Kasadara

Integration of Google SERP API with Amazon Bedrock Agent via AWS Lambda

Case Study - Google SERP API using AWS

CHINNASAMY

Kasadara

Table of Content

Kasadara

1. Introduction

1.1 Purpose of the Integration

The goal of this project is to integrate the Google SERP API with Amazon Bedrock Agents using an AWS Lambda function. This integration enables Bedrock to dynamically fetch real-time search engine results in response to user prompts.

1.2 Key Components Overview

- **Amazon Bedrock Agent**: A managed service for building and deploying generative AI agents.
- **AWS Lambda**: Serverless compute used to call external APIs and process logic.
- **Google SERP API (SerpAPI)**: A third-party service providing real-time Google Search results through a RESTful API.

1.3 Expected Outcomes

- Fully automated interaction between Amazon Bedrock and Google Search.
- Custom Lambda function deployed and triggered by Bedrock.
- Real-time, structured search results returned to end-users.

2. Prerequisites

2.1 Google SERP API Account Setup

- Navigate to your dashboard and copy your unique API key

2.2 API Key Management

- Use AWS Secrets Manager to securely store the SerpAPI key.
- Grant Lambda function permission to access the secret.

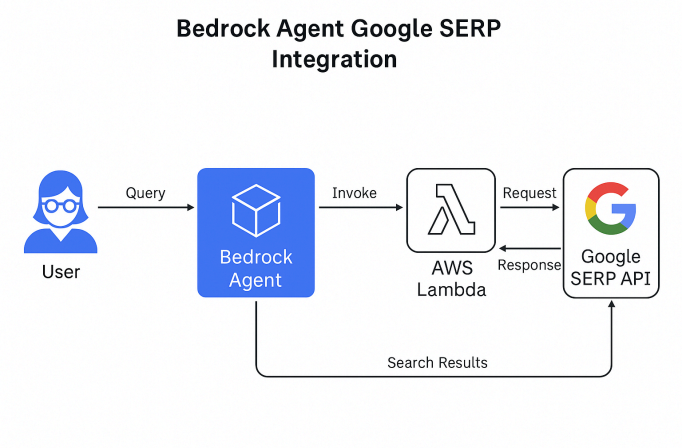
2.3 IAM Role and Permissions Required

- Create a role for the Lambda function with the following permissions:
- AWSLambdaBasicExecutionRole
- secretsmanager:GetSecretValue

- bedrock:InvokeAgent

3. Solution Architecture

3.1 Architectural Diagram



3.2 Component Breakdown

- Bedrock Agent triggers Lambda Function via Action Group.
- Lambda queries Google SERP API and returns a structured response

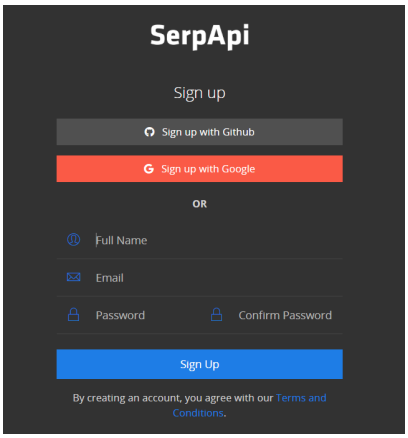
3.3 Request Flow Overview

1. User asks a search query to Bedrock.
2. Bedrock invokes Lambda with the query.
3. Lambda contacts SerpAPI and fetches the results.
4. Results returned to Bedrock and displayed to the user.

4. Google SERP API Setup

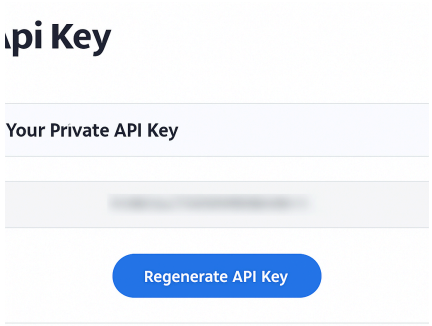
4.1 Create an Account at SerpAPI

- Sign up and log in to SerpAPI.



4.2 Generate and Secure the API Key

- Copy the key from the dashboard. Save the key in AWS Secrets Manager.



