Lambda Performance Testing Strategy

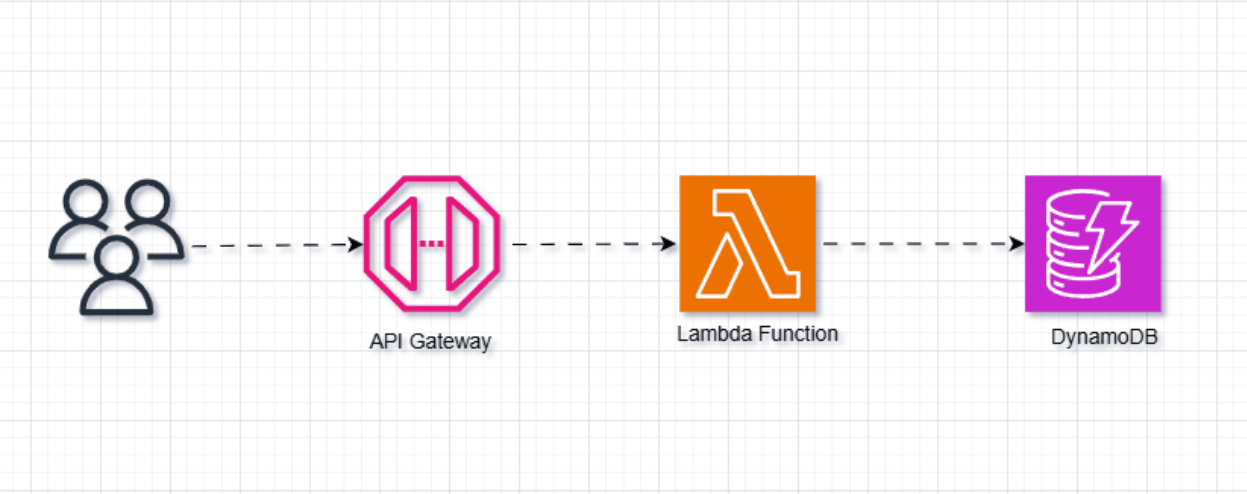
Prepared by
Biswajit Dhar

INTRODUCTION

Understanding the memory and cost dynamics of AWS Lambda, along with its behavior under load, is essential for building scalable and cost-efficient serverless applications. This project delivers a Lambda-powered API pipeline and evaluates its performance using two complementary approaches:

Lambda Power Tuning for optimizing memory and cost
Postman Load Testing for simulating real-world traffic

Together, these tests provide actionable insights into how memory settings affect execution time and cost, and how the system responds under concurrent load—enabling informed decisions that balance speed, reliability, and budget.



This document is structured into four comprehensive sections, each designed to guide through the implementation, evaluation, and optimization of a serverless pipeline using AWS services:

1. **SERVERLESS PIPELINE DEPLOYMENT**

This section outlines the end-to-end setup of a serverless architecture using AWS Lambda, DynamoDB, and API Gateway. It includes:

- Creating IAM policies and roles for Lambda execution
- Building the Lambda function to handle CRUD operations
- Setting up a DynamoDB table for persistent storage
- Configuring API Gateway to expose the Lambda function via HTTP
- Testing the integration using Postman and command-line tools

The goal is to establish a functional pipeline that can ingest and retrieve data through a RESTful interface backed by serverless compute and storage.

2. **LAMBDA POWER TUNING FOR MEMORY AND COST OPTIMIZATION**

This section leverages the AWS Lambda Power Tuning framework to benchmark the Lambda function across multiple memory configurations (128 MB, 256 MB, 512 MB, 1024 MB). It covers:

- Deploying the power tuning orchestration via the Serverless Application Repository

- Executing the Step Function with a controlled payload
- Analyzing the resulting performance graph to understand trade-offs between execution time and cost

This helps identify the optimal memory setting for your Lambda function based on your workload characteristics and business priorities.

3. POSTMAN LOAD TESTING SIMULATION

Here, the focus shifts to simulating real-world traffic using Postman's performance testing feature. You'll:

- Configure virtual users and duration to simulate concurrent API calls
- Capture metrics such as average response time, throughput, and latency percentiles (P90, P95, P99)
- Interpret the results to assess scalability, reliability, and responsiveness under load

This test complements the power tuning analysis by revealing how the system behaves under stress and concurrency.

