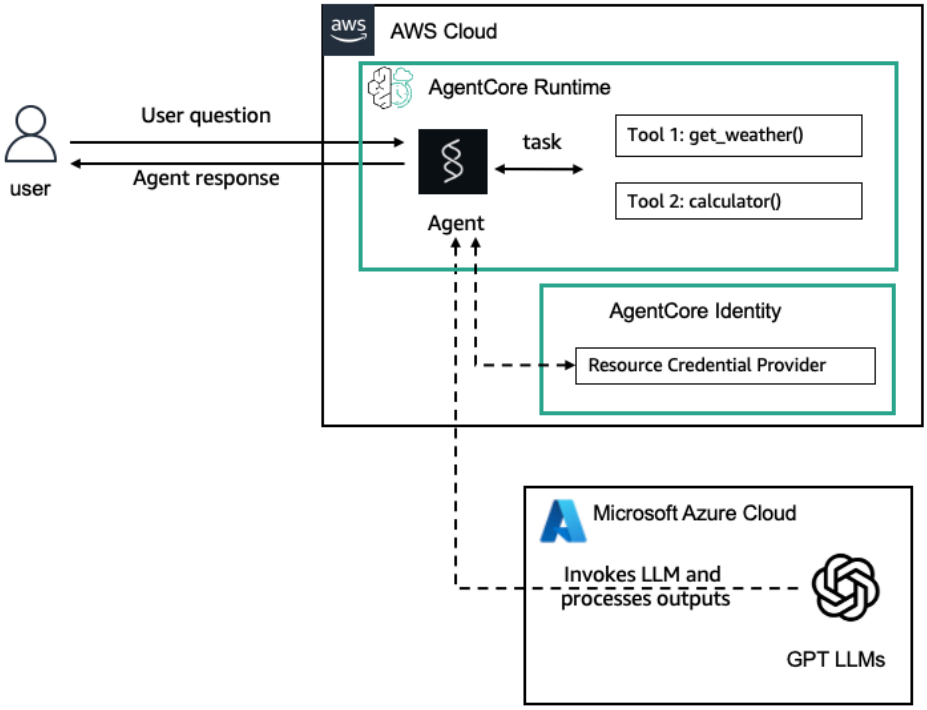
API Key Credential Provider

Introduction

In this Lab, we will focus on implementing Outbound Auth using the API Key credential provider in AgentCore. We'll demonstrate this authentication pattern by configuring an agent to securely access OpenAI services through AgentCore's credential management system. You will learn how to set up an API Key credential provider to store OpenAI credentials and configure your agent code to leverage this secure authentication mechanism.

High-Level Architecture



Architecture Flow:

1. **Agent Invocation**: Client invokes the agent through AgentCore Runtime
2. **Credential Request**: Agent requests API key from AgentCore Identity credential provider
3. **Secure Retrieval**: AgentCore Identity returns the encrypted API key
4. **External Service Call**: Agent uses the API key to authenticate with external service (Azure OpenAI)
5. **Response Processing**: Agent processes the external service response and returns results

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
- **External service API key** (Azure OpenAI, OpenAI, or similar for testing)

API Keys & Configuration Setup

1. Azure OpenAI API Key Setup

To obtain an Azure OpenAI API key:

1. **Create Azure Account**: Sign up at portal.azure.com
2. **Create OpenAI Resource**:
   - Navigate to "Create a resource" → "AI + Machine Learning" → "Azure OpenAI"
   - Choose subscription, resource group, and region
   - Select pricing tier (Standard S0 recommended for testing)
3. **Get API Credentials**:
   - Go to your OpenAI resource → "Keys and Endpoint"
   - Copy the API Key, Endpoint URL, and API Version
4. **Deploy Model**:
   - Navigate to "Model deployments" → "Create new deployment"
   - Select GPT-4 or GPT-3.5-turbo model
   - Note the deployment name

Required Values:

- `AZURE_API_KEY`: Your Azure OpenAI API key
- `AZURE_API_BASE`: Your Azure OpenAI endpoint (e.g., `https://your-resource.openai.azure.com/`)
- `AZURE_API_VERSION`: API version (e.g., `2023-12-01-preview`)

2. Alternative: OpenAI API Key Setup

For standard OpenAI API:

1. **Create OpenAI Account**: Sign up at platform.openai.com
2. **Generate API Key**:
   - Go to "API Keys" section
   - Click "Create new secret key"
   - Copy and securely store the key
3. **Set Usage Limits**: Configure billing and usage limits for cost control

3. Environment Variable Configuration

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

```
# Azure OpenAI Configuration
AZURE_API_KEY=your_azure_openai_key
AZURE_API_BASE=https://your-resource.openai.azure.com/
AZURE_API_VERSION=2023-12-01-preview

# Or for standard OpenAI
OPENAI_API_KEY=your_openai_key

# AWS Configuration
AWS_REGION=us-west-2
```

4. Resource credential providers

This is a component that agent code uses to retrieve credentials of downstream resource servers (e.g., Google, GitHub) to access them, e.g., fetch emails from Gmail, add a meeting to Google Calendar. It removes the heavy-lifting of agent developers implementing 2LO and 3LO OAuth2 orchestration flows across end-users, agent code, and external authorization servers. AgentCore provides both a custom OAuth2 credential provider and a list of built-in providers such Google, GitHub, Slack, Salesforce with authorization server endpoint and provider-specific parameters pre-filled.

Bedrock AgentCore Identity provides OAuth2 and API Key Credential Providers for agent developers to authenticate with external resources that support OAuth2 or API key. In the following example, we will walk you through configuring an API Key credential provider. An agent can then use the API Key credential provider to retrieve the API key for any agent operations. Please refer to the documentation for the other credential providers.

▼ **Here is an example of creating a API Key resource credential provider.**

```python
from bedrock_agentcore.services.identity import IdentityClient
identity_client = IdentityClient(region="us-west-2")

api_key_provider = identity_client.create_api_key_credential_provider({
    "name": "APIKey-provider-name",
    "apiKey": "<my-api-key>" # Replace it with the API key you obtain from the external applicati
})
print(api_key_provider)
```

5. Retrieving access tokens or API keys from the Resource credential provider.

Here is an example of retrieving the API key from the API Key credential provider. The agent can use the API key to interact with services like a LLM or other services that use an API key configuration. In order to retrieve credentials like access_token or API key from the credential provider.

▼ **you can decorate your function as shown below.**

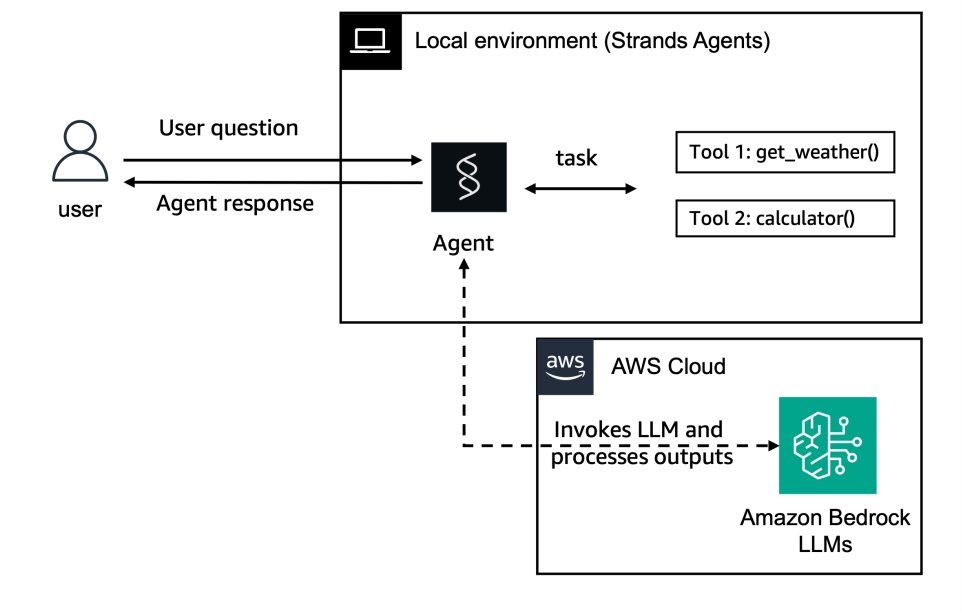
```python
import asyncio
from bedrock_agentcore.identity.auth import requires_access_token, requires_api_key

@requires_api_key(
    provider_name="APIKey-provider" # replace with your own credential provider name
)
async def need_api_key(*, api_key: str):
    print(f'received api key for async func: {api_key}')

await need_api_key(api_key="")
```

For more information on managing credential providers with AgentCore Identity, review

Creating your agents and experimenting locally

Before we deploy our agents to AgentCore Runtime, let's develop and run them locally for experimentation purposes.