4.3 Understand Query Parameters

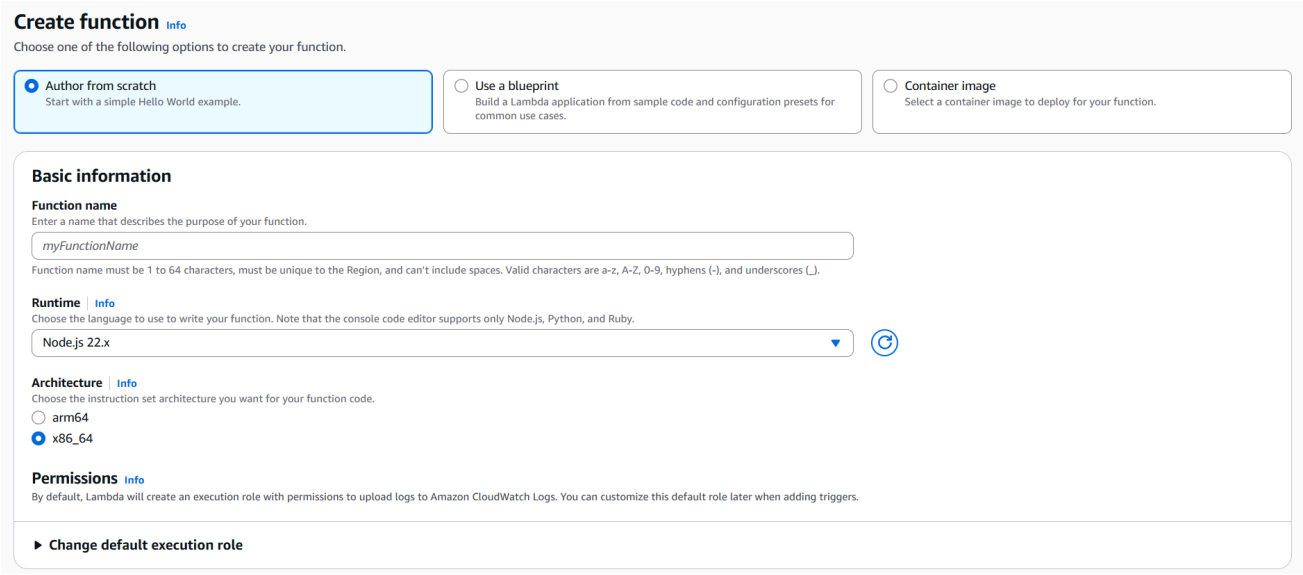
- Use parameters like q , engine=google , and num=5 .

Example:

https://serpapi.com/search?q=AWS+Lambda&engine=google&api_key=YOUR_API_KEY

5. Lambda Function Development

5.1 New Lambda Function



5.2 Configure Execution Role

- Choose existing role with necessary permissions

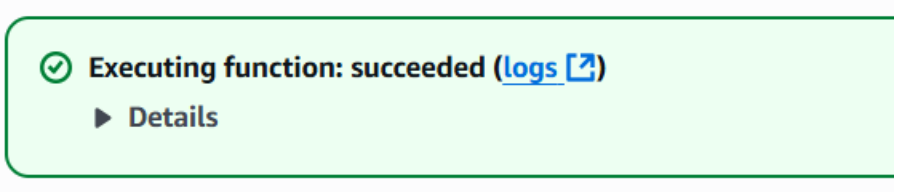
5.3 Write and Deploy the Lambda Code

```
secret_name = "serpapi_key"
region_name = "us-east-1"

# Fetch secret
session = boto3.session.Session()
client = session.client(service_name='secretsmanager', region_name=region_name)
secret = json.loads(client.get_secret_value(SecretId=secret_name)['SecretString'])
api_key = secret['key']

query = event['query']
url = f"https://serpapi.com/search?q={query}&engine=google&api_key={api_key}"
response = requests.get(url)

return {
```

```
'statusCode': 200,
'body': response.json()
}
```

