

How to use amazon bedrock as agentcore

First install AWS CLI using the below documentations :-

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

Now create aws account and there create user **agentcore-ai** and then assign it a group where select many policies:-

1. [AdministratorAccess](#)
2. [AmazonBedrockFullAccess](#)
3. [AWSCodeBuildAdminAccess](#)
4. [BedrockAgentCoreFullAccess](#)

Now go to user you created and there generate access key now

Go to terminal and

1. Check using the below command :- ``aws --version``
2. Now configure aws using the below command with access id, secret access key, region.
3. ``aws sts get-caller-identity`` to get the entered details.
4. ``aws s3 ls`` to get aws s3 list
5. Now we check using ``agentcore --help``.
6. Go to the project and activate environment
7. Now use the below command for existing file ``agentcore configure -e ./01_agentcore_runtime.py``
8. Add grq key ``agentcore launch --env GROQ_API_KEY=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``
9. Now give prompt and check :- `agentcore invoke '{"prompt": "Tell me about romaing activations"}'`
10. `agentcore status`

Here's a **refined, interview-ready and practical version** of your content. I've corrected commands, improved structure, and added best practices.

How to Use Amazon Bedrock with AgentCore (Step-by-Step)

1) Install AWS CLI

Install the AWS Command Line Interface.

Follow the official documentation:

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

After installation, verify:

```
aws --version
```

2) Create AWS Account and IAM User

1. Log in to your AWS Console.
2. Go to **IAM (Identity and Access Management)**.
3. Create a new user:
 - Username: **agentcore-ai**
 - Access type: **Programmatic access**
4. Create or attach a group with these policies:

Recommended (for testing or learning):

- AdministratorAccess
- AmazonBedrockFullAccess
- AWSCodeBuildAdminAccess
- BedrockAgentCoreFullAccess

 **Best Practice (Production):**

Instead of AdministratorAccess, create a **custom least-privilege policy**.

3) Generate Access Keys

1. Go to the created user.
 2. Open **Security credentials**.
 3. Create an **Access key**.
 4. Save:
 - Access Key ID
 - Secret Access Key
-

4) Configure AWS CLI

In terminal, run:

```
aws configure
```

Enter:

AWS Access Key ID: <your-access-key>

AWS Secret Access Key: <your-secret-key>

Default region: us-east-1 (or your region)

Default output format: json

5) Verify AWS Configuration

Check identity:

```
aws sts get-caller-identity
```

List S3 buckets:

```
aws s3 ls
```

6) Install and Check AgentCore

Check if AgentCore is installed:

```
agentcore --help
```

If not installed, install it (example):

```
pip install agentcore
```

7) Activate Project Environment

Navigate to your project:

```
cd your-project-folder
```

Activate virtual environment:

Linux/Mac

```
source venv/bin/activate
```

Windows

```
venv\Scripts\activate
```

8) Configure AgentCore Runtime

If you have an existing agent file:

```
agentcore configure -e ./01_agentcore_runtime.py
```

This command links your runtime script to AgentCore.

9) Launch Agent with Environment Variables

Example using a GROQ API key:

```
agentcore launch --env GROQ_API_KEY=your_api_key_here
```

 Never hardcode or share real API keys in commands or code.

10) Invoke the Agent

Send a test prompt:

```
agentcore invoke '{"prompt": "Tell me about roaming activations"}'
```

11) Check Agent Status

agentcore status

(agentcore-crash-course) motherson@motherson-Precision-3660:~/Documents/Agentcore\$ agentcore invoke '{"prompt": "activate it for dubai"}'

```
┌────────────────────────────────────────────────────────────────────────────────── agentcore_lang_runtime
│
│ Session: 31dc707d-dde9-4e3c-af63-b9cf815c0daf
│ Request ID: 5ec65c34-1a35-4db7-b5a5-4b868d7493f7
│ ARN:
│ arn:aws:bedrock-agentcore:ap-south-1:008859907571:runtime/agentcore_lang_run |
│ time-ZqJ58r6AQW
│ Logs: aws logs tail
│ /aws/bedrock-agentcore/runtimes/agentcore_lang_runtime-ZqJ58r6AQW-DEFAULT |
│ --log-stream-name-prefix "2026/02/16/[runtime-logs]" --follow
│ aws logs tail
│ /aws/bedrock-agentcore/runtimes/agentcore_lang_runtime-ZqJ58r6AQW-DEFAULT |
│ --log-stream-name-prefix "2026/02/16/[runtime-logs]" --since 1h
│ GenAI Dashboard:
│ https://console.aws.amazon.com/cloudwatch/home?region=ap-south-1#gen-ai-obs |
│ rvability/agent-core
└──────────────────────────────────────────────────────────────────────────────────
```