4. KEY FINDINGS AND RECOMMENDATIONS

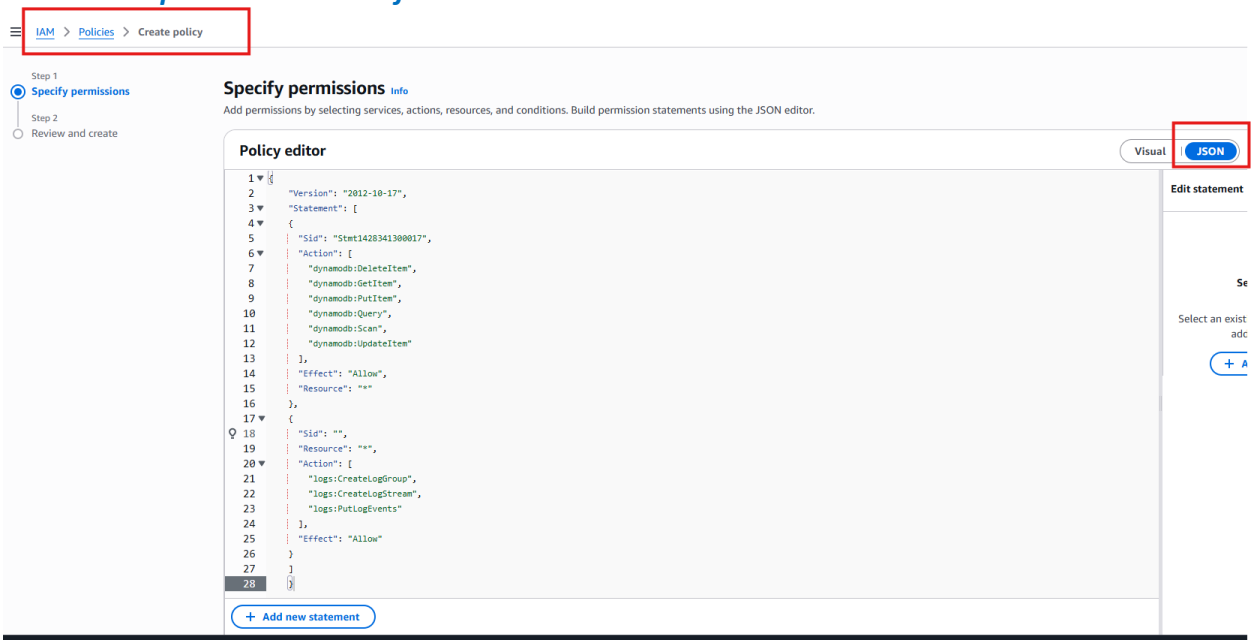
The final section synthesizes insights from both tuning and load testing. It provides:

- A comparative analysis of memory settings and their impact on cost and latency
- Observations on cold start behavior and performance spikes
- Strategic recommendations for memory allocation, concurrency handling, and API design

These findings are intended to guide future deployments and ensure your serverless architecture is both performant and cost-effective.