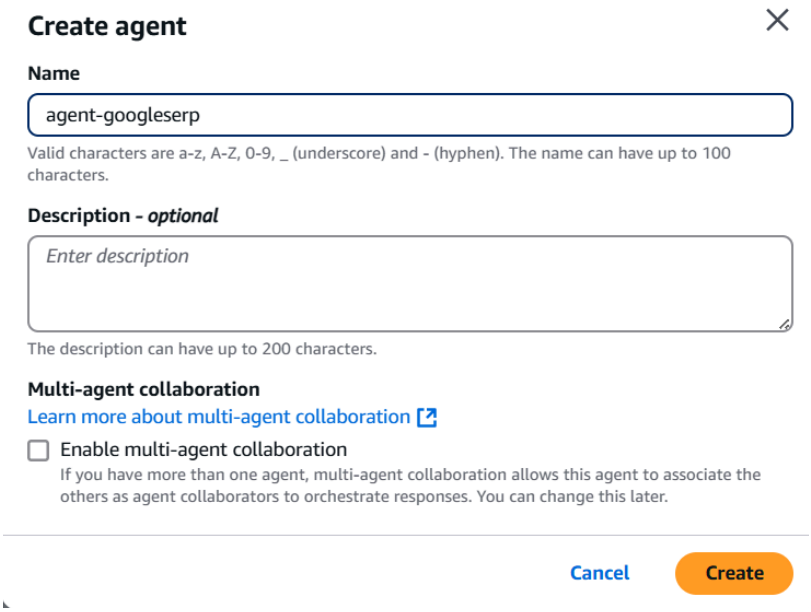
5.4 Test Lambda Function Independently

- Create test event with payload: { "query": "Latest AI news" }
- Check the returned results



6. Amazon Bedrock Agent Configuration

6.1 Bedrock Agent

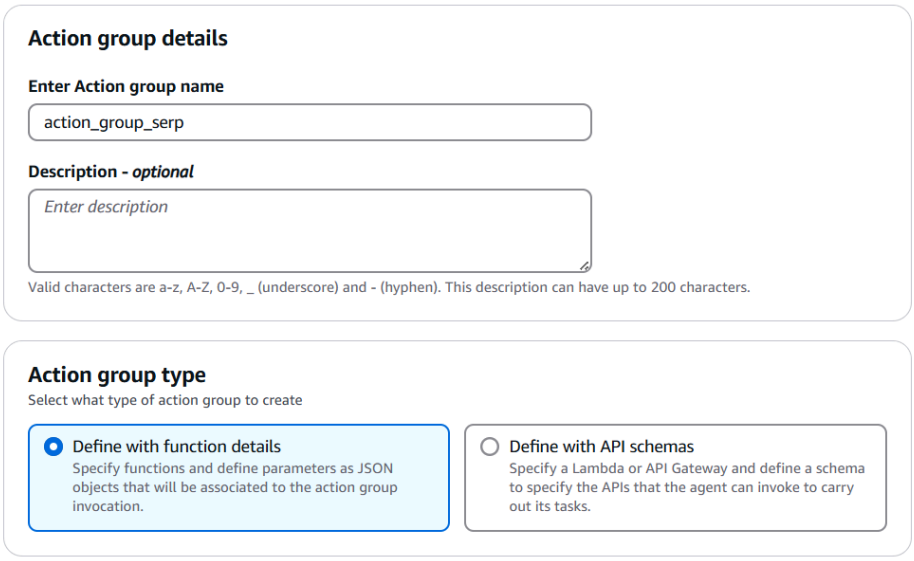


6.2 Configure Bedrock Agent Knowledge Base (optional)

- Optionally attach a knowledge base if combining web search with enterprise knowledge.