Response:

```
{"result": "***Activating your SIM for use in Dubai**\n\n1. **Insert the SIM**\n\n - Power off your phone, insert the new SIM card, and power it back\non.\n\n2. **Complete the KYC (Know-Your-Customer) process** \n - Open the app\nor go to the web portal and upload a government-issued ID. \n - Verify your\nphone number ownership (usually via a one-time code sent to the number).\n\n3.\n**Restart the device** \n - After KYC is cleared, restart your phone. The\nnetwork registration will finish automatically.\n\n4. **Enable roaming (if\nyou're using a roaming pack)** \n - In your phone's settings, go to *Mobile\ndata* → *Roaming* and turn it on. \n - Purchase a Dubai-specific roaming pack\nthrough the app or website before you travel.\n\n5. **Check coverage** \n -\nOnce the SIM is activated and roaming is enabled, you should see the local\nnetwork icon (e.g., "4G" or "5G") in the status bar.\n\nIf you encounter any\nissues, contact customer support with your SIM serial number and the error
```

```
message you see."}
```

```
(agentcore-crash-course) motherson@motherson-Precision-3660:~/Documents/Agentcore$
```

```
agentcore invoke '{"prompt": "which country am i talking about"}'
```

```
|----- agentcore_lang_runtime
```

```
|  
Session: 31dc707d-dde9-4e3c-af63-b9cf815c0daf |  
Request ID: 6e06823c-0040-435b-a413-49592f467124 |  
ARN: |  
arn:aws:bedrock-agentcore:ap-south-1:008859907571/runtime/agentcore_lang_run |  
time-ZqJ58r6AQW |  
Logs: aws logs tail |  
/aws/bedrock-agentcore/runtimes/agentcore_lang_runtime-ZqJ58r6AQW DEFAULT |  
--log-stream-name-prefix "2026/02/16/[runtime-logs]" --follow |  
    aws logs tail |  
/aws/bedrock-agentcore/runtimes/agentcore_lang_runtime-ZqJ58r6AQW DEFAULT |  
--log-stream-name-prefix "2026/02/16/[runtime-logs]" --since 1h |  
GenAI Dashboard: |  
https://console.aws.amazon.com/cloudwatch/home?region=ap-south-1#gen-ai-obse |  
rvability/agent-core |
```

Response:

```
{
  "result": "I'm not sure which country you're referring to. Could you provide a bit more context or describe the topic you're asking about? That will help me give you the correct answer."
}
```

```
(agentcore-crash-course) motherson@motherson-Precision-3660:~/Documents/Agentcore$
```

Now lets setup the memory —

On amazon bedrock agent click on memory and click on create memory:-

Create memory

Memory detail

Memory name

Valid characters are a-z, A-Z, 0-9, and _ (underscore). The name can have up to 48 characters. Character count: 12/48.

Short-term memory (raw event) expiration

Set the short-term (raw event) memory duration for the agent's interactions. Events older than the specified duration will expire and no longer be stored.

days

Specify a duration between 7 and 365 days.

Additional configurations - *optional*

Long-term memory extraction strategies - *optional*

If you need long-term memory for context recall across sessions, you can setup memory extraction strategies to extract the relevant memory from the raw events. Use built-in strategies for quick setup, use built-in strategies with override to specify models and prompt templates, or self-managed strategies to fully manage memory processing pipeline. You can configure up to 6 strategies. [Learn more](#)

Which strategy to choose?

Built-in strategies

Info

Built-in strategies to extract long-term memories from raw interactions.
Summarize interactions to preserve critical context and key insights.

Extract general factual knowledge, concepts and meanings from raw conversations in a context-independent format.

Extract user behavior patterns from raw conversations.

Transforms events into structured episodes and enables the agent to learn from past actions using reflections. Reflections are analyzed insights from past experiences that help agents improve future performance.

Built-in strategy with override

Info

Override memory processing through foundation model and prompt templates.


Add strategy


Self-managed strategy


Use AgentCore memory for event storage with custom triggers. Define memory processing logic in your own environment.


