

Inbound Auth

Inbound Auth with Amazon Cognito

Introduction

AgentCore Identity lets you validate inbound access (Inbound Auth) for users and applications calling agents or tools in an AgentCore Runtime or validate access to AgentCore Gateway targets. It also provides secure outbound access (Outbound Auth) from an agent to external services or a Gateway target. It integrates with your existing identity providers (such as Amazon Cognito) while enforcing permission boundaries for agents acting independently or on behalf of users (via OAuth).

Inbound Auth validates callers attempting to invoke agents or tools, whether they're hosted in AgentCore Runtime, AgentCore Gateway, or in other environments. Inbound Auth works with IAM (SigV4 credentials) or with OAuth authorization.

By default, Amazon Bedrock AgentCore uses IAM credentials, meaning user requests to the agent are authenticated with the user's IAM credentials. If you use OAuth, you will need to specify the following when configuring your AgentCore Runtime resources or AgentCore Gateway endpoints:

- OAuth discovery server Url — A string that must match the pattern ^.+/.well-known/openid-configuration\$ for OpenID Connect discovery URLs
- Allowed audiences — List of allowed audiences for JWT tokens
- Allowed clients — List of allowed client identifiers

If you use the AgentCore CLI, you can specify the type of authorization (and OAuth discovery server) for an AgentCore Runtime when you use the **configure** command. You can also use the CreateAgentRuntime operation and Amazon Bedrock AgentCore console. If you are creating a Gateway, you use the CreateGateway operation, or the console.

Before the user can use the agent, the client application must have the user authenticate with the OAuth authorizer. Your client receives a bearer token which it then passes to the agent in an invocation request. Upon receipt the agent validates the token with the authorization server before allowing access.

Why is this Important?

Inbound Auth is a critical security feature that validates callers attempting to invoke agents or tools, whether they're hosted in AgentCore Runtime, AgentCore Gateway, or other environments. It ensures that only authenticated and authorized users can access your AI agents, providing a secure foundation for enterprise AI applications.

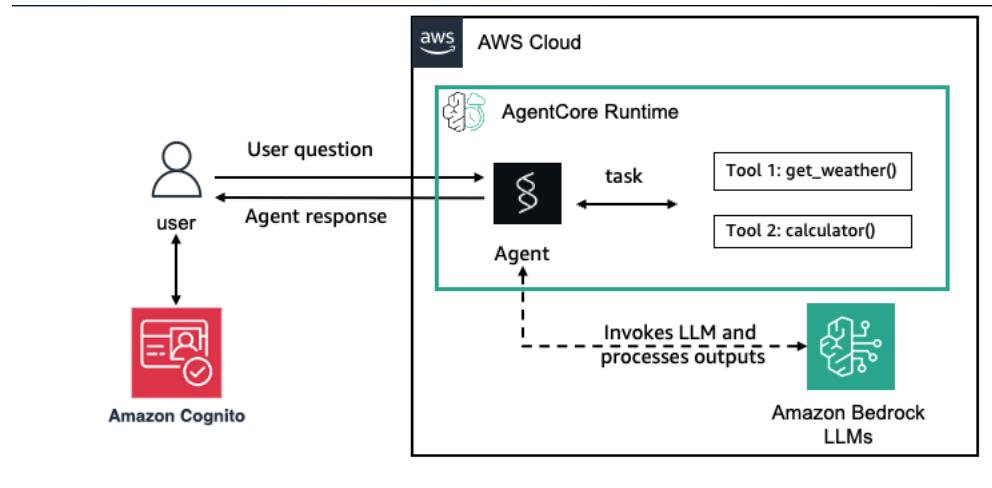
In production environments, AI agents often handle sensitive data and perform critical business operations. Without proper authentication:

- **Unauthorized access** could lead to data breaches or misuse of agent capabilities
- **Lack of audit trails** makes it difficult to track who accessed what resources
- **No user context** prevents agents from providing personalized experiences
- **Security compliance** requirements cannot be met

Inbound Auth solves these challenges by integrating with enterprise identity providers and enforcing strict access controls.

In this lab, you will learn how to host your existing agent using Amazon Bedrock AgentCore Runtime with Inbound Auth using the Cognito user pool.

