Gateway Search Tools

Gateway Semantic Search

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

Introduction

Amazon Bedrock AgentCore Gateway addresses one of the most challenging problems in enterprise AI agent development: efficiently managing and selecting from hundreds or thousands of available tools. Traditional approaches to tool selection require agents to process extensive tool metadata in their prompts, leading to increased costs, higher latency, and poor tool selection accuracy.

AgentCore Gateway's semantic search capability revolutionizes this approach by enabling agents to intelligently discover and select only the most relevant tools for a given context or question. This functionality dramatically improves agent performance while reducing operational costs and response times.

Why is this Important?

Modern enterprise environments often expose hundreds of API endpoints and functions through MCP servers. Enterprise SaaS integration scenarios involve services like Salesforce, Zendesk, Slack, and JIRA, each exposing 50-200+ API endpoints. Traditional tool management approaches create significant challenges: tool sets frequently exceed LLM context limits, agents struggle to choose optimal tools from extensive lists, and processing hundreds of tool descriptions significantly delays response times while causing cost escalation problems.

AgentCore Gateway Search Solutions

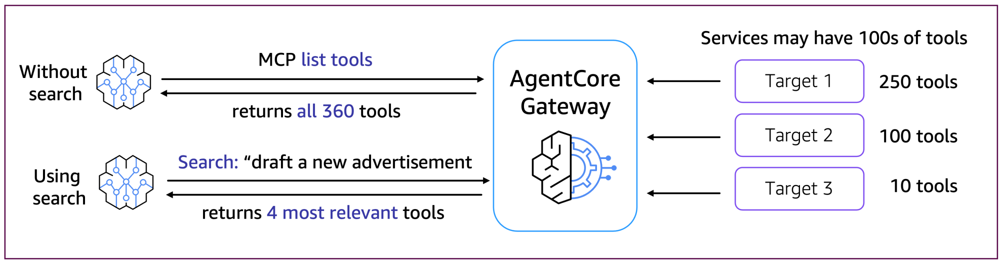
- **Intelligent Tool Discovery**: Semantic search finds relevant tools based on natural language queries
- **Context-Aware Selection**: Tools are filtered based on current conversation context
- **Performance Optimization**: Up to 3x latency improvement by focusing on relevant tools
- **Cost Reduction**: Dramatically reduced token usage through targeted tool selection
- **Scalable Architecture**: Supports enterprise-scale tool inventories ::

In this lab, you will:

- Configure AgentCore Gateway with multiple Lambda-backed targets containing numerous tools
- Implement semantic search to dynamically discover relevant tools from tool sets
- Compare performance metrics between traditional and search-enabled approaches
- Build intelligent agents that efficiently navigate complex tool ecosystems
- Optimize agent behavior using search-driven tool selection
- Measure latency improvements in real-world scenarios

By the end of this lab, you should have hands-on experience with Gateway's search capabilities and understand how to scale agent tooling to enterprise requirements. The implementation demonstrates how semantic search transforms agent performance in enterprise environments with hundreds of available tools.

High-Level Architecture



This architecture showcases enterprise-scale tool management through intelligent search capabilities. The **Strands Agent** provides framework-agnostic functionality using Amazon Bedrock models for natural language processing and reasoning. The **AgentCore Gateway** serves as the central hub for tool management, featuring built-in semantic search that transforms how agents interact with tool inventories. **Lambda Targets** represent multiple AWS Lambda functions across different tool categories, simulating real-world enterprise environments with diverse functionality. The **Semantic Search Engine** provides intelligent tool discovery capabilities that eliminate the need to process extensive tool lists.

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)
- **AWS Lambda permissions** for function creation and execution
- **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:*",
                "lambda:*",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PassRole",
                "agentcore:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Lambda Function Setup

The lab uses multiple Lambda functions to simulate a diverse tool ecosystem:

Calculator Service - Mathematical operations and computations **Restaurant Service** - Food service APIs for reservations and menu queries

Each Lambda function will be automatically deployed with the following structure:

```python
# Example Lambda function structure
import json

def lambda_handler(event, context):
    """
    Generic Lambda handler for tool operations
    """
    operation = event.get('operation')
    parameters = event.get('parameters', {})

    # Tool-specific logic here
    result = process_operation(operation, parameters)

    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

Environment Configuration

