise API integration through Gateway, multi-user identity management, and comprehensive observability - unlocking the full power of agentic AI at enterprise scale.

i Incremental Workshop Journey: This is Lab 1 of a 5-part progressive workshop. Each lab builds upon the previous one, adding new AgentCore capabilities to your customer support agent.

Architecture for Lab1



Note: Simple Agent prototype running locally. In subsequent labs, we'll migrate this to AgentCore Runtime with shared tools, persistent memory, and production-grade observability.

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.

After loading the necessary libraries, our first step is to create Tools:

Step 1: Create Customer Support Tools

To provide more capabilities to the Agent, we can build specialized functions that can interact with external systems and data sources. Each tool represents a specific capability that allows the agent to take actions in the real world, from looking up orders to checking policies.

Principle: Each tool should have a single, well-defined responsibility (Single Responsibility Principle) and provide clear, structured outputs that the agent can reason about.

While you'll see three tools in the Python notebook, let's examine one specific example of how to create a tool in Strands.

Sample Tool: Get Return Policy

► **Code Snippet**

Step 2: Configure the Foundation Model

Foundation Model sets up the "brain" of our agent that will power reasoning and decision-making. The model configuration directly impacts your agent's behavior, cost, and performance. Each parameter serves a specific purpose in optimizing for your use case.

► **Code Snippet**

Step 3: Create and Configure the Customer Support Agent

Now, let's assemble all components into a functioning agent by combining the model, tools, and behavioral instructions and add a System Prompt.

The **system prompt** is crucial - it defines your agent's personality, capabilities, and operational guidelines. This is where you encode business rules and service standards.

► Code Snippet

Step 4: Test Your Agent

To validate that our agent works correctly, we can test it with realistic customer scenarios.

► Code Snippet

What happens when you call `agent()`:

1. **Query analysis:** Agent analyzes the customer's question
2. **Tool selection:** Agent determines which tool(s) to use (if any)
3. **Tool execution:** Agent calls the appropriate tool with correct parameters
4. **Response synthesis:** Agent combines tool results with its knowledge to create a helpful response
5. **Quality check:** Agent ensures the response meets the standards in the system prompt

You've now built a foundational customer support agent that showcases core AI capabilities - from understanding queries to executing actions - demonstrating the essential building blocks needed for production-ready AI assistants.

Congratulations on Building Your Agent Prototype!

You've successfully created the foundation of a customer support agent! This prototype demonstrates the core building blocks of AI agents - understanding queries, selecting appropriate tools, and synthesizing helpful responses.

Where you are in the journey: You've completed the "easy" part - building a basic agent prototype. The real challenges come next: making this agent production-ready with proper memory, observability, security, and scalability.

What's next: In the following labs, we'll enhance this agent to deliver exceptional customer experiences:

- **Customer-based conversation history and context** - Remember every interaction to provide seamless support
- **Customer preference learning** - Understand individual needs and adapt responses accordingly
- **Personalized interactions** - Tailor every conversation to each customer's unique situation


We'll show you how AgentCore services make these customer-obsessed capabilities achievable at enterprise scale.

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-01-create-an-agent.ipynb`

Self paced

For the complete working implementation and examples: [Create the Agent Prototype](#) .

Ready to Continue? [Lab 2: Add Memory to Your Agent](#) →

Enhance your customer support agent with conversational context and long-term customer preferences using AgentCore Memory services.

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