Hosting MCP server

Overview

Amazon Bedrock AgentCore Runtime lets you deploy Model Context Protocol (MCP) servers without managing infrastructure. This guide covers how to host MCP tools using the Amazon Bedrock AgentCore Python SDK.

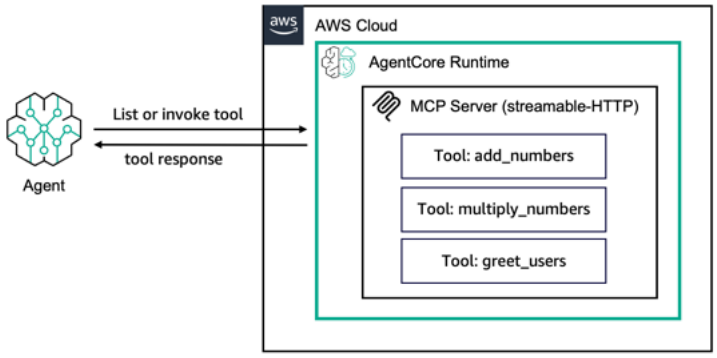
The SDK wraps your agent functions as MCP-compatible servers, handling protocol details so you can focus on logic. It supports both HTTP and MCP protocols for tool and agent interaction.

For tool hosting, the SDK uses **Stateless Streamable HTTP** with the `Mcp-Session-Id` header for session isolation. Servers must run on `0.0.0.0:8000` and expose `/mcp`.

AgentCore automatically adds `Mcp-Session-Id` to support stateless, session-aware communication. The `InvokeAgentRuntime` API passes through all payloads, enabling seamless MCP message proxying.

High Level Architecture

We will use a very simple MCP server with 3 tools: `add_numbers`, `multiply_numbers` and `greet_users`



1. **Agent Initialization:** The agent starts and is configured to interact with the configured tools via the MCP Server hosted on AWS using the AgentCore Runtime.
2. **Tool Discovery or Invocation:** The agent sends a request to the MCP Server (using streamable-HTTP) to either list available tools or invoke a specific tool (e.g., add_numbers, multiply_numbers, or greet_users).
3. **Tool Execution:** The MCP Server, running inside the AgentCore Runtime, processes the request and routes it to the appropriate tool implementation.
4. **Tool Response Returned:** Once the selected tool completes its logic (e.g., performs a computation or generates a greeting), the MCP Server returns the response to the agent.
5. **Agent Processes Result:** The agent receives the result from the MCP Server and uses it to continue its reasoning or return a final response to the user or system.

Getting Started

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

Create MCP Server

Key MCP Server Components Explained:

- **FastMCP**: Creates an MCP server that can host your tools
- **@mcp.tool()**: Decorator that turns your Python functions into MCP tools
- **stateless_http=True**: Required for AgentCore Runtime compatibility
- **Tools**: Three simple tools demonstrating different types of operations

Create Local Testing Client

▶ **Code Snippet**

Testing Locally

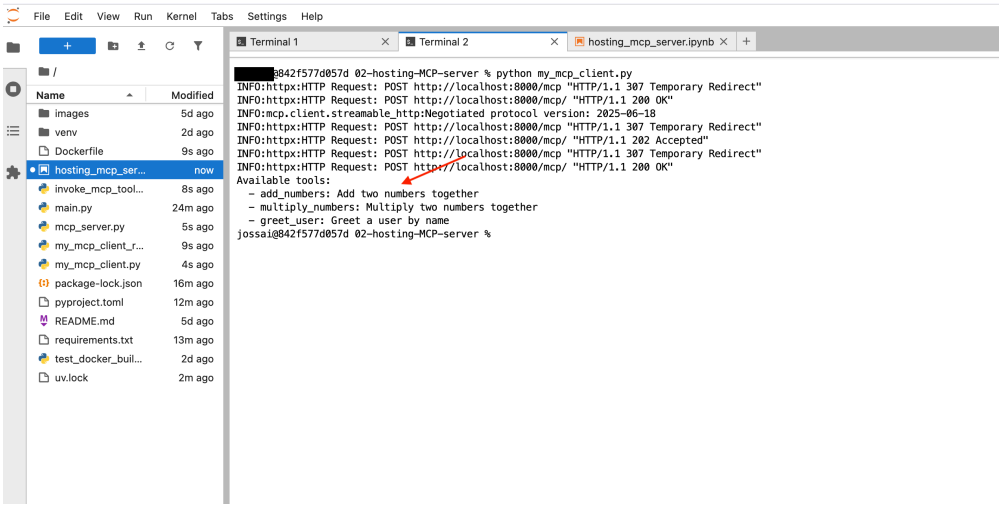
To test your MCP server locally:

1. **Terminal 1**: Start the MCP server

```
python amazon-bedrock-agentcore-workshop/01-AgentCore-runtime/02-hosting-MCP-server/mcp_
```

2. **Terminal 2**: Run the test client

```
python amazon-bedrock-agentcore-workshop/01-AgentCore-runtime/02-hosting-MCP-server/my_
```



You should see your three tools listed in the output.