6.3 Enable Lambda Function as a Bedrock Action

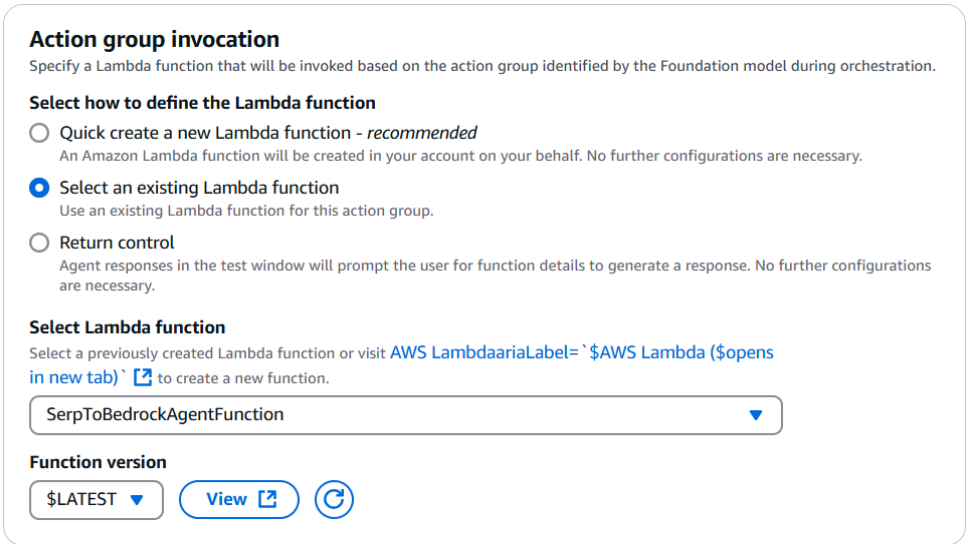
- Add an Action Group.
- Define Action Schema with the input/output parameters.
- Choose Lambda ARN as the action target



7. Lambda and Bedrock Agent Integration

7.1 Link Lambda as an Action Group in Bedrock

- Navigate to Bedrock Agent > Add Action Group

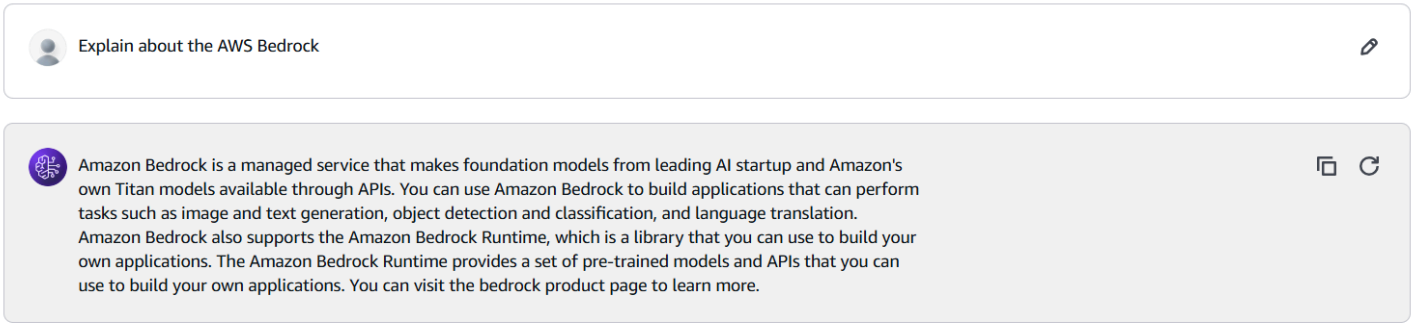


7.2 Define the Action Schema

```
{
  "actionGroup": "search",
  "actionName": "searchGoogle",
  "parameters": [
        { "name": "query", "type": "string", "required": true }
  ],
  "output": [
        { "name": "results", "type": "list" }
  ]
 }
```

7.3 Testing via Agent Playground

- Ask a question: "Search latest developments in generative AI"
- Validate real-time results.