Add self-managed strategy


(agentcore-crash-course) motherson@motherson-Precision-3660:~/Documents/Agentcore\$
agentcore configure -e ./02_agentcore_memory.py
Configuring Bedrock AgentCore...
✓ Using file: 02_agentcore_memory.py


 Inferred agent name: 02_agentcore_memory
Press Enter to use this name, or type a different one (alphanumeric without '-')
Agent name [02_agentcore_memory]: agentcore_with_memory
✓ Using agent name: agentcore_with_memory


 Detected dependency file: pyproject.toml
Press Enter to use this file, or type a different path (use Tab for autocomplete):
Path or Press Enter to use detected dependency file: pyproject.toml
✓ Using requirements file: pyproject.toml

 Deployment Configuration
Select deployment type:
1. Direct Code Deploy (recommended) - Python only, no Docker required
2. Container - For custom runtimes or complex dependencies
Choice [1]: 2
✓ Deployment type: Container

 Execution Role
Press Enter to auto-create execution role, or provide execution role ARN/name to use existing
Execution role ARN/name (or press Enter to auto-create):
✓ Will auto-create execution role

 ECR Repository
Press Enter to auto-create ECR repository, or provide ECR Repository URI to use existing
ECR Repository URI (or press Enter to auto-create):
✓ Will auto-create ECR repository

 Authorization Configuration
By default, Bedrock AgentCore uses IAM authorization.
Configure OAuth authorizer instead? (yes/no) [no]: no
✓ Using default IAM authorization

 Request Header Allowlist
Configure which request headers are allowed to pass through to your agent.
Common headers: Authorization, X-Amzn-Bedrock-AgentCore-Runtime-Custom-*
Configure request header allowlist? (yes/no) [no]: no

- ✓ Using default request header configuration

Configuring BedrockAgentCore agent: agantcore_with_memory

Memory Configuration

Tip: Use `--disable-memory` flag to skip memory entirely

Existing memory resources found:

1. customercareagentmemory

ID: customercareagentmemory-eOBzZWDzYW

Options:

- Enter a number to use existing memory
- Press Enter to create new memory
- Type 's' to skip memory setup

Your choice: 1

- ✓ Using existing memory: `customer care agent memory-eOBzZWDzYW`

Using existing memory resource: customercareagentmemory-eOBzZWDzYW

Network mode: PUBLIC

⚠️ Platform mismatch: Current system is 'linux/amd64' but Bedrock AgentCore requires 'linux/arm64', so local builds won't work.

Please use default launch command which will do a remote cross-platform build using code build. For deployment other options and workarounds, see:

<https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/getting-started-custom.html>

 Generated Dockerfile:

```
/home/motherson/Documents/Agentcore/.bedrock_agentcore/agantcore_with_memory/Doc
kerfile
```

Changing default agent from 'agentcore_lang_runtime' to 'agantcore_with_memory'

Configuration Success

Agent Details	
Agent Name:	agantcore_with_memory
Deployment:	container
Region:	ap-south-1
Account:	008859907571
Configuration	
Execution Role:	Auto-create
ECR Repository:	Auto-create
Network Mode:	Public
ECR Repository:	Auto-create
Authorization:	IAM (default)

Memory: Short-term + Long-term memory (30-day retention)



Config saved to:

/home/motherson/Documents/Agentcore/.bedrock_agentcore.yaml

Next Steps:

agentcore deploy

Below is a **clean, professional, developer-ready document** you can share internally or externally.

It includes **setup, memory configuration, and a working Python example** for Amazon Bedrock AgentCore with memory.

Amazon Bedrock AgentCore with Memory

Step-by-Step Setup and Python Example

This document explains how to:

1. Set up AWS and AgentCore
2. Deploy an agent
3. Enable memory
4. Use memory in a Python agent
5. Test the agent

1. Install AWS CLI

Follow the official AWS documentation:

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

Verify installation:

aws --version

2. Create AWS Account and IAM User

1. Open AWS Console
2. Go to **IAM (Identity and Access Management)**
3. Create a new user

User details:

Username: agentcore-ai

Access type: Programmatic access

Attach policies

For testing:

- AdministratorAccess
- AmazonBedrockFullAccess
- AWSCodeBuildAdminAccess
- BedrockAgentCoreFullAccess



Production best practice:

Use least-privilege policies instead of AdministratorAccess.