Setting up Amazon Cognito for Authentication

AgentCore Runtime requires authentication. We'll use Amazon Cognito to provide JWT tokens for accessing our deployed MCP server. We also modify the Python path so that modules in the parent

directory can be imported. It imports utility functions used to set up the Cognito user pool and IAM role

The code below is the helper function to create and configure a Cognito user pool. The result is stored in `cognito_config`, which contains values such as `user_pool_id`, `client_id`, and `discovery_url`.

▼ **Code Snippet**

```
import sys
import os

# Get the current notebook's directory
current_dir = os.path.dirname(os.path.abspath('__file__' if '__file__' in globals() else '.'))

utils_dir = os.path.join(current_dir, '..')
utils_dir = os.path.abspath(utils_dir)

# Add to sys.path
sys.path.insert(0, utils_dir)

from utils import create_agentcore_role, setup_cognito_user_pool
```

```
cognito_config = setup_cognito_user_pool()
```

Create IAM Execution role for the AgentCore Runtime

This code snippet creates an IAM role specifically for the MCP server. This role is required for deploying to AgentCore Runtime and is tied to the tool_name.

▼ **Code Snippet**

```
tool_name = "mcp_server_ac"
agentcore_iam_role = create_agentcore_role(agent_name=tool_name)
```

Configuring AgentCore Runtime Deployment

This sets up the AWS SDK session, determines the current region, and checks for the required files (`mcp_server.py`, `requirements.txt`). If files are missing, it raises an error.

▼ **Code Snippet**

```
from bedrock_agentcore_starter_toolkit import Runtime
from boto3.session import Session
import time
import os

boto_session = Session()
region = boto_session.region_name

required_files = ['mcp_server.py', 'requirements.txt']
for file in required_files:
    if not os.path.exists(file):
        raise FileNotFoundError(f"Required file {file} not found")
```

Next, we initialize the `Runtime` instance and set up authentication configuration using Cognito. This ensures only valid tokens from the user pool can invoke the MCP server.

▼ **Code Snippet**

```
agentcore_runtime = Runtime()

auth_config = {
    "customJWTAuthorizer": {
        "allowedClients": [
            cognito_config['client_id']
        ],
        "discoveryUrl": cognito_config['discovery_url'],
    }
}
```

Here, we configure the MCP runtime using this script as the entry point. It creates an ECR repository automatically and prepares the runtime to deploy the MCP server using the defined IAM role and authentication.

▼ **Code Snippet**

```
response = agentcore_runtime.configure(
    entrypoint="mcp_server.py",
    execution_role=agentcore_iam_role['Role']['Arn'],
    auto_create_ecr=True,
    requirements_file="requirements.txt",
    region=region,
    authorizer_configuration=auth_config,
    protocol="MCP",
    agent_name=tool_name
)
```

Documentation Note
We use the provided starter toolkit to configure the AgentCore Runtime with an entrypoint, execution role, and requirements file. It will also auto-create the Amazon ECR repository during setup. A Dockerfile will be generated from your application code as part of this step.

Launch MCP Server to AgentCore Runtime

This next script initializes a Bedrock AgentCore `Runtime` session using the Runtime class. It sets the AWS region, checks for the required source files, and defines a custom JWT authentication configuration using Amazon Cognito credentials. The `configure()` method sets up the runtime deployment with all necessary parameters—such as IAM role, Python entrypoint, and required libraries—preparing it for launch.

