

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

Case Study - Google SERP API using AWS



CHINNASAMY

Table of Content

- 1.Introduction..... 5**
 - 1.1 Purpose of the Integration..... 5
 - 1.2 Key Components Overview..... 5
 - 1.3 Expected Outcomes..... 5
- 2.Prerequisites.....5**
 - 2.1 AWS Account Setup..... 5
 - 2.2 API Key Management.....6
 - 2.3 IAM Role and Permissions Required..... 6
- 3.Solution Architecture..... 6**
 - 3.1 Architectural Diagram..... 6
 - 3.2 Component Breakdown..... 7
 - 3.3 Request Flow Overview..... 7
- 4.Amazon Bedrock Agent Configuration.....7**
 - 4.1 Create a Bedrock Agent..... 7
 - 4.2 Configure Bedrock Agent Knowledge Base (if applicable)..... 7
 - 4.3 Enable Lambda Function as a Bedrock Action..... 8
- 5.Google SERP API Setup..... 8**
 - 5.1 Create an Account at SerpAPI.....8
 - 5.2 Generate and Secure the API Key..... 8
 - 5.3 Understand Query Parameters..... 9
 - 5.4 Test Lambda Function Independently..... 9
- 6.Lambda Function Development.....10**
 - 6.1 Create a New Lambda Function..... 10
 - 6.2 Configure Execution Role..... 10
 - 6.3 Write and Deploy the Lambda Code..... 10
- 7.Lambda and Bedrock Agent Integration..... 11**
 - 7.1 Link Lambda as an Action Group in Bedrock..... 11
 - 7.2 Define the Action Schema.....12
 - 7.3 Testing via Agent Playground..... 12
- 8.Testing the Integration.....12**
 - 8.1 Test Cases and Scenarios..... 12
 - 8.2 Validating the Response from SERP API..... 13
 - 8.3 Reviewing Execution Logs.....13
- 9.Conclusion..... 14**

Appendices..... 14

- A. Sample Lambda Code
- B. API Reference Links
- C. IAM Policy JSON Template

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.

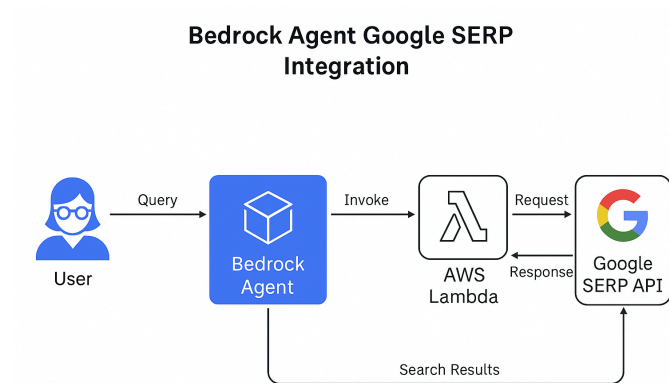
2.3 IAM Role and Permissions Required

- 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

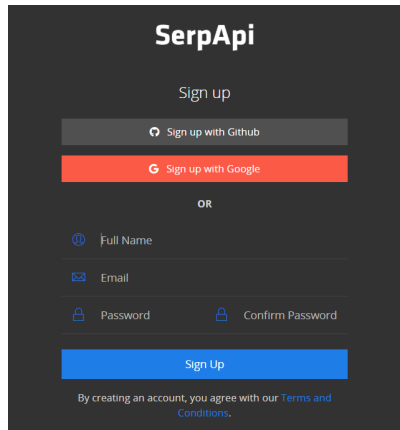
3.3 Request Flow Overview

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

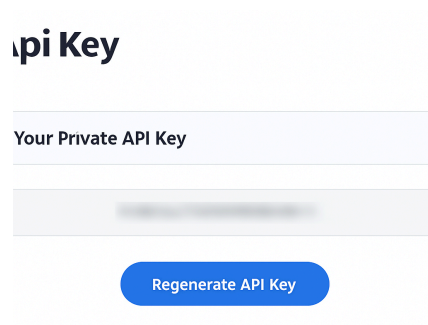
- Sign up and log in to SerpAPI.



The image shows the SerpAPI sign-up interface. It features a dark background with the 'SerpApi' logo at the top. Below the logo is a 'Sign up' heading. There are two buttons for social sign-up: 'Sign up with Github' and 'Sign up with Google'. Below these is an 'OR' separator. Then, there are input fields for 'Full Name', 'Email', 'Password', and 'Confirm Password'. A blue 'Sign Up' button is at the bottom. At the very bottom, there is a small text line: 'By creating an account, you agree with our Terms and Conditions.'

4.2 Generate and Secure the API Key

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



The image shows the SerpAPI 'API Key' dashboard. It has a light gray background. At the top, it says 'API Key'. Below that is a section titled 'Your Private API Key'. Under this title, there is a blurred area representing the API key. At the bottom of this section is a blue button labeled 'Regenerate API Key'.

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

Create function [Info](#)
Choose one of the following options to create your function.

☒ **Author from scratch**
Start with a simple Hello World example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**
Select a container image to deploy for your function.