SERVERLESS PIPELINE DEPLOYMENT

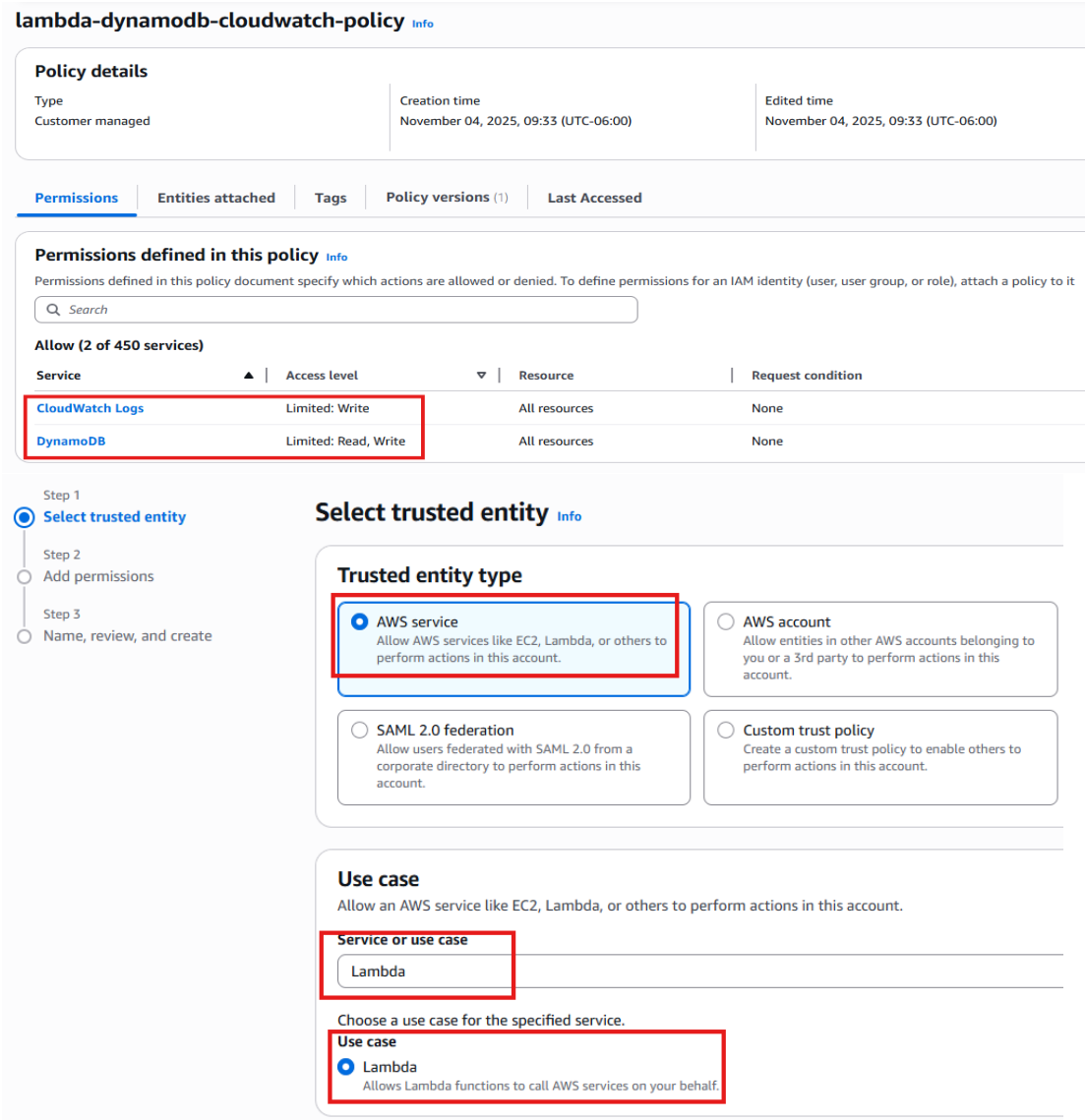
Create IAM policies and roles for Lambda execution



Add this json content:


```
{
  "Version": "2012-10-17",
  "Statement": [
  {
   "Sid": "Stmt1428341300017",
   "Action": [
    "dynamodb:DeleteItem",
    "dynamodb:GetItem",
    "dynamodb:PutItem",
    "dynamodb:Query",
    "dynamodb:Scan",
    "dynamodb:UpdateItem"
   ],
   "Effect": "Allow",
   "Resource": "*"
  },
  {
   "Sid": "",
   "Resource": "*",
   "Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents"
```

```
  ],
  "Effect": "Allow"
 }
 ]
 }
```



Add permissions Info

Permissions policies (1/1079) Info

Choose one or more policies to attach to your new role.

Filter by Type

dyna

All types

Policy name	Type
AmazonDynamoDBFullAccess	AWS managed
AmazonDynamoDBFullAccess_v2	AWS managed
AmazonDynamoDBFullAccesswithDataPipeline	AWS managed
AmazonDynamoDBReadOnlyAccess	AWS managed
AWSLambdaDynamoDBExecutionRole	AWS managed
AWSLambdaInvocation-DynamoDB	AWS managed
lambda-dynamodb-cloudwatch-policy	Customer managed

▶ Set permissions boundary - *optional*

Role name

Enter a meaningful name to identify this role.

lambda-apig-role

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description

Add a short explanation for this role.