3. Generate Access Keys

1. Open IAM → Users → **agentcore-ai**
2. Go to **Security credentials**
3. Create **Access key**
4. Save:

Access Key ID

Secret Access Key

4. Configure AWS CLI

Run:

```
aws configure
```

Enter:

AWS Access Key ID: <your-access-key>

AWS Secret Access Key: <your-secret-key>

Default region: ap-south-1

Default output format: json

5. Verify AWS Configuration

Check identity:

```
aws sts get-caller-identity
```

List S3 buckets:

```
aws s3 ls
```

6. Install AgentCore

Inside your project:

```
pip install agentcore
```

Verify:

```
agentcore --help
```

7. Activate Virtual Environment

Linux/Mac:

```
source venv/bin/activate
```

Windows:

```
venv\Scripts\activate
```

8. Create a Basic Agent (Python Example)

Create file:

```
01_agentcore_runtime.py
```

Basic Agent without memory

```
from agentcore import app
from langchain_groq import ChatGroq
import os

model = ChatGroq(
    model="llama3-70b-8192",
    api_key=os.environ["GROQ_API_KEY"]
)

@app.entrypoint()
def handler(payload: dict):
    prompt = payload.get("prompt", "")
    response = model.invoke(prompt)
    return {"result": response.content}
```

9. Configure AgentCore Runtime

```
agentcore configure -e ./01_agentcore_runtime.py
```

10. Launch the Agent

```
agentcore launch --env GROQ_API_KEY=your_api_key
```

11. Test the Agent

```
agentcore invoke '{"prompt": "Tell me about roaming activations"}'
```

Check status:

```
agentcore status
```

12. Enable Memory in Bedrock AgentCore

Step 1: Create Memory Resource

In AWS Console:

1. Go to **Amazon Bedrock**
2. Open **AgentCore**
3. Click **Memory**
4. Click **Create memory**

Memory Settings

Memory name:

```
customercareagentmemory
```

Short-term memory expiration:

7–365 days (recommended: 30 days)

Long-term memory strategies (optional)

You can enable:

- Conversation summarization
 - Knowledge extraction
 - User behavior patterns
 - Episodic memory with reflections
-

13. Create Agent with Memory

Create new file:

02_agentcore_memory.py

Python Agent with Memory

```
from agentcore import app
from langchain_groq import ChatGroq
import os

model = ChatGroq(
    model="llama3-70b-8192",
    api_key=os.environ["GROQ_API_KEY"]
)

@app.entrypoint()
def handler(payload: dict, memory=None):
    """
    payload: incoming request
    memory: injected memory interface from AgentCore
    """

    user_prompt = payload.get("prompt", "")

    # Retrieve memory context
```

```

memory_context = ""
if memory:
    past_events = memory.search(user_prompt, top_k=3)
    memory_context = "\n".join([e["text"] for e in past_events])

# Combine memory + current prompt
final_prompt = f"""
Past context:
{memory_context}

User question:
{user_prompt}
"""

response = model.invoke(final_prompt)

# Store interaction in memory
if memory:
    memory.store({
        "text": f"User: {user_prompt} | Assistant: {response.content}"
    })

return {"result": response.content}

```

14. Configure Agent with Memory

Run:

```
agentcore configure -e ./02_agentcore_memory.py
```

When prompted:

- Choose existing memory
- or
- Create new memory

Example:

Select memory: 1

Using existing memory: customercareagentmemory

15. Deploy the Agent

agentcore deploy

16. Launch the Memory-Enabled Agent

agentcore launch --env GROQ_API_KEY=your_api_key

17. Test Memory Behavior

First request

agentcore invoke '{"prompt": "activate it for dubai"}'

Follow-up request

agentcore invoke '{"prompt": "which country am i talking about"}'

With memory enabled, the agent should correctly respond:

You were talking about Dubai.

18. Useful Commands

Check agent status:

agentcore status

View logs:

```
aws logs tail <log-group> --follow
```

Best Practices

Security

- Never hardcode API keys
- Use environment variables
- Use least-privilege IAM roles

Performance

- Use short-term memory for session context
- Use long-term memory for user behavior or knowledge

Production Setup

- Private network mode
 - Custom IAM roles
 - Observability via CloudWatch
-

Project Structure Example

agentcore-project/

```
|
|— venv/
|— 01_agentcore_runtime.py
|— 02_agentcore_memory.py
|— pyproject.toml
|— README.md
```