Basic information
Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 22.x 

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

☐ arm64
☒ x86_64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.
[► Change default execution role](#)

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

✔ Executing function: succeeded ([logs](#))
▶ Details

6. Amazon Bedrock Agent Configuration

6.1 Bedrock Agent

Create agent

Name

agent-google serp

Valid characters are a-z, A-Z, 0-9, _ (underscore) and - (hyphen). The name can have up to 100 characters.

Description - optional

Enter description

The description can have up to 200 characters.

Multi-agent collaboration

[Learn more about multi-agent collaboration](#)

☐ Enable multi-agent collaboration

If you have more than one agent, multi-agent collaboration allows this agent to associate the others as agent collaborators to orchestrate responses. You can change this later.

Cancel

Create

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

Action group details
Enter Action group name

Description - optional

Valid characters are a-z, A-Z, 0-9, _ (underscore) and - (hyphen). This description can have up to 200 characters.

Action group type
Select what type of action group to create

☒ **Define with function details**
Specify functions and define parameters as JSON objects that will be associated to the action group invocation.

☐ **Define with API schemas**
Specify a Lambda or API Gateway and define a schema to specify the APIs that the agent can invoke to carry out its tasks.

7. Lambda and Bedrock Agent Integration

7.1 Link Lambda as an Action Group in Bedrock

- Navigate to Bedrock Agent > Add Action Group

Action group invocation

Specify a Lambda function that will be invoked based on the action group identified by the Foundation model during orchestration.

Select how to define the Lambda function

- ☐ Quick create a new Lambda function - *recommended*
An Amazon Lambda function will be created in your account on your behalf. No further configurations are necessary.
- ☒ Select an existing Lambda function
Use an existing Lambda function for this action group.
- ☐ Return control
Agent responses in the test window will prompt the user for function details to generate a response. No further configurations are necessary.

Select Lambda function

Select a previously created Lambda function or visit [AWS Lambda](#) to create a new function.

SerpToBedrockAgentFunction

Function version

\$LATEST

[View](#)



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.



Explain about the AWS Bedrock



Amazon Bedrock is a managed service that makes foundation models from leading AI startup and Amazon's own Titan models available through APIs. You can use Amazon Bedrock to build applications that can perform tasks such as image and text generation, object detection and classification, and language translation. Amazon Bedrock also supports the Amazon Bedrock Runtime, which is a library that you can use to build your own applications. The Amazon Bedrock Runtime provides a set of pre-trained models and APIs that you can use to build your own applications. You can visit the bedrock product page to learn more.



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



Explain about the AI



Artificial intelligence (AI) is intelligence—perceiving, synthesizing, and inferring information—demonstrated by software or machines. It is the capability of a machine to perform tasks commonly associated with intelligent beings. AI applications include advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and DALL-E), automated decision-making, and competing at the highest level in strategic game systems (such as chess and Go).

AI technologies are employed in a range of applications, including healthcare (e.g., diagnosing cancer), transportation (e.g., self-driving cars), and customer service (e.g., chatbots). AI is also used in research, where it is used to develop new algorithms and to analyze large amounts of data.



8.3 Reviewing Execution Logs

- CloudWatch logs to review errors or latency

Log events [Actions](#) [Start tailing](#) [Create metric filter](#)

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter events - press enter to search Clear 1m 30m 1h 12h Custom UTC timezone

[Display](#)

Timestamp	Message
No more records within selected time range Retry	
2025-07-30T14:13:50.666Z	INIT_START Runtime Version: python:3.13.v50 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:83a0b29e480e14176225231a6e561282a...
2025-07-30T14:13:50.751Z	[WARNING] 2025-07-30T14:13:50.751Z LAMBDA_WARNING: Unhandled exception. The most likely cause is an issue in the function code. Howev...
2025-07-30T14:13:50.751Z	[ERROR] Runtime.ImportModuleError: Unable to import module 'lambda_function': No module named 'requests' Traceback (most recent call ...
2025-07-30T14:13:50.797Z	INIT_REPORT Init Duration: 131.53 ms Phase: init Status: error Error Type: Runtime.ImportModuleError
2025-07-30T14:13:50.854Z	[WARNING] 2025-07-30T14:13:50.854Z LAMBDA_WARNING: Unhandled exception. The most likely cause is an issue in the function code. Howev...
2025-07-30T14:13:50.854Z	[ERROR] Runtime.ImportModuleError: Unable to import module 'lambda_function': No module named 'requests' Traceback (most recent call ...
2025-07-30T14:13:50.888Z	INIT_REPORT Init Duration: 78.36 ms Phase: invoke Status: error Error Type: Runtime.ImportModuleError
2025-07-30T14:13:50.888Z	START RequestId: 4855d072-822e-4a25-9195-e2ac2fb09d8f Version: \$LATEST
2025-07-30T14:13:50.891Z	END RequestId: 4855d072-822e-4a25-9195-e2ac2fb09d8f
2025-07-30T14:13:50.891Z	REPORT RequestId: 4855d072-822e-4a25-9195-e2ac2fb09d8f Duration: 87.77 ms Billed Duration: 88 ms Memory Size: 128 MB Max Memory Used:...
No more records within selected time range Auto retry paused. Resume	

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": "*"
    }
  ]
}
```