▼ **Code Snippet**

```
from bedrock_agentcore_starter_toolkit import Runtime
from boto3.session import Session
import time
import os

boto_session = Session()
region = boto_session.region_name

required_files = ['mcp_server.py', 'requirements.txt']
for file in required_files:
    if not os.path.exists(file):
        raise FileNotFoundError(f"Required file {file} not found")

agentcore_runtime = Runtime()

auth_config = {
    "customJWTAuthorizer": {
        "allowedClients": [
            cognito_config['client_id']
        ],
```

```
        "discoveryUrl": cognito_config['discovery_url'],
    }
}

response = agentcore_runtime.configure(
    entrypoint="mcp_server.py",
    execution_role=agentcore_iam_role['Role']['Arn'],
    auto_create_ecr=True,
    requirements_file="requirements.txt",
    region=region,
    authorizer_configuration=auth_config,
    protocol="MCP",
    agent_name=tool_name
)
```

Now, we launch the configured runtime, deploying the MCP server to Amazon Bedrock AgentCore Runtime. The result includes identifiers like `agent_arn` and `agent_id`.

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

With the Dockerfile ready, we deploy the MCP server to AgentCore Runtime. This step creates both the Amazon ECR repository and the AgentCore Runtime environment. Lets wait for about 20 seconds for the AgentCore runtime status to be ready before we store the configurations for remote access.

Before we can invoke our deployed MCP server, let's store the Agent ARN and Cognito configuration in AWS Systems Manager Parameter Store and AWS Secrets Manager for easy retrieval:

▼ **Code Snippet**

```
import boto3
import json

ssm_client = boto3.client('ssm', region_name=region)
secrets_client = boto3.client('secretsmanager', region_name=region)

try:
    cognito_credentials_response = secrets_client.create_secret(
        Name='mcp_server/cognito/credentials',
        Description='Cognito credentials for MCP server',
        SecretString=json.dumps(cognito_config)
    )
except secrets_client.exceptions.ResourceExistsException:
    secrets_client.update_secret(
        SecretId='mcp_server/cognito/credentials',
        SecretString=json.dumps(cognito_config)
    )

agent_arn_response = ssm_client.put_parameter(
    Name='/mcp_server/runtime/agent_arn',
    Value=launch_result.agent_arn,
    Type='String',
    Description='Agent ARN for MCP server',
    Overwrite=True
)
```

Creating Remote Testing Client

This snippet will create a test client that will retrieve the necessary credentials from AWS and connect to the deployed server.

▼ **Code Snippet**

```
%%writefile my_mcp_client_remote.py
import asyncio
```

```python
import boto3
import json
import sys
from boto3.session import Session

from mcp import ClientSession
from mcp.client.streamable_http import streamablehttp_client

async def main():
    boto_session = Session()
    region = boto_session.region_name

    print(f"Using AWS region: {region}")

    try:
        ssm_client = boto3.client('ssm', region_name=region)
        agent_arn_response = ssm_client.get_parameter(Name='/mcp_server/runtime/agent_arn')
        agent_arn = agent_arn_response['Parameter']['Value']
        print(f"Retrieved Agent ARN: {agent_arn}")

        secrets_client = boto3.client('secretsmanager', region_name=region)
        response = secrets_client.get_secret_value(SecretId='mcp_server/cognito/credentials')
        secret_value = response['SecretString']
        parsed_secret = json.loads(secret_value)
        bearer_token = parsed_secret['bearer_token']
        print("✓ Retrieved bearer token from Secrets Manager")

    except Exception as e:
        print(f"Error retrieving credentials: {e}")
        sys.exit(1)

    if not agent_arn or not bearer_token:
        print("Error: AGENT_ARN or BEARER_TOKEN not retrieved properly")
        sys.exit(1)

    encoded_arn = agent_arn.replace(':', '%3A').replace('/', '%2F')
    mcp_url = f"https://bedrock-agentcore.{region}.amazonaws.com/runtimes/{encoded_arn}/invocati
    headers = {
        "authorization": f"Bearer {bearer_token}",
        "Content-Type": "application/json"
    }

    print(f"\nConnecting to: {mcp_url}")
    print("Headers configured ✓")

    try:
        async with streamablehttp_client(mcp_url, headers, timeout=120, terminate_on_close=False
            read_stream,
            write_stream,
            _,
        ):
            async with ClientSession(read_stream, write_stream) as session:
                print("\n🔄 Initializing MCP session...")
                await session.initialize()
                print("✓ MCP session initialized")

                print("\n🔄 Listing available tools...")
                tool_result = await session.list_tools()

                print("\n📋 Available MCP Tools:")
                print("=" * 50)
                for tool in tool_result.tools:
                    print(f"🔧 {tool.name}")
                    print(f"   Description: {tool.description}")
                    if hasattr(tool, 'inputSchema') and tool.inputSchema:
                        properties = tool.inputSchema.get('properties', {})
                        if properties:
                            print(f"   Parameters: {list(properties.keys())}")
                    print()

                print(f"✅ Successfully connected to MCP server!")
                print(f"Found {len(tool_result.tools)} tools available.")