The architecture here will look as following:



Preparing your agent for deployment on AgentCore Runtime and use the Resource Credential Provider

Let's now deploy our agents to AgentCore Runtime. To do so we need to:

- Import the Runtime App with from bedrock_agentcore.runtime import BedrockAgentCoreApp
- Initialize the App in our code with app = BedrockAgentCoreApp()
- Decorate the invocation function with the @app.entrypoint decorator
- Let AgentCoreRuntime control the running of the agent with app.run()
- Retrieve the openAI key from the resource credential provider created earlier

Strands Agents with OpenAI model

▼ **Let's start with our Strands Agent using the GPT 4.1 mini model. All the others will work exactly the same.**

```
import asyncio
from bedrock_agentcore.identity.auth import requires_access_token, requires_api_key
from strands import Agent, tool
from strands_tools import calculator
import argparse
import json
from strands.models.litellm import LiteLLMModel
import os
from bedrock_agentcore.runtime import BedrockAgentCoreApp

AZURE_API_KEY_FROM_CREDS_PROVIDER = ""


@requires_api_key(
    provider_name="openai-apikey-provider" # replace with your own credential provider name
)
async def need_api_key(*, api_key: str):
    global AZURE_API_KEY_FROM_CREDS_PROVIDER
    print(f'received api key for async func: {api_key}')
    AZURE_API_KEY_FROM_CREDS_PROVIDER = api_key

# Don't print empty value at module level - will print in entrypoint function

app = BedrockAgentCoreApp()

# API key will be set dynamically in the entrypoint function
#Update the below configuration with your Azure API Key details.
os.environ["AZURE_API_BASE"] = "<YOUR_API_BASE>"
os.environ["AZURE_API_VERSION"] = "<YOUR_API_VERSION>"
```

```python

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

# Global agent variable
agent = None

@app.entrypoint
async def strands_agent_open_ai(payload):
    """
    Invoke the agent with a payload
    """
    global AZURE_API_KEY_FROM_CREDS_PROVIDER, agent

    print(f"Entrypoint called with AZURE_API_KEY_FROM_CREDS_PROVIDER: '{AZURE_API_KEY_FROM_CREDS_

    # Get API key if not already retrieved
    if not AZURE_API_KEY_FROM_CREDS_PROVIDER:
        print("Attempting to retrieve API key...")
        try:
            await need_api_key(api_key="")
            print(f"API key retrieved: '{AZURE_API_KEY_FROM_CREDS_PROVIDER}'")
            os.environ["AZURE_API_KEY"] = AZURE_API_KEY_FROM_CREDS_PROVIDER
            print("Environment variable AZURE_API_KEY set")
        except Exception as e:
            print(f"Error retrieving API key: {e}")