```bash
# 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

# Performance Testing
export PERFORMANCE_METRICS_ENABLED=true
export SEARCH_COMPARISON_MODE=true
```

Implementation

Core Search Concepts

AgentCore Gateway's search functionality provides tool discovery through semantic search capabilities. Semantic search uses natural language understanding to find tools based on meaning rather than exact keyword matches. The search allows agents to find relevant tools from tool sets using natural language queries, rather than processing all available tools.

Key search parameters include:

- **Query**: Natural language description of desired functionality. You can set maximum number of tools to return in the query.
- **Tool Selection**: Results are ranked by relevance to help agents choose the most appropriate tools. The most relevant tools are first on the list. The initial implementation of search gives back up to 10 results.

Complete implementation examples and performance benchmarks are available in the Gateway Search tutorial.

Gateway Creation with Multiple Targets

```python
import boto3
from bedrock_agentcore_client import BedrockAgentCoreGatewayClient

# Initialize Gateway client
```

```python
gateway_client = BedrockAgentCoreGatewayClient()

# Create Gateway for tool search demonstration
gateway_response = gateway_client.create_gateway(
    gatewayName="enterprise-tools-gateway",
    description="Gateway demonstrating semantic search across tool sets"
)

gateway_arn = gateway_response['gatewayArn']
gateway_url = gateway_response['mcpUrl']
```

Lambda Target Configuration

```python
# Configure Calculator Lambda target
calculator_target = gateway_client.create_target(
    gatewayArn=gateway_arn,
    targetName="calculator-service",
    targetConfiguration={
        "lambdaTargetConfiguration": {
            "functionArn": calculator_lambda_arn,
            "invocationType": "RequestResponse"
        }
    }
)

# Configure Restaurant Lambda target
restaurant_target = gateway_client.create_target(
    gatewayArn=gateway_arn,
    targetName="restaurant-service",
    targetConfiguration={
        "lambdaTargetConfiguration": {
            "functionArn": restaurant_lambda_arn,
            "invocationType": "RequestResponse"
        }
    }
)
```

Target Configuration Explained:

- `lambdaTargetConfiguration`: Specifies Lambda function as tool provider
- `functionArn`: AWS Lambda function ARN containing tool implementations
- `invocationType`: Request/response pattern for synchronous tool execution
- **Multiple Targets**: Each target can contain dozens of individual tools

Search-Enabled Agent Implementation

```python
from strands import Agent, tool
from strands.models import BedrockModel
import requests
import time

# Configure Bedrock model
model = BedrockModel(model_id="us.anthropic.claude-sonnet-4-20250514-v1:0")

class GatewaySearchClient:
    def __init__(self, gateway_url: str):
        self.gateway_url = gateway_url
        self.headers = {"Content-Type": "application/json"}

    def search_tools(self, query: str, limit: int = 5):
        """
        Search for relevant tools using Gateway semantic search
        """
        # Implementation details depend on actual Gateway API
        pass

    def list_all_tools(self):
        """
        Get complete list of available tools (for comparison)
        """
        # Implementation details depend on actual Gateway API
        pass

    def call_tool(self, tool_name: str, arguments: dict):
        """
        Execute a specific tool through the Gateway
        """
        # Implementation details depend on actual Gateway API
        pass

# Initialize Gateway client
gateway_client = GatewaySearchClient(gateway_url)
```

Search Implementation Components:

- **Search Method**: Gateway provides semantic tool discovery capabilities
- **Query Parameter**: Natural language description of desired functionality
- **Limit Parameter**: Maximum number of tools to return from search
- **Tool Ranking**: Results ordered by relevance
- **Performance Tracking**: Measurement of search and execution times

Different Approaches/Options

Search Examples

Basic Tool Search

```python
# Search for relevant tools using natural language
search_results = gateway_client.search_tools(
    query="calculate mathematical expressions",
    limit=5
)
```

Search with Different Queries

```python
# Search with different query formulations
search_results = gateway_client.search_tools(
    query="business operations",
    limit=10
)
```

Framework Integration Patterns

Strands Agents | LangGraph | CrewAI

```python
from strands import Agent, tool

@tool
def intelligent_tool_selection(task: str):
    relevant_tools = gateway_client.search_tools(task)
    return execute_best_tool(relevant_tools, task)

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

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/03-search-tools/01-gateway-search.ipynb`

Self-paced

For the complete working implementation and examples: Gateway Search Tools Tutorial.

The tutorial contains:

- `README.md` - Detailed implementation guide
- `01-gateway-search.ipynb` - Interactive notebook with working examples
- Complete code samples and API usage patterns

Congratulations!

Outstanding work! You've successfully implemented AgentCore Gateway's semantic search capabilities and demonstrated how to efficiently manage tool catalogs in enterprise AI applications. Your implementation showcases the dramatic performance improvements possible when agents can intelligently discover and select relevant tools rather than processing exhaustive tool inventories, creating a foundation for production-ready agents that can efficiently navigate complex enterprise tool ecosystems!

Previous Next