Smithy APIs to MCP tools

Transform Smithy APIs into MCP Tools using AgentCore Gateway

Documentation Note
This document provides a conceptual overview of AgentCore Gateway's Smithy transformation capabilities. For complete implementation details, working code examples, and actual API specifications, please refer to the official tutorial at: AgentCore Gateway Smithy APIs to MCP Tools

Introduction

Amazon Bedrock AgentCore Gateway enables enterprises to seamlessly transform their existing Smithy API specifications into fully-managed MCP (Model Context Protocol) tools without infrastructure management overhead. This capability is particularly powerful for integrating with AWS services like S3, DynamoDB, and other AWS APIs that are defined using Smithy interface definition language.

Smithy is a protocol-agnostic interface definition language and set of tools for generating clients, servers, and documentation for any programming language. It provides a clean, type-safe way to define APIs that can be transformed into multiple target formats, including MCP tools through AgentCore Gateway.

Why is this Important?

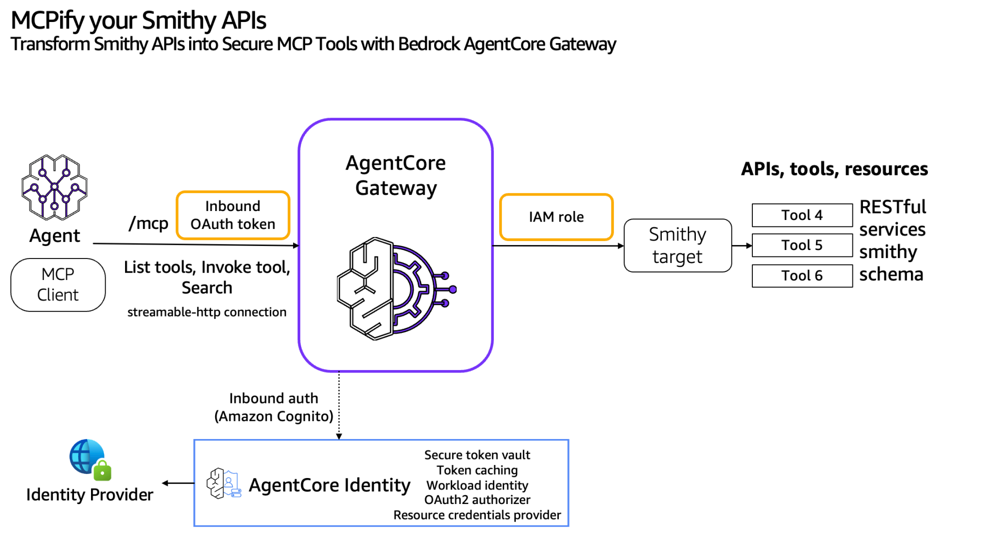
Modern enterprise environments rely heavily on AWS services and APIs defined using Smithy specifications. Traditional approaches require extensive custom development, API wrapper creation, and ongoing maintenance. Smithy-based API integration presents unique challenges: AWS services expose hundreds of operations through Smithy models, each with complex parameter structures and authentication requirements.

In this lab, you will:

- Transform existing Smithy API specifications into MCP tools using AgentCore Gateway
- Integrate AWS services like Amazon S3 through Smithy models
- Configure IAM-based authentication for secure AWS resource access
- Build operational AI agents that can interact with AWS infrastructure
- Demonstrate real-world use cases with S3 bucket management

By the end of this lab, you should have hands-on experience transforming Smithy APIs into MCP tools and understand how to build operational AI agents that can securely manage AWS resources while maintaining security and type safety.

High-Level Architecture



This architecture demonstrates enterprise-scale AWS service integration through Smithy API transformation. The **Strands Agent** provides framework-agnostic functionality using Amazon Bedrock models for natural language processing and operational reasoning. The **AgentCore Gateway** serves as the transformation engine, converting Smithy specifications into MCP-compatible tools with built-in security and authentication. **Smithy API Specifications** represent the source definitions for AWS services, stored in S3 and processed by the Gateway. The **AWS IAM Integration** provides secure, role-based access to AWS services without exposing credentials to agents.

Prerequisites & Setup

Environment Requirements

Before starting this lab, ensure you have:

- **Python 3.10+** installed on your system
- **Jupyter Notebook** environment for interactive development
- **AWS credentials** configured with appropriate permissions
- **Amazon Bedrock model access** (Anthropic Claude or Amazon Nova)
- **S3 bucket** for storing Smithy specifications
- **Amazon Cognito** for inbound authentication
- **UV package manager** for Python dependency management

AWS Permissions & Configuration Setup

Required AWS Permissions

Ensure your AWS credentials include the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:*",
                "s3:*",
                "dynamodb:*",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PassRole",
                "agentcore:*",
                "cognito-idp:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Smithy Specification Setup

The lab uses Smithy specifications for AWS services, specifically Amazon S3:

S3 Smithy Model - Complete S3 API operations defined in Smithy format **IAM Integration** - Role-based authentication for secure AWS service access

Example Smithy specification structure:

```
{
  "smithy": "2.0",
  "shapes": {
    "com.amazon.s3#ListBuckets": {
      "type": "operation",
      "output": {
        "target": "com.amazon.s3#ListBucketsOutput"
      }
    },
    "com.amazon.s3#ListBucketsOutput": {
      "type": "structure",
      "members": {
        "Buckets": {
          "target": "com.amazon.s3#Buckets"
        }
      }
    }
  }
}
```

Environment Configuration

```
# AWS Configuration
export AWS_DEFAULT_REGION=us-west-2
export AWS_ACCESS_KEY_ID=your_access_key
export AWS_SECRET_ACCESS_KEY=your_secret_key

# S3 Configuration for Smithy specs
export SMITHY_SPEC_BUCKET=your-smithy-specs-bucket
export SMITHY_SPEC_KEY=s3-apis.json
```

Implementation

Core Smithy Transformation Concepts

AgentCore Gateway's Smithy transformation functionality provides automated conversion of Smithy API specifications into MCP-compatible tools. The transformation preserves the type safety and validation rules defined in the Smithy model while exposing operations through the standardized MCP protocol.

Key transformation features include:

- **Operation Mapping**: Each Smithy operation becomes an individual MCP tool
- **Type Preservation**: Smithy shape definitions are converted to MCP parameter schemas
- **Authentication Integration**: IAM roles provide secure access to AWS services
- **Validation**: Smithy constraints are enforced during tool execution

Complete implementation examples and transformation details are available in the Gateway Smithy tutorial.

Gateway Creation with Cognito Authentication

```python
import boto3
from bedrock_agentcore_client import BedrockAgentCoreGatewayClient

# Initialize Gateway client
gateway_client = BedrockAgentCoreGatewayClient()

# Configure Cognito authentication
auth_config = {
    "customJWTAuthorizer": {
        "allowedClients": [client_id],
        "discoveryUrl": cognito_discovery_url
    }
}

# Create Gateway for Smithy transformation
gateway_response = gateway_client.create_gateway(
    name='SmithyToMCPGateway',
    roleArn=gateway_iam_role_arn,
    protocolType='MCP',
    authorizerType='CUSTOM_JWT',
    authorizerConfiguration=auth_config,
    description='Gateway for transforming Smithy APIs to MCP tools'
)

gateway_id = gateway_response['gatewayId']
gateway_url = gateway_response['gatewayUrl']
```

Smithy Target Configuration

```python
# Configure S3-stored Smithy specification
smithy_target_config = {
    "mcp": {
        "smithyModel": {
            "s3": {
                "uri": "s3://your-bucket/s3-apis.json"
            }
        }
    }
}

# Configure IAM authentication for AWS services
credential_config = {
    "credentialProviderType": "GATEWAY_IAM_ROLE"
}

# Create Smithy target
smithy_target = gateway_client.create_gateway_target(
    gatewayIdentifier=gateway_id,
    name="S3SmithyTarget",
    description="Amazon S3 operations via Smithy transformation",
    targetConfiguration=smithy_target_config,
    credentialProviderConfigurations=[credential_config]
)
```

Target Configuration Explained:

- `smithyModel`: References the Smithy specification file containing API definitions
- `s3.uri`: S3 location of the Smithy JSON specification
- `GATEWAY_IAM_ROLE`: Uses Gateway's IAM role for AWS service authentication
- **Automatic Transformation**: All operations in the Smithy model become MCP tools

Different Approaches/Options

Framework Integration Patterns

Strands Agents | LangGraph | CrewAI

```python
from strands import Agent, tool
from strands.tools.mcp.mcp_client import MCPClient

@tool
def aws_infrastructure_manager(task: str):
    """Manage AWS infrastructure using Smithy-transformed tools"""
    available_tools = mcp_client.list_tools_sync()
    relevant_tools = [t for t in available_tools if 's3' in t.name.lower()]
    return execute_aws_operation(relevant_tools, task)

agent = Agent(model=model, tools=[aws_infrastructure_manager])
```

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 `02-AgentCore-gateway/02-transform-apis-into-mcp-tools/02-transform-smithyapis-into-mcp-tools/01-s3-smithy-into-mcp-iam.ipynb`

Self-paced

For the complete working implementation and examples: Smithy APIs to MCP Tools Tutorial.

The tutorial contains:

- `README.md` - Detailed implementation guide
- `01-s3-smithy-into-mcp-iam.ipynb` - Interactive notebook with working examples
- `smithy-specs/` - Example Smithy specifications for AWS services
- Complete code samples and API transformation patterns

Congratulations!

capabilities and demonstrated how to build operational AI agents that can interact directly with AWS infrastructure. Your implementation showcases the power of transforming existing Smithy specifications into MCP-compatible tools, enabling agents to perform real AWS operations while maintaining security and type safety!

Previous Next