Diving Deep into Bedrock AgentCore

- Diving Deep into Bedrock AgentCore
- Prerequisites
- Amazon Bedrock AgentCore Fundamentals
- AgentCore Runtime
  - Hosting Runtime Agent
  - Hosting MCP server
  - Advanced Concepts for Agentcore Runtime
- AgentCore Gateway
- AgentCore Identity
- AgentCore Memory
  - AgentCore Memory: Short-Term Memory
  - AgentCore Memory: Long-Term Memory Strategies
- AgentCore Tools
  - AgentCore 1P Tool - AgentCore Code Interpreter
  - AgentCore 1P Tool - AgentCore Browser
    - AgentCore Browser - Nova Act SDK
    - AgentCore Browser - Browser-Use
- AgentCore Observability
  - AgentCore Observability for Runtime hosted Agent
  - Observability for Non-Runtime hosted Agents

AgentCore

AgentCore Documentation

AWS account access

Workshop catalog in AWS Builder Center

Content preferences

Exit event

Event ends in 17 hours 59 minutes.

Hosting Runtime Agent

Overview

Amazon Bedrock AgentCore Runtime enables developers to deploy and run AI Agents in a secure and scalable serverless environment. The Bedrock AgentCore Runtime provides a secure, serverless hosting environment necessary for deploying and running AI agents and tools without managing infrastructure. By offering a framework-agnostic and model-flexible runtime, the AgentCore Runtime enables organizations to leverage their existing agent code and models, rather than being limited to a specific development framework or AI architecture.

How it works

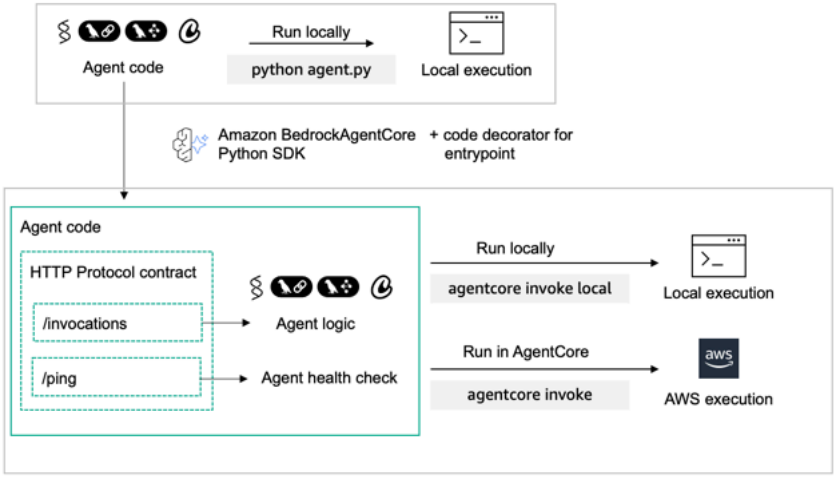
The Amazon Bedrock AgentCore Python SDK acts as a wrapper that:

- **Transforms** your agent code into AgentCore's standardized protocols
- **Handles** HTTP and MCP server infrastructure automatically
- **Lets you focus** on your agent's core functionality
- **Supports** two protocol types:
  - **HTTP Protocol**: Traditional request/response REST API endpoints
  - **MCP Protocol**: Model Context Protocol for tools and agent servers

High Level Architecture

When hosting agents, the SDK automatically:

- Hosts your agent on port `8080`
- Provides two key endpoints:
  - `/invocations`: Primary agent interaction (JSON input → JSON/SSE output)
  - `/ping`: Health check for monitoring



Getting Started

First, you need to implement an AI Agent using the framework of your choice. In this example, we implement a simple AI Agent using the Strands Agent framework:

```python
from strands import Agent, tool
from strands_tools import calculator
import argparse
import json
from strands.models import BedrockModel

@tool
def weather():
    """ Get the weather """
    return "sunny"

model_id = "us.anthropic.claude-3-7-sonnet-20250219-v1:0"
model = BedrockModel(
    model_id=model_id,
)
agent = Agent(
    model=model,
    tools=[calculator, weather],
    system_prompt="You're a helpful assistant. You can perform simple math calculations and tell
)

def strands_agent_bedrock(payload):
    """
    Invoke the agent with a payload
    """
    user_input = payload.get("prompt")
    response = agent(user_input)
    return response.message['content'][0]['text']

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("payload", type=str)
    args = parser.parse_args()
    response = strands_agent_bedrock(json.loads(args.payload))
    print(response)
```

Prepare the Agent for AgentCore Runtime

The Amazon Bedrock AgentCore Python SDK provides a lightweight wrapper that helps you deploy your agent functions as HTTP services compatible with Amazon Bedrock AgentCore.

You can convert **your existing agent function** into an Amazon Bedrock AgentCore-compatible service with just **four steps**:

1. Import the Runtime App with `from bedrock_agentcore.runtime import BedrockAgentCoreApp`
2. Initialize the App in our code with `app = BedrockAgentCoreApp()`
3. Decorate the invocation function with the `@app.entrypoint` decorator
4. Let AgentCoreRuntime control the running of the agent with `app.run()`

```python
# strands_claude.py
from strands import Agent, tool
from strands_tools import calculator # Import the calculator tool
import argparse
import json
from bedrock_agentcore.runtime import BedrockAgentCoreApp
from strands.models import BedrockModel

app = BedrockAgentCoreApp()

# Create a custom tool
@tool
def weather():
    """ Get weather """ # Dummy implementation
    return "sunny"


model_id = "us.anthropic.claude-sonnet-4-20250514-v1:0"
model = BedrockModel(
    model_id=model_id,
)
agent = Agent(
    model=model,
    tools=[calculator, weather],
    system_prompt="You're a helpful assistant. You can do simple math calculation, and tell the weather."
)

@app.entrypoint
def strands_agent_bedrock(payload):
    """
    Invoke the agent with a payload
    """
    user_input = payload.get("prompt")
    print("User input:", user_input)
    response = agent(user_input)
    return response.message['content'][0]['text']

if __name__ == "__main__":
    app.run()
```

Invoke your Agent

You can invoke the agent using the `InvokeAgentRuntime` operation:

```python
import boto3
import json

# Initialize the Bedrock AgentCore client
agent_core_client = boto3.client('bedrock-agentcore', region_name="us-east-1")


# Prepare the payload
payload = json.dumps({"prompt": prompt}).encode()

# Invoke the agent
response = agent_core_client.invoke_agent_runtime(
    agentRuntimeArn=agent_arn,
    runtimeSessionId=session_id,
    payload=payload
)
```

Try it out

At an AWS Event

In the JupyterLab UI, navigate to `01-AgentCore-runtime/01-hosting-agent/01-strands-with-bedrock-model`

Self-paced

Here's are notebooks that lets you try out the above and extend the patterns to other frameworks and models

Example	Framework	Model	Description
strands-with-bedrock-model	Strands Agents	Amazon Bedrock	Basic agent hosting with AWS native models
langgraph-with-bedrock-model	LangGraph	Amazon Bedrock	LangGraph agent workflows

Example	Framework	Model	Description
strands-with-openai-model	Strands Agents	OpenAI	Integration with external LLM providers

Previous Next