

```
        "aws:SourceAccount": "123456789012"  
      },  
      "ArnLike": {  
        "aws:SourceArn": "arn:aws:bedrock-agentcore:us-  
east-1:123456789012:*"  
      }  
    }  
  }  
]  
}
```

Get started with AgentCore Runtime

You can use the following tutorials to get started with Amazon Bedrock AgentCore Runtime.

The [Amazon Bedrock AgentCore Starter Toolkit](#) is a Command Line Interface (CLI) that simplifies the infrastructure setup for containerizing and deploying an agent to an AgentCore Runtime.

Topics

- [Get started with the Amazon Bedrock AgentCore starter toolkit](#)
- [Get started without the starter toolkit](#)
- [Get started with Amazon Bedrock AgentCore Runtime direct code deployment](#)
- [Get started with WebSocket streaming in AgentCore Runtime](#)

Get started with the Amazon Bedrock AgentCore starter toolkit

This tutorial shows you how to use the Amazon Bedrock AgentCore [starter toolkit](#) to deploy an agent to an Amazon Bedrock AgentCore Runtime.

The starter toolkit is a Command Line Interface (CLI) toolkit that you can use to deploy AI agents to an Amazon Bedrock AgentCore Runtime. You can use the toolkit with popular Python agent frameworks, such as LangGraph or [Strands Agents](#). This tutorial uses Strands Agents.

For information about the HTTP protocol that the agent uses, see [HTTP protocol contract](#).

Topics

- [Prerequisites](#)

- [Step 1: Set up project and install dependencies](#)
- [Step 2: Create your agent](#)
- [Step 3: Test locally](#)
- [Step 4: Configure your agent](#)
- [Step 5: Enable observability for your agent](#)
- [Step 6: Deploy to Amazon Bedrock AgentCore Runtime](#)
- [Step 7: Test your deployed agent](#)
- [Step 8: Invoke your agent programmatically](#)
- [Step 9: Clean up](#)
- [Find your resources](#)
- [Common issues and solutions](#)
- [Advanced options \(Optional\)](#)

Prerequisites

Before you start, make sure you have:

- **AWS Account** with credentials configured. To configure your AWS credentials, see [Configuration and credential file settings in the AWS CLI](#).
- **Python 3.10+** installed
- [Boto3](#) installed
- **AWS Permissions:** To create and deploy an agent with the starter toolkit, you must have appropriate permissions. For information, see [Use the starter toolkit](#).
- **Model access:** Anthropic Claude Sonnet 4.0 [enabled](#) in the Amazon Bedrock console. For information about using a different model with the Strands Agents see the *Model Providers* section in the [Strands Agents SDK](#) documentation.

Step 1: Set up project and install dependencies

Create a project folder and install the required packages:

```
mkdir agentcore-runtime-quickstart
cd agentcore-runtime-quickstart
```

```
python3 -m venv .venv
source .venv/bin/activate
```

Note

On Microsoft Windows, use: **`.venv\Scripts\activate`**

Upgrade pip to the latest version:

```
pip install --upgrade pip
```

Install the following required packages:

- **bedrock-agentcore** - The Amazon Bedrock AgentCore SDK for building AI agents
- **strands-agents** - The [Strands Agents](#) SDK
- **bedrock-agentcore-starter-toolkit** - The Amazon Bedrock AgentCore starter toolkit

```
pip install bedrock-agentcore strands-agents bedrock-agentcore-starter-toolkit
```

Verify installation:

```
agentcore --help
```

Step 2: Create your agent

Create a source file for your agent code named `my_agent.py`. Add the following code:

```
from bedrock_agentcore import BedrockAgentCoreApp
from strands import Agent

app = BedrockAgentCoreApp()
agent = Agent()

@app.entrypoint
def invoke(payload):
    """Your AI agent function"""
    user_message = payload.get("prompt", "Hello! How can I help you today?")
```

```
result = agent(user_message)
return {"result": result.message}

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

Create `requirements.txt` and add the following:

```
bedrock-agentcore
strands-agents
```

Step 3: Test locally

Make sure port 8080 is free before starting. See *Port 8080 in use (local only)* in [Common issues and solutions](#).

Open a terminal window and start your agent with the following command:

```
python my_agent.py
```

Test your agent by opening another terminal window and enter the following command:

```
curl -X POST http://localhost:8080/invocations \
-H "Content-Type: application/json" \
-d '{"prompt": "Hello!"}'
```

Success: You should see a response like `{"result": "Hello! I'm here to help..."}`. In the terminal window that's running the agent, enter **Ctrl+C** to stop the agent.

Step 4: Configure your agent

Configure and deploy your agent to AWS using the starter toolkit. The toolkit automatically creates the IAM execution role, container image, and Amazon Elastic Container Registry repository needed to host the agent in an AgentCore Runtime. By default the toolkit hosts the agent in an AgentCore Runtime that is in the `us-west-2` AWS Region.

Configure the agent using the default values:

```
agentcore configure -e my_agent.py
```

- The `-entrypoint` flag specifies the entrypoint file for your agent (the Python file containing your agent code)
- This command creates configuration for deployment to AWS
- Accept the default values unless you have specific requirements
- The configuration information is stored in a hidden file named `.bedrock_agentcore.yaml`

(Optional) Use a different AWS Region

By default, the starter toolkit deploys to the `us-west-2` AWS Region. To use a different Region:

```
agentcore configure -e my_agent.py -r us-east-1
```

Step 5: Enable observability for your agent

[Amazon Bedrock AgentCore Observability](#) helps you trace, debug, and monitor agents that you host in Amazon Bedrock AgentCore Runtime. First enable CloudWatch Transaction Search by following the instructions at [Enabling Amazon Bedrock AgentCore runtime observability](#). To observe your agent, see [View observability data for your Amazon Bedrock AgentCore agents](#).

Step 6: Deploy to Amazon Bedrock AgentCore Runtime

Host your agent in AgentCore Runtime:

```
agentcore launch
```

This command:

- Builds your container using AWS CodeBuild (no Docker required locally)
- Creates necessary AWS resources (ECR repository, IAM roles, etc.)
- Deploys your agent to Amazon Bedrock AgentCore Runtime
- Configures CloudWatch logging

In the output from `agentcore launch` note the following:

- The Amazon Resource Name (ARN) of the agent. You need it to invoke the agent with the [InvokeAgentRuntime](#) operation.
- The location of the logs in Amazon CloudWatch Logs

If the deployment fails check for [common issues](#). For other deployment options, see [Deployment modes](#).

Step 7: Test your deployed agent

Test your deployed agent:

```
agentcore invoke '{"prompt": "tell me a joke"}'
```

If you see a joke in the response, your agent is now running in an Amazon Bedrock AgentCore Runtime and can be invoked. If not, check for [common issues](#).

Step 8: Invoke your agent programmatically

You can invoke the agent using the AWS SDK [InvokeAgentRuntime](#) operation. To call `InvokeAgentRuntime`, you need the ARN of the agent that you noted in Step 6: Deploy to Amazon Bedrock AgentCore Runtime. You can also get the ARN from the `bedrock_agentcore:` section of the `.bedrock_agentcore.yaml` (hidden) file that the toolkit creates. Use the following boto3 (AWS SDK) code to invoke your agent. Replace *Agent ARN* with the ARN of your agent. Make sure that you have `bedrock-agentcore:InvokeAgentRuntime` permissions. Create a file named `invoke_agent.py` and add the following code:

```
import json
import uuid
import boto3

agent_arn = "Agent ARN"
prompt = "Tell me a joke"

# Initialize the Amazon Bedrock AgentCore client
agent_core_client = boto3.client('bedrock-agentcore')

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

# Invoke the agent
response = agent_core_client.invoke_agent_runtime(
    agentRuntimeArn=agent_arn,
    runtimeSessionId=str(uuid.uuid4()),
    payload=payload,
    qualifier="DEFAULT"
```

```
)

content = []
for chunk in response.get("response", []):
    content.append(chunk.decode('utf-8'))
print(json.loads(''.join(content)))
```

Open a terminal window and run the code with the following command:

```
python invoke_agent.py
```

If successful, you should see a joke in the response. If the call fails, check the logs that you noted in [Step 6: Deploy to Amazon Bedrock AgentCore Runtime](#).

Note

If you plan on integrating your agent with OAuth, you can't use the AWS SDK to call `InvokeAgentRuntime`. Instead, make a HTTPS request to `InvokeAgentRuntime`. For more information, see [the section called "Authenticate and authorize with Inbound Auth and Outbound Auth"](#).

Step 9: Clean up

If you no longer want to host the agent in the AgentCore Runtime, use the `destroy` command to delete the AWS resources that the starter toolkit created for you.

```
agentcore destroy
```

Find your resources

After deployment, view your resources in the AWS Console:

Resource locations

Resource	Location
Agent Logs	CloudWatch → Log groups → /aws/bedrock-agentcore/runtimes/{agent-id}-DEFAULT

Resource	Location
Container Images	ECR → Repositories → bedrock-agentcore-{agent-name}
Build Logs	CodeBuild → Build history
IAM Role	IAM → Roles → Search for "BedrockAgentCore"

Common issues and solutions

Common issues and solutions when getting started with the Amazon Bedrock AgentCore starter toolkit. For more troubleshooting information, see [Troubleshoot Amazon Bedrock AgentCore Runtime](#).

Permission denied errors

Verify your AWS credentials and permissions:

- Verify AWS credentials: **aws sts get-caller-identity**
- Check you have the required policies attached
- Review caller permissions policy for detailed requirements

Docker not found warnings

You can ignore this warning:

- **Ignore this!** Default deployment uses CodeBuild (no Docker needed)
- Only install Docker/Finch/Podman if you want to use `--local` or `--local-build` flags

Model access denied

Enable model access in the Bedrock console:

- Enable Anthropic Claude 4.0 in the Bedrock console
- Make sure you're in the correct AWS Region (us-west-2 by default)

CodeBuild build error

Check build logs and permissions:

- Check CodeBuild project logs in AWS console
- Verify your caller permissions include CodeBuild access

Port 8080 in use (local only)

Find and stop processes that are using port 8080:

Use `lsof -ti:8080` to get a list of process using port 8080.

Use `kill -9 PID` to stop the process. Replace *PID* with the process ID.

Region mismatch

Verify the AWS Region with `aws configure get region` and make sure resources are in same Region

Advanced options (Optional)

The starter toolkit has advanced configuration options for different deployment modes and custom IAM roles. For more information, see [Runtime commands for the starter toolkit](#).

Deployment modes

Choose the right deployment approach for your needs:

Default: CodeBuild + Cloud Runtime (RECOMMENDED)

Suitable for production, managed environments, teams without Docker:

```
agentcore launch # Uses CodeBuild (no Docker needed)
```

Local Development

Suitable for development, rapid iteration, debugging:

```
agentcore launch --local # Build and run locally (requires Docker/Finch/Podman)
```

Hybrid: Local Build + Cloud Runtime

Suitable for teams with Docker expertise needing build customization:

```
agentcore launch --local-build # Build locally, deploy to cloud (requires Docker/Finch/Podman)
```

Note

Docker is only required for `--local` and `--local-build` modes. The default mode uses AWS CodeBuild.

Custom execution role

Use an existing IAM role:

```
agentcore configure -e my_agent.py --execution-role arn:aws:iam::111122223333:role/MyRole
```

Why ARM64?

Amazon Bedrock AgentCore Runtime requires ARM64 containers (AWS Graviton). The toolkit handles this automatically:

- **Default (CodeBuild):** Builds ARM64 containers in the cloud - no Docker needed
- **Local with Docker:** Only containers built on ARM64 machines will work when deployed to agentcore runtime

Get started without the starter toolkit

You can create a AgentCore Runtime agent without the starter toolkit. Instead you can use a combination of command line tools to configure and deploy your agent to an AgentCore Runtime.

This tutorial shows how to deploy a custom agent without using the starter toolkit. A custom agent is an agent built without using the AgentCore Python SDK. In this tutorial, the custom agent is built using FastAPI and Docker. The custom agent follows the [AgentCore Runtime requirements](#), meaning the agent must expose `/invocations` POST and `/ping` GET endpoints and be packaged in a Docker container. Amazon Bedrock AgentCore requires ARM64 architecture for all deployed agents.

Note

You can also use this approach for agents that you build with the AgentCore Python SDK.