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`

Step 1: Select trusted entities

Trust policy

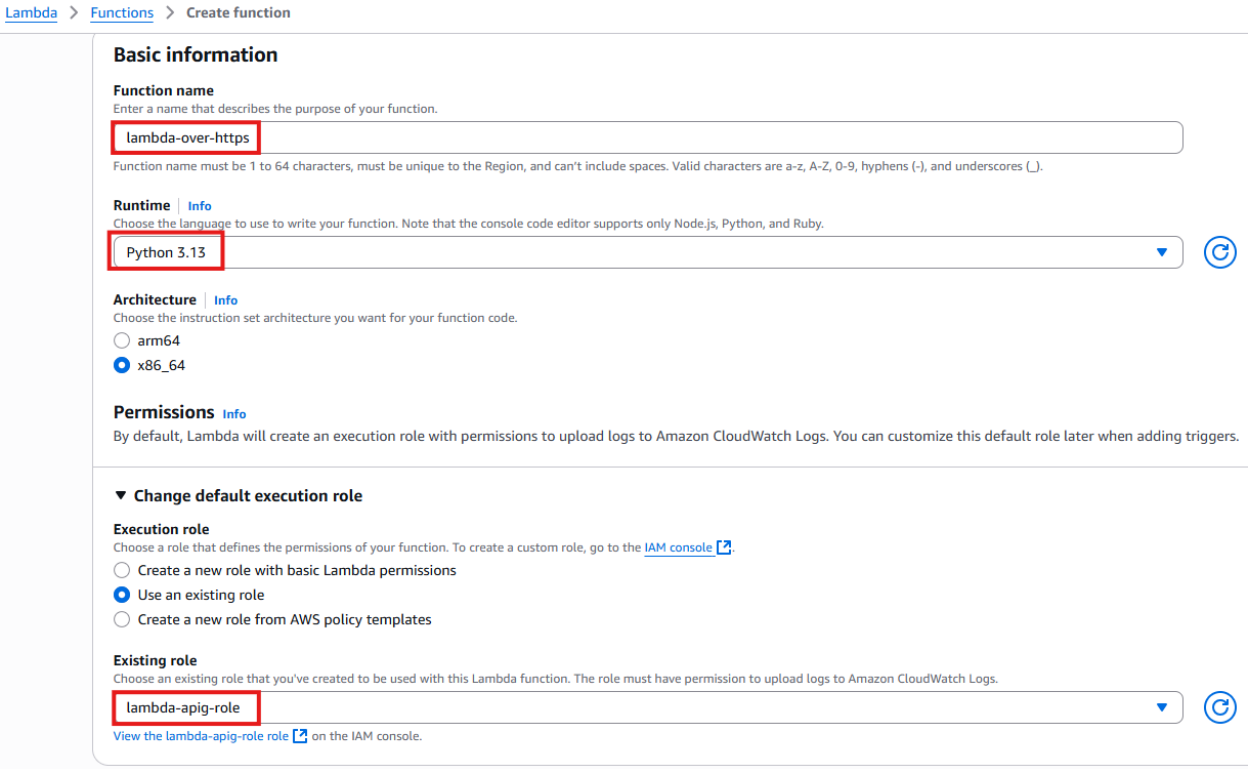
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            }
        }
    ]
}
```

Step 2: Add permissions

