Software Requirements Specification (SRS) Document

Routing lambda function calls through EC2 to avoid cold start latency.

Team 48

Maneesh Manoj

Kavish Kapoor

Aravind Jagannathan Tearle

Nikhil Sriram

Umang Patel

Brief Problem Statement

A “cold start” in AWS Lambda is the initial delay that occurs when a function is invoked for the first time or after being idle. This results in a high initial response time for the first request to a "cold" lambda, while the response time for subsequent requests when the lambda is "warm" are lower. The goal is to keep a tiny EC2 instance always on to redirect requests to warm lambdas when it receives requests, and if none are available, warm a lambda while concurrently serving the requests.

An analysis of the average and max response times between only cold starts and this method are required. This analysis should be achieved through use of a rudimentary chatbot that will service requests by calling the lambda functions.

System Requirements

- AWS Lambda
- AWS Cloudwatch
- APIGateway
- AWS EC2
- DynamoDB
- React
- NGINX
- AWS Cloudwatch Metrics

User profile

Users interacting with a chatbot that returns a random response.

Feature requirements

No.	Use Case Name	Description	Release
1.	Chatbot	Chatbot returns a random response based on any user input	R2
2.	Lambda Functions	Call respective lambda functions to store the user input, handle the user request and send a response	R2
3.	Data Visualisation	Data Visualisation is achieved through the cloudwatch metrics dashboard available on AWS	R2
4.	EC2 Instance to host chatbot and route lambda calls	A tiny EC2 instance is kept running in order to route lambda function calls from the chatbot to warm environments	R2

Use Case Description

Use Case Number:	UC-01
Use Case Name:	Chatbot
Overview	Chatbot returns a random response based on any user input, by calling the corresponding lambda functions through the created API's
Actors:	AWS Lambda functions, Chatbot, AWS Cloudwatch, DynamoDB
Pre-Condition:	Chatbot is running
Flow:	1. User enters input into chatbot 2. Chatbot calls corresponding lambda functions 3. Chatbot receives reply 4. Attaches and displays reply to user
Alternate Flow:	-
Post Condition	Chatbot is ready to service the next request

Use Case Number:	UC-02
Use Case Name:	Lambda Functions
Overview	Call respective lambda functions to store the user input, handle the user request and send a response
Actors:	AWS Lambda function Store, AWS Lambda function getRequest, AWS Lambda function reply, DynamoDB, AWS Cloudwatch
Pre-Condition:	Functions are available
Flow:	1. Lambda functions receive input from Chatbot and corresponding functionalities 2. Handle request appropriately and return response to the chatbot 3. Metrics are sent to the Cloudwatch Dashboard
Alternate Flow:	-
Post Condition	Functions are warm and available to service a new request

Use Case Number:	UC-03
Use Case Name:	Data Visualisation
Overview	Data Visualisation is achieved through the cloudwatch metrics dashboard available on AWS
Actors:	AWS Lambda, AWS Cloudwatch Metrics, AWS Cloudwatch
Pre-Condition:	Functions are available
Flow:	1. Lambda function data is obtained via AWS Cloudwatch 2. Metrics are displayed on the dashboard
Alternate Flow:	-
Post Condition	Dashboard remains online in order to facilitate further events

Use Case Number:	UC-04
Use Case Name:	EC2 Instance
Overview	A tiny EC2 instance is kept running in order to route lambda function calls from the chatbot to warm environments
Actors:	AWS EC2, AWS Lambda
Pre-Condition:	Lambda functions are available, EC2 instance is running
Flow:	1. EC2 instance receives call to Lambda function from the chatbot 2. Checks if any Lambda functions are warm 3. No Lambda functions are warm, so handles request manually while concurrently warming a Lambda
Alternate Flow:	1. Lambda function is already warm and EC2 instance is aware of this 2. Request is routed to the already warm function
Post Condition	EC2 instance is running