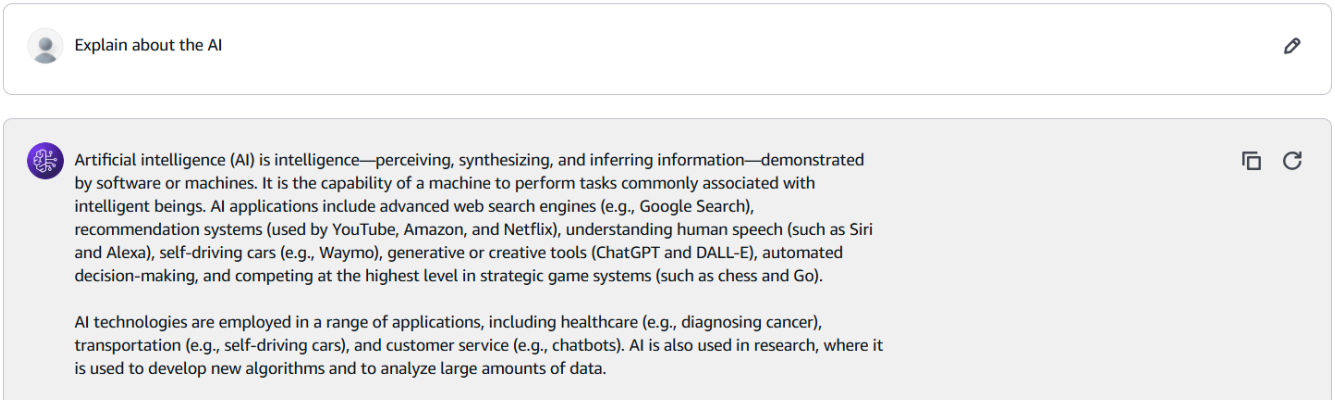
8. Testing the Integration

8.1 Test Cases and Scenarios

- Valid queries
- Invalid queries
- API rate limits

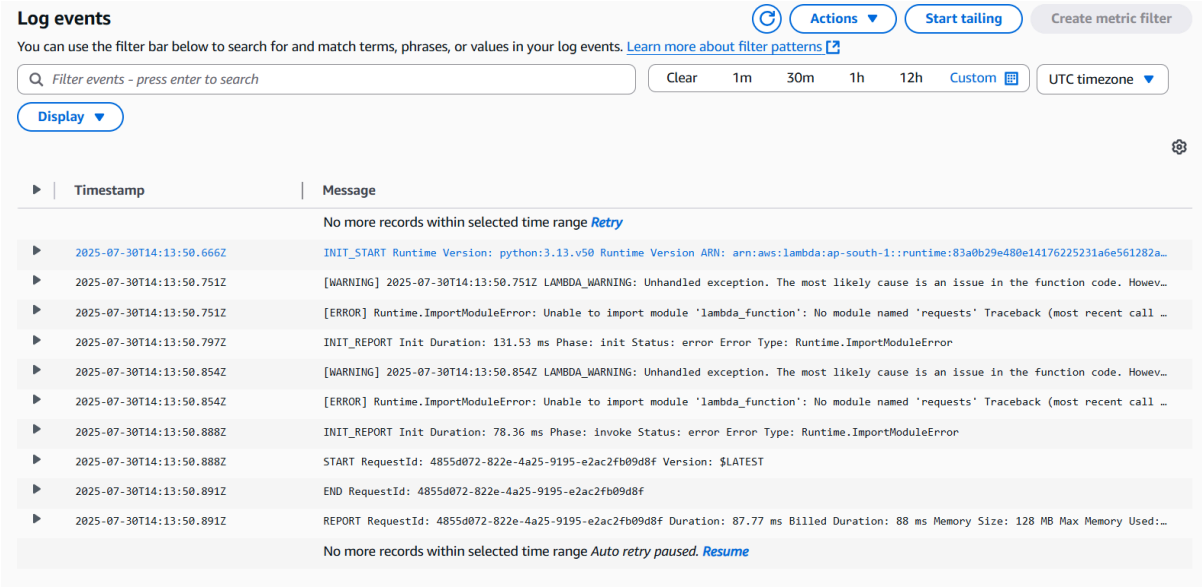
8.2 Validating the Response from SERP API

- Ensure *organic_results* or relevant keys are present



8.3 Reviewing Execution Logs

- CloudWatch logs to review errors or latency



9. Conclusion

- **Scalable Orchestration**: Bedrock Agent orchestrates intelligent search using structured prompts.
- **Lambda as Executor**: AWS Lambda handles runtime logic to invoke the Google SERP API.
- **Real-Time Retrieval**: API responses are processed and returned dynamically to users.
- **Enhanced Contextual Responses**: Bedrock generates relevant answers using retrieved data.
- **Traceable Workflow**: Full orchestration trace shows steps like query processing, summarization, and response generation.
- **Extensible Pattern**: Supports building advanced AI-powered apps requiring dynamic web search.
- **Operational Control**: Maintains visibility and control using native AWS services.

12. Appendices

A. Sample Lambda Code

- (Included in Section 6.3)

B. API Reference Links

- SerpAPI Docs
- Amazon Bedrock Docs

C. IAM Policy JSON Template

```
{
  "Version": "2012-10-17",
  "Statement": [
        {
        "Effect": "Allow",
        "Action": [
        "secretsmanager:GetSecretValue",
        "logs:*",
        "bedrock:*"
        ],
        "Resource": "*"
        }

    ]

}
```