            raise
    else:
        print("API key already available")

    # Initialize agent after API key is set
    if agent is None:
        print("Initializing agent with API key...")
        model = "azure/gpt-4.1-mini"
        litellm_model = LiteLLMModel(
            model_id=model, params={"max_tokens": 32000, "temperature": 0.7}
        )

        agent = Agent(
            model=litellm_model,
            tools=[calculator, weather],
            system_prompt="You're a helpful assistant. You can do simple math calculation, and t
        )
        print("Agent initialized successfully")

    user_input = payload.get("prompt")
    print(f"User input: {user_input}")

    try:
        response = agent(user_input)
        print(f"Agent response: {response}")
        return response.message['content'][0]['text']
    except Exception as e:
        print(f"Error in agent processing: {e}")
        raise

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

Key AgentCore Components Explained:

- **@requires_api_key**: Decorator that securely retrieves API keys from credential providers
- **IdentityClient**: Client for managing credential providers and authentication flows
- **Async Pattern**: Required for credential retrieval operations to maintain performance
- **Global State Management**: Efficient credential reuse across multiple agent invocations

What happens behind the scenes? When you use BedrockAgentCoreApp, it automatically:

- Creates an HTTP server that listens on the port 8080
- Implements the required /invocations endpoint for processing the agent's requirements
- Implements the /ping endpoint for health checks (very important for asynchronous agents)
- Handles proper content types and response formats
- Manages error handling according to the AWS standards

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.

> **Note**
> Operations best practice is to package code as container and push to ECR using CI/CD pipelines and IaC

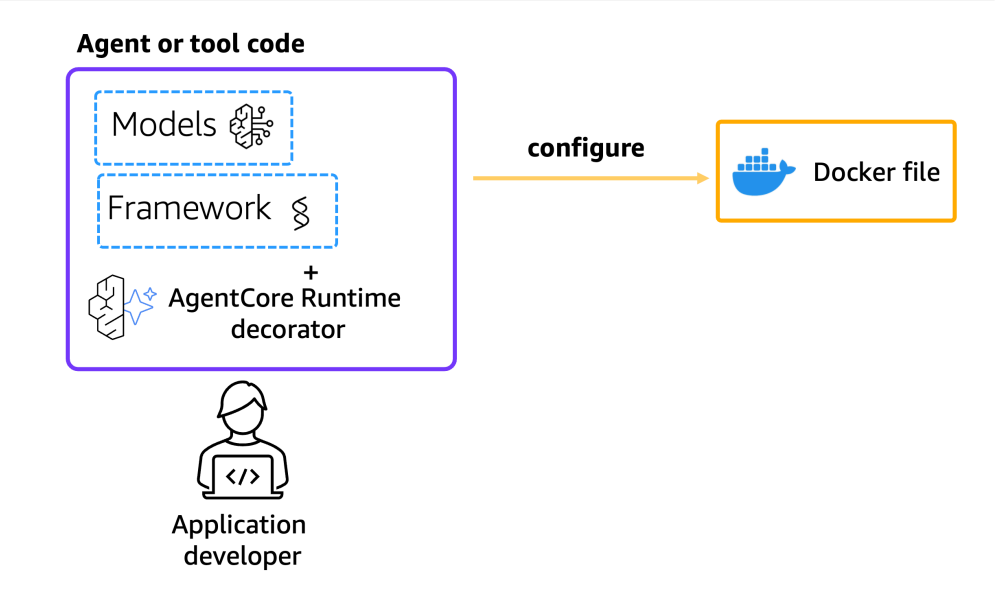
In this lab, we will use the Amazon Bedrock AgentCode Python SDK to easily package your artifacts and deploy them to AgentCore runtime.

Creating runtime role Before starting, let's create an IAM role for our AgentCore Runtime. We will do so using the utils function pre-developed for you.

Configure 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

> **Note**
> The authorizer_configuration is configured for Inbound Auth with Cognito.

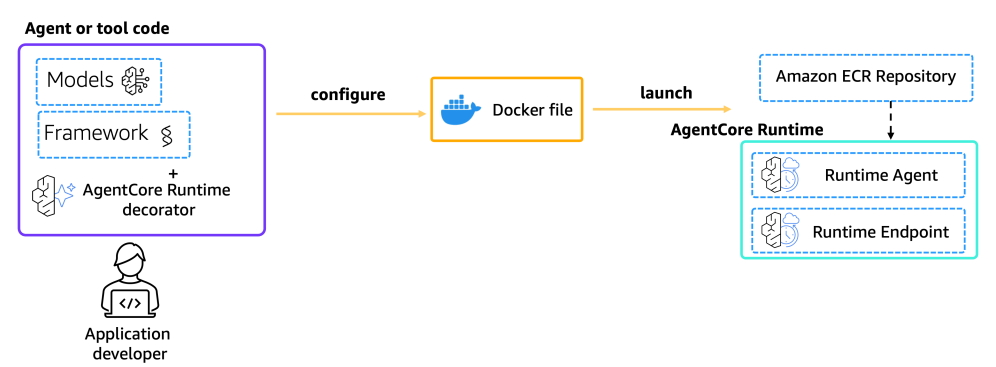


▼ **Here is an example of creating a AgentCore Runtime.**