Permissions policy summary

Policy name	Type
lambda-dynamodb-cloudwatch-policy	Customer managed

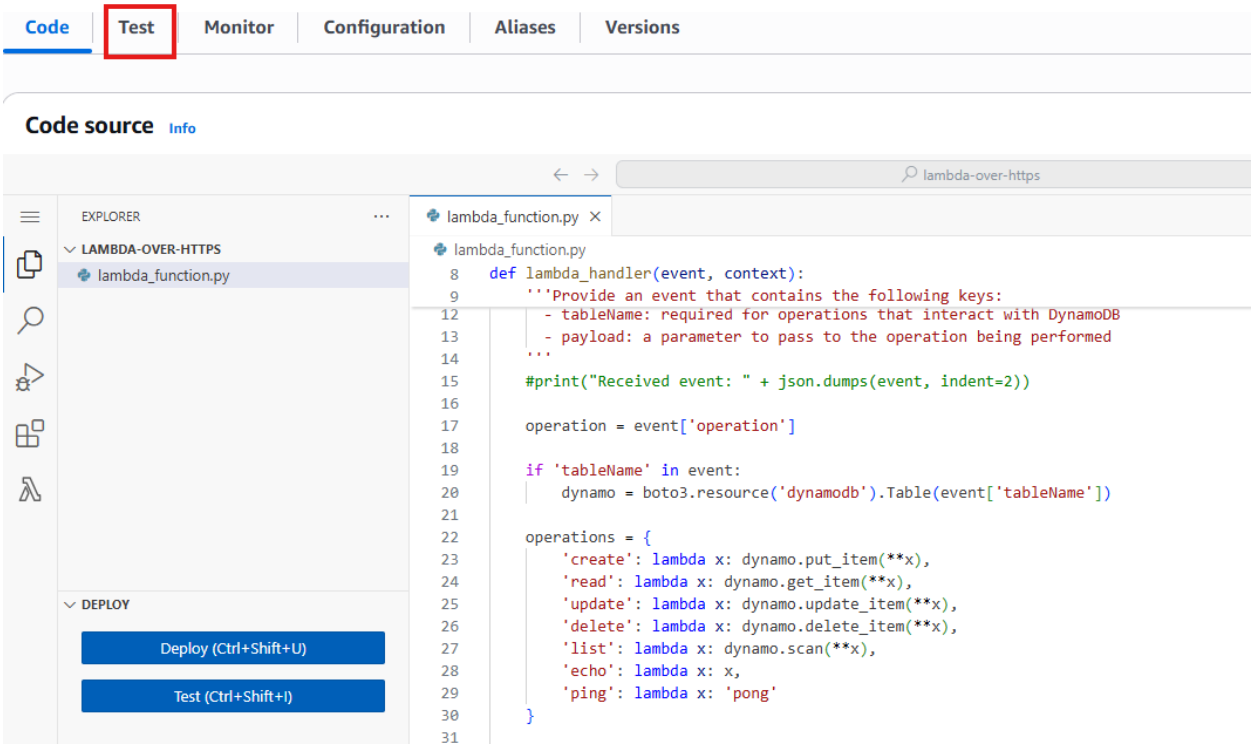
Build the Lambda function to handle CRUD operations



Add these python code for CRUD operations on DynamoDB:

```
from __future__ import print_function
import boto3
import json

print('Loading function')


def lambda_handler(event, context):
    '''Provide an event that contains the following keys:

      - operation: one of the operations in the operations dict below
      - tableName: required for operations that interact with DynamoDB
      - payload: a parameter to pass to the operation being performed
    '''
    #print("Received event: " + json.dumps(event, indent=2))

    operation = event['operation']

    if 'tableName' in event:
```

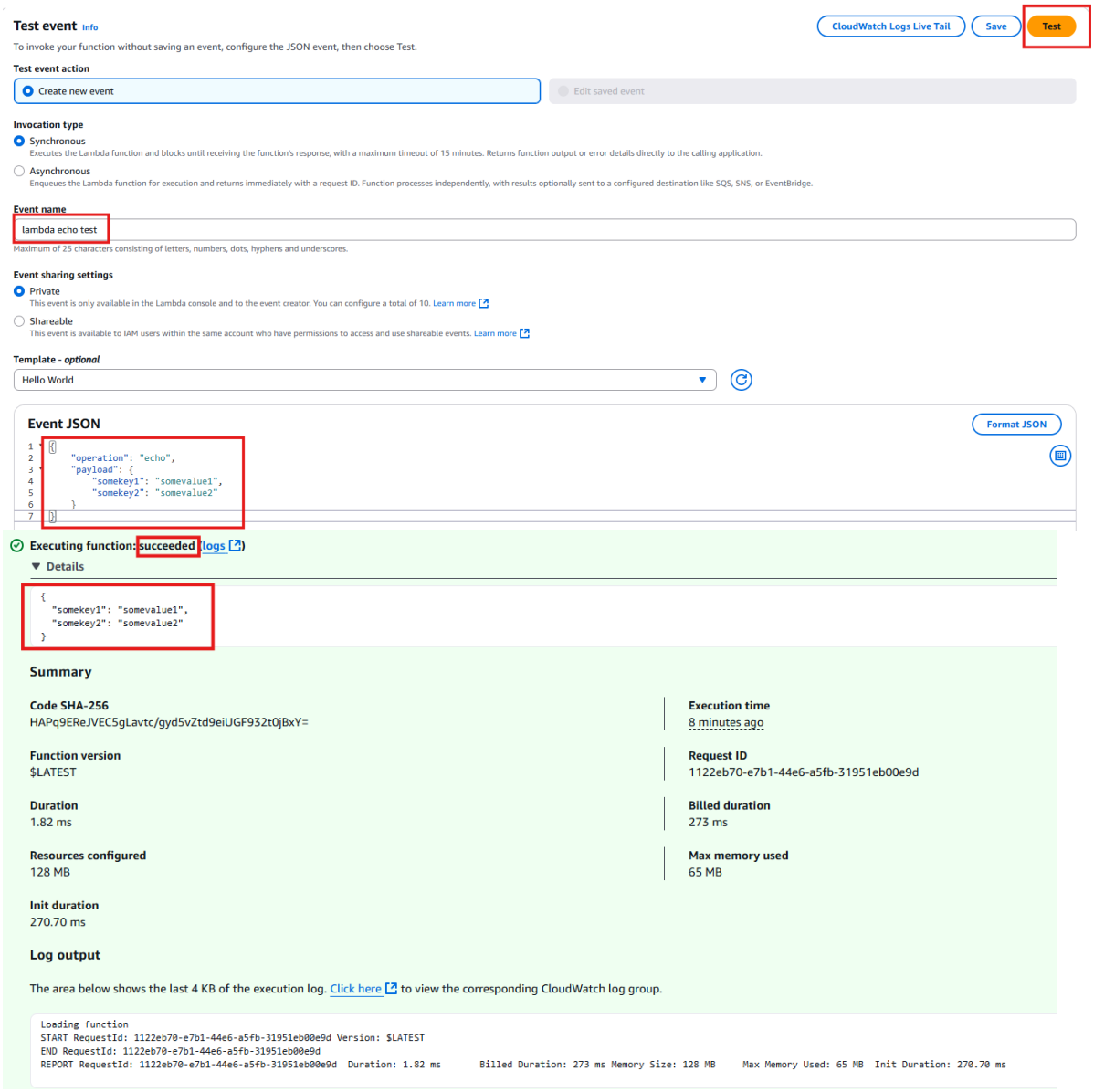
```
    dynamo = boto3.resource('dynamodb').Table(event['tableName'])

operations = {
    'create': lambda x: dynamo.put_item(**x),
    'read': lambda x: dynamo.get_item(**x),
    'update': lambda x: dynamo.update_item(**x),
    'delete': lambda x: dynamo.delete_item(**x),
    'list': lambda x: dynamo.scan(**x),
    'echo': lambda x: x,
    'ping': lambda x: 'pong'
}

if operation in operations:
    return operations[operation](event.get('payload'))
else:
    raise ValueError('Unrecognized operation "{}"'.format(operation))
```



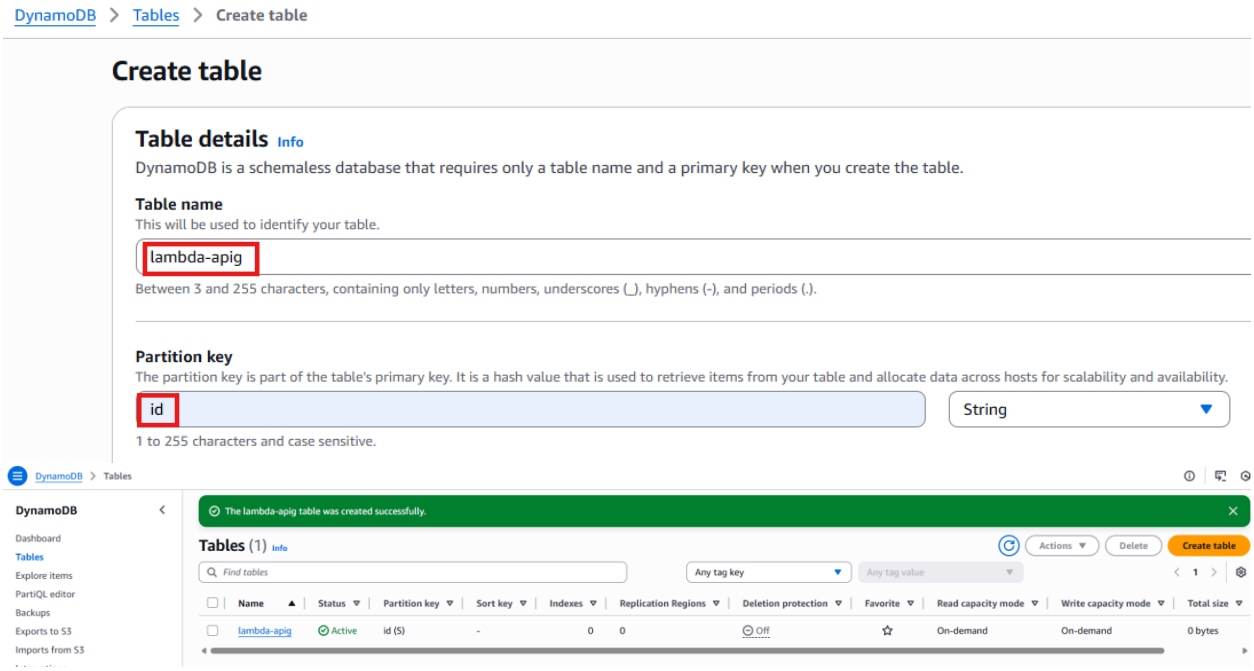
Add following json code to test:

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
```

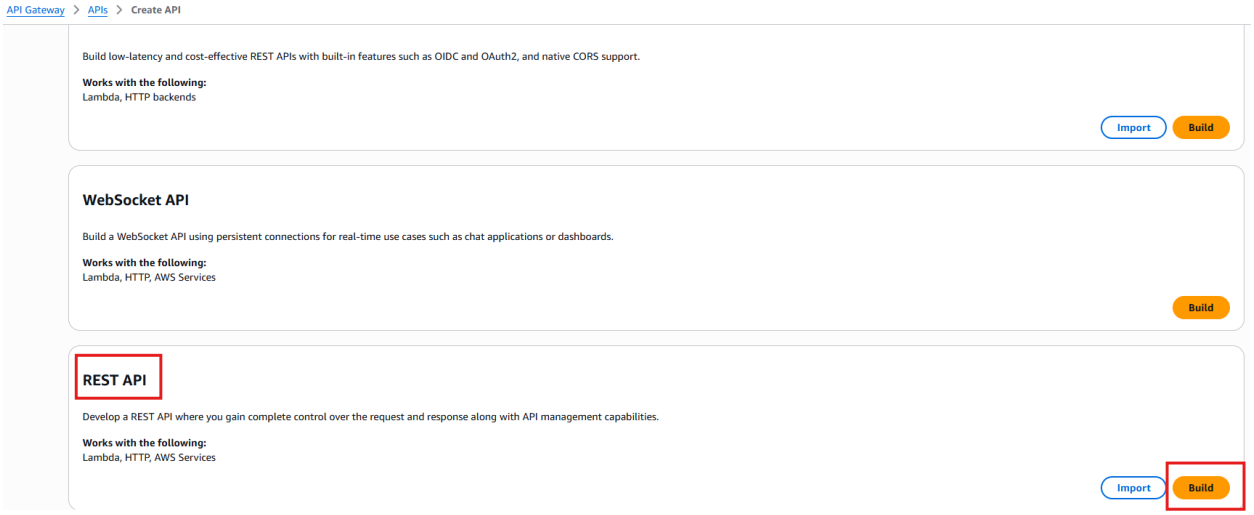
```
}
```