```

```
    except Exception as e:
        print(f"❌ Error connecting to MCP server: {e}")
        sys.exit(1)

if __name__ == "__main__":
    asyncio.run(main())
```

Testing

Here are a few ways to test your config:

1. Testing Your Deployed MCP Server

```
print("Testing deployed MCP server...")
print("=" * 50)
!python my_mcp_client_remote.py
```

2. Invoking MCP Tools Remotely

This is an advanced client that goes beyond listing tools. It will also invokes them to showcase the full capabilities of the MCP system.

▼ **Code Snippet**

```
%%writefile invoke_mcp_tools.py
import asyncio
import boto3
import json
import sys
from boto3.session import Session

from mcp import ClientSession
from mcp.client.streamable_http import streamablehttp_client

async def main():
    boto_session = Session()
    region = boto_session.region_name

    print(f"Using AWS region: {region}")

    try:
        ssm_client = boto3.client('ssm', region_name=region)
        agent_arn_response = ssm_client.get_parameter(Name='/mcp_server/runtime/agent_arn')
        agent_arn = agent_arn_response['Parameter']['Value']
        print(f"Retrieved Agent ARN: {agent_arn}")

        secrets_client = boto3.client('secretsmanager', region_name=region)
        response = secrets_client.get_secret_value(SecretId='mcp_server/cognito/credentials')
        secret_value = response['SecretString']
        parsed_secret = json.loads(secret_value)
        bearer_token = parsed_secret['bearer_token']
        print("✓ Retrieved bearer token from Secrets Manager")

    except Exception as e:
        print(f"Error retrieving credentials: {e}")
        sys.exit(1)

    encoded_arn = agent_arn.replace(':', '%3A').replace('/', '%2F')
    mcp_url = f"https://bedrock-agentcore.{region}.amazonaws.com/runtimes/{encoded_arn}/invocat
    headers = {
        "authorization": f"Bearer {bearer_token}",
        "Content-Type": "application/json"
    }

    print(f"\nConnecting to: {mcp_url}")

```

```
    try:
        async with streamablehttp_client(mcp_url, headers, timeout=120, terminate_on_close=Fals
            read_stream,
            write_stream,
            _,
        ):
            async with ClientSession(read_stream, write_stream) as session:
                print("\n🔄 Initializing MCP session...")
                await session.initialize()
                print("✓ MCP session initialized")

                print("\n🔄 Listing available tools...")
                tool_result = await session.list_tools()

                print("\n📋 Available MCP Tools:")
                print("=" * 50)
                for tool in tool_result.tools:
                    print(f"🔧 {tool.name}: {tool.description}")

                print("\n🪄 Testing MCP Tools:")
                print("=" * 50)

                try:
                    print("\n➕ Testing add_numbers(5, 3)...")
                    add_result = await session.call_tool(
                        name="add_numbers",
                        arguments={"a": 5, "b": 3}
                    )
                    print(f"   Result: {add_result.content[0].text}")
                except Exception as e:
                    print(f"   Error: {e}")

                try:
                    print("\n✖️  Testing multiply_numbers(4, 7)...")
                    multiply_result = await session.call_tool(
                        name="multiply_numbers",
                        arguments={"a": 4, "b": 7}
                    )
                    print(f"   Result: {multiply_result.content[0].text}")
                except Exception as e:
                    print(f"   Error: {e}")

                try:
                    print("\n👋 Testing greet_user('Alice')...")
                    greet_result = await session.call_tool(
                        name="greet_user",
                        arguments={"name": "Alice"}
                    )
                    print(f"   Result: {greet_result.content[0].text}")
                except Exception as e:
                    print(f"   Error: {e}")

                print("\n✅ MCP tool testing completed!")

    except Exception as e:
        print(f"❌ Error connecting to MCP server: {e}")
        sys.exit(1)

if __name__ == "__main__":
    asyncio.run(main())
```

Try it out

At an AWS Event

If you are following the workshop via workshop studio, go to JupyterLab in SageMaker Studio. In the JupyterLab UI navigate to 01-AgentCore-runtime/02-hosting-MCP-server.

Self-paced

Here's a notebook that lets you try out the above: hosting_mcp_server.ipynb.