```
from bedrock_agentcore_starter_toolkit import Runtime
from boto3.session import Session
import boto3
import json
boto_session = Session()
region = boto_session.region_name
```

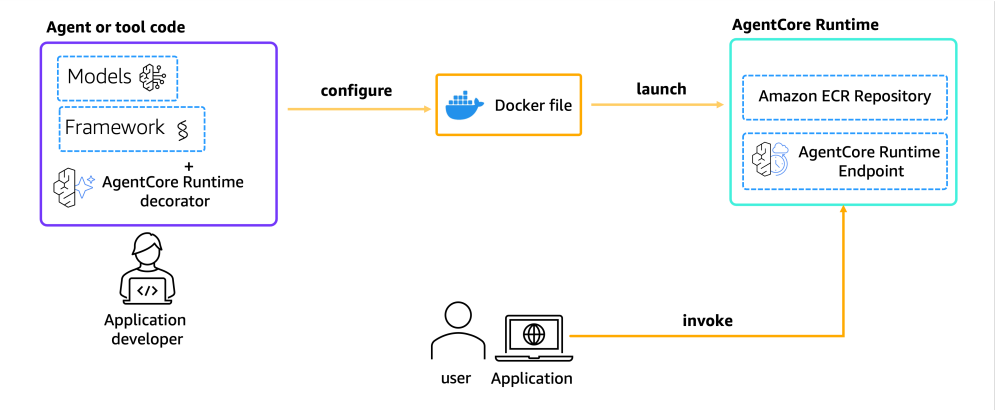
```
region

agentcore_runtime = Runtime()
agent_name="strands_agents_openai"

response = agentcore_runtime.configure(
    entrypoint="strands_agents_openai.py",
    execution_role=agentcore_iam_role['Role']['Arn'],
    auto_create_ecr=True,
    agent_name=agent_name,
    requirements_file="requirements.txt",
    region=region
)
response
```

Launching agent to AgentCore Runtime Now that we've got a docker file, let's launch the agent to the AgentCore Runtime. This will create the Amazon ECR repository and the AgentCore Runtime




```
launch_result = agentcore_runtime.launch()
launch_result
```

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




```
invoke_response = agentcore_runtime.invoke({"prompt": "Hello"}, user_id="userid_1234567890")
invoke_response
```

We can now process our invocation results to include it in an application

Invoking AgentCore Runtime with boto3

Now that your AgentCore Runtime was created you can invoke it with any AWS SDK.

▼ **For instance, you can use the boto3 invoke_agent_runtime method for it.**

```
agent_arn = launch_result.agent_arn
agentcore_client = boto3.client(
    'bedrock-agentcore',
    region_name=region
)

boto3_response = agentcore_client.invoke_agent_runtime(
    agentRuntimeArn=agent_arn,
    runtimeUserId="userid_1234567890",
    qualifier="DEFAULT",
    payload=json.dumps({"prompt": "How much is 2X2?"})
)
if "text/event-stream" in boto3_response.get("contentType", ""):
    content = []
    for line in boto3_response["response"].iter_lines(chunk_size=1):
        if line:
            line = line.decode("utf-8")
            if line.startswith("data: "):
                line = line[6:]
                logger.info(line)
                content.append(line)
    display(Markdown("\n".join(content)))
else:
    try:
        events = []
        for event in boto3_response.get("response", []):
```



```
            events = [f"Error reading EventStream: {e}"]
        display(Markdown(json.loads(events[0].decode("utf-8"))))
```

Framework-Agnostic Implementation

AgentCore Outbound Auth works with multiple agentic frameworks:

Strands Agents (Current Example)

```
from strands import Agent, tool
from strands.models.litellm import LiteLLMModel

agent = Agent(
    model=LiteLLMModel(model_id="azure/gpt-4"),
    tools=[calculator, weather]
)
```

Credential Provider

API Key Providers (Current Example)

```
# Simple API key storage and retrieval
api_key_provider = identity_client.create_api_key_credential_provider({
    "name": "service-api-key",
    "apiKey": "your-service-api-key"
})
```

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/04-Outbound Auth`

`example/runtime_with_strands_and_openai_models.ipynb`

Self-paced

Here's a notebook that lets you try out the above: Outbound Auth with API Key Provider Example.

Congratulations!

You have successfully implemented Outbound Auth for your AgentCore Runtime agent using the API Key credential provider. You learned how to:

- Create and configure resource credential providers
- Use decorators to retrieve credentials securely
- Deploy agents that can access external services
- Understand the security benefits of centralized credential management

Your completed implementation demonstrates how AgentCore Identity simplifies secure external service integration while maintaining the highest security standards. This foundation can be extended to support OAuth flows, multiple external services, and complex multi-agent scenarios.

Previous Next