High-Level Architecture



Architecture Flow:

- User Authentication**: Client application authenticates user with Amazon Cognito
- Token Issuance**: Cognito returns a JWT access token to the client
- Agent Invocation**: Client includes the JWT token when invoking the agent
- Token Validation**: AgentCore Runtime validates the token with Cognito's discovery endpoint
- Secure Access**: Only valid tokens allow access to the agent's capabilities

Prerequisites & Setup

Environment Requirements

Before starting this lab, ensure you have:

- **Python 3.10+** installed on your system
- **AWS credentials** configured with appropriate permissions
- **Docker** running for containerization
- **Amazon Bedrock AgentCore SDK** installed
- **Strands Agents** framework installed

API Keys & Configuration Setup

1. AWS Credentials Configuration

Ensure your AWS credentials are properly configured:

```

1  # Option 1: Using AWS CLI
2  aws configure
3
4  # Option 2: Using environment variables
5  export AWS_ACCESS_KEY_ID=your_access_key
6  export AWS_SECRET_ACCESS_KEY=your_secret_key
7  export AWS_DEFAULT_REGION=us-west-2

```



2. Amazon Cognito Setup

You'll need to create a Cognito User Pool for authentication. The [setup script ↗](#) will handle this automatically, but here's what it creates:

Cognito User Pool Components:

- **User Pool:** Container for user accounts and authentication settings
- **App Client:** Configuration for your application to interact with the User Pool
- **Test User:** A sample user account for testing authentication
- **Access Token:** JWT token for API authentication

 **Note**

The Cognito access_token is valid for 2 hours only. If the access_token expires you can vend another access_token by running the # Authenticate User cmdlet from setup_cognito_uer_pool.sh

Important Values to Save:

- Cognito Discovery URL (format: `https://cognito-idp.{region}.amazonaws.com/{userPoolId}/.well-known/openid-configuration`)
- App Client ID (unique identifier for your application)
- Access Token (JWT token for testing)

3. Environment Variable Configuration

Create a `.env` file or set environment variables:

```
1 # Cognito Configuration
2 COGNITO_DISCOVERY_URL=<your-cognito-discovery-url>
3 COGNITO_CLIENT_ID=<your-app-client-id>
4 COGNITO_ACCESS_TOKEN=<your-test-token>
5
6 # AWS Region Configuration
7 AWS_REGION=us-west-2
```



AgentCore Feature Implementation

Core Inbound Auth Concepts

AgentCore Identity's Inbound Auth provides several authentication mechanisms:

- **IAM (SigV4):** Default AWS credential-based authentication
- **OAuth/JWT:** Token-based authentication with identity providers

Key Configuration Parameters:

- `discoveryUrl`: OpenID Connect discovery endpoint
- `allowedClients`: List of authorized client applications
- `allowedAudiences`: Valid token audiences (optional)

Implementation with AgentCore Runtime

1. Strands with Amazon Bedrock model

Let's start with our Strands Agent we created in the [01-AgentCore-runtime ↗](#) tutorial and configure it with Inbound Auth that uses Amazon Cognito as the Identity Provider.

Key AgentCore Identity Components Explained:

- `BedrockAgentCoreApp()`: Runtime application wrapper that handles HTTP server infrastructure
- `@app.entrypoint`: Decorator marking the function as the agent's main entry point

- **Authentication Integration:** The runtime automatically handles token validation before reaching your code

2. Deploying the agent to AgentCore Runtime

The CreateAgentRuntime operation supports comprehensive configuration options, letting you specify container images, environment variables and encryption settings. You can also configure protocol settings (HTTP, MCP) and authorization mechanisms to control how your clients communicate with the agent.

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

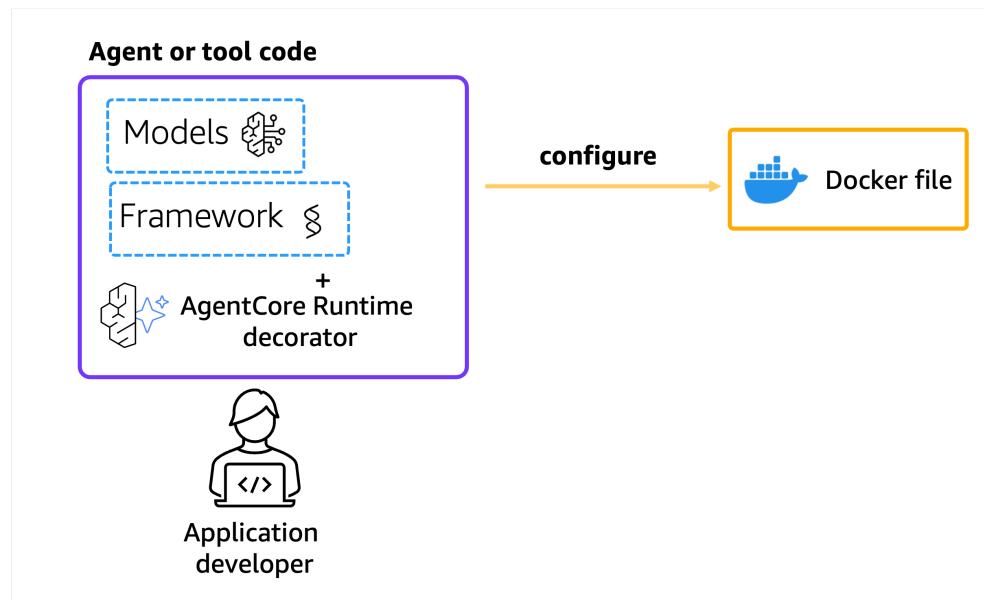
In this lab, we will use the Amazon Bedrock AgentCore Python SDK to easily package our artifacts and deploy them to AgentCore runtime.

Ensure you have the runtime role created. If it is not you can create it using the [utils ↗](#) function pre-developed for you.

Update AgentCore Runtime deployment

Next we will use our starter toolkit to configure the AgentCore Runtime deployment with an entrypoint, the execution role we just created and a requirements file. We will also configure the starter kit to auto create the Amazon ECR repository on launch.

During the configure step, your docker file will be generated based on your application code



⚠️ Important

Update the Cognito Discovery url and the Cognito App client id from the previous steps.

▼ Update the existing Runtime configuration to the code below

```

1  from bedrock_agentcore_starter_toolkit import Runtime
2  from boto3.session import Session
3  boto_session = Session()
4  region = boto_session.region_name
5  region
6
7  #Update the Cognito Discovery url below. You can get the discovery url from the "Provision a Cog
8  discovery_url = '<your-cognito-user-pool-discovery-url>'
```



```

10 #Update the Cognito App Client id below. You can get the Cognito App client from the "Provision"
11 client_id = '<your-cognito-app-client-id>'
12
13 agentcore_runtime = Runtime()
14
15 response = agentcore_runtime.configure(
16     entrypoint="strands_claude.py",
17     execution_role=agentcore_iam_role['Role']['Arn'],
18     auto_create_ecr=True,
19     requirements_file="requirements.txt",
20     region=region,
21     agent_name=agent_name,
22     authorizer_configuration={
23         "customJWTAuthorizer": {
24             "discoveryUrl": discovery_url,

```

  amit-sharma ▾

Diving Deep into Bedrock AgentCore

 Event ends in 20 hours 12 minutes. 

Diving Deep into Bedrock AgentCore

► Prerequisites

Amazon Bedrock AgentCore Fundamentals

► AgentCore Runtime

► AgentCore Gateway

▼ AgentCore Identity

Inbound Auth

▼ Outbound Auth

API Key Credential Provider

User-Delegated Access with Cognito and 3-Legged OAuth flow with GitHub

User-Delegated Access with Cognito and 3-Legged OAuth flow with Google

▼ 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

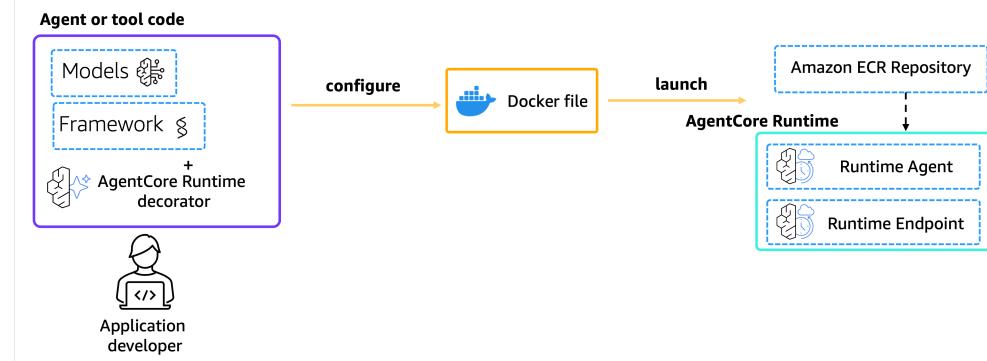
AgentCore ↗

AgentCore Documentation ↗

Configuration Parameters Explained:

- `customJWTAuthorizer`: JWT Bearer Token Authentication, You can configure your agent runtime to accept JWT bearer tokens by providing authorizer configuration during agent creation.
- `discoveryUrl`: A string that must match the pattern `^./well-known/openid-configuration$` for OpenID Connect discovery URLs
- `allowedClients`: A list of permitted client identifiers that will be validated against the `client_id` claim in the JWT token

Launching the updated agent to AgentCore Runtime



```

launch_result = agentcore_runtime.launch()
launch_result

```

Now that we've deployed the AgentCore Runtime, let's check for its deployment status before we test out the Authentication implementation

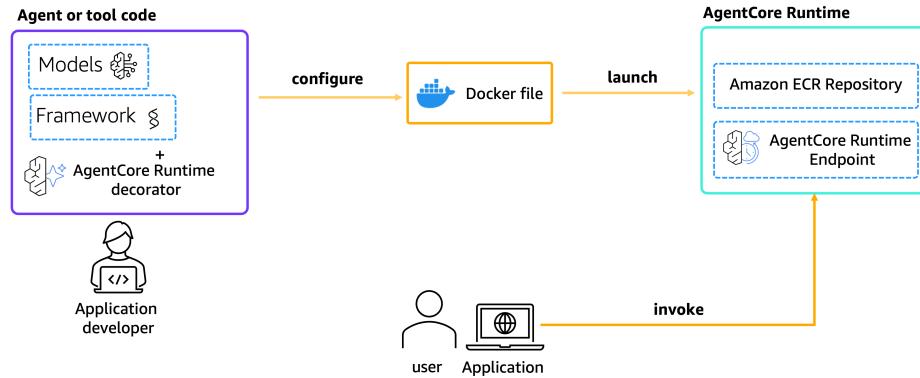
3. Invoking AgentCore Runtime

AWS account access

Workshop catalog in AWS Builder Center [↗](#)

Content preferences

Exit event



Invoking AgentCore Runtime without authorization

Finally, we can invoke our AgentCore Runtime with a payload.

```

1  invoke_response = agentcore_runtime.invoke({"prompt": "How is the weather now?"})
2  invoke_response

```



When you invoke without authorization you will see an error that says "AccessDeniedException: An error occurred (AccessDeniedException) when calling the InvokeAgentRuntime operation: Agent is configured for a different authorization token type".

Invoking AgentCore Runtime with authorization

Lets invoke the agent with the right authorization token type. In our case, it will be the Cognito access token. Copy the access token from the cell "Provision a Cognito User Pool"

```

1  #Update the Cognito access token here. Copy the access token from the cell "Provision a Cognito User Pool"
2  cognito_bearer_token=<cognito-access-token>
3  invoke_response = agentcore_runtime.invoke({"prompt": "How is the weather now?"}, bearer_token=cognito_bearer_token)
4  invoke_response

```



Authentication Validation Process:

1. **Token Extraction:** Runtime extracts JWT from Authorization header
2. **Signature Verification:** Validates token signature using Cognito's public keys
3. **Claims Validation:** Checks expiration, audience, and issuer claims
4. **Client Authorization:** Verifies client ID against allowed clients list
5. **Request Processing:** Only valid tokens reach your agent code

Different Approaches & Framework Options

Framework-Agnostic Implementation

AgentCore Identity works with multiple agentic frameworks:

Strands Agents (Current Example)

```

1  from strands import Agent, tool
2  from strands.models import BedrockModel
3
4  agent = Agent(
5      model=BedrockModel(model_id="claude-sonnet-4"),
6
7

```



```
tools=[calculator, weather]
)
```

Custom Framework Integration (For Information Only, Not covered in the current lab)

```
1  from bedrock_agentcore.runtime import BedrockAgentCoreApp
2
3  app = BedrockAgentCoreApp()
4
5  @app.entrypoint
6  def custom_agent(payload):
7      # Your custom agent logic with authentication
8      return process_request(payload)
```



Authentication Method Options

OAuth with Different Providers

Amazon Cognito (Current Example)

```
1  authorizer_configuration={
2      "customJWTAuthorizer": {
3          "discoveryUrl": "https://cognito-idp.us-west-2.amazonaws.com/{pool-id}/.well-known/openid
4          "allowedClients": ["your-client-id"]
5      }
6  }
```



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 03-AgentCore-identity/03-Inbound Auth example/inbound_auth_runtime_with_strands_and_bedrock_models.ipynb

Self-paced

Here's a notebook that lets you try out the above: [Inbound Auth with Cognito Example ↗](#).

Congratulations!

You have successfully implemented Inbound Auth for your AgentCore Runtime agent using Amazon Cognito. You learned how to:

- Set up Amazon Cognito as an identity provider
- Configure AgentCore Runtime with OAuth authentication
- Deploy and test an authenticated agent
- Understand the security benefits of Inbound Auth

Your completed implementation provides enterprise-grade security that scales with your organization's needs while maintaining the flexibility to integrate with various identity providers and agentic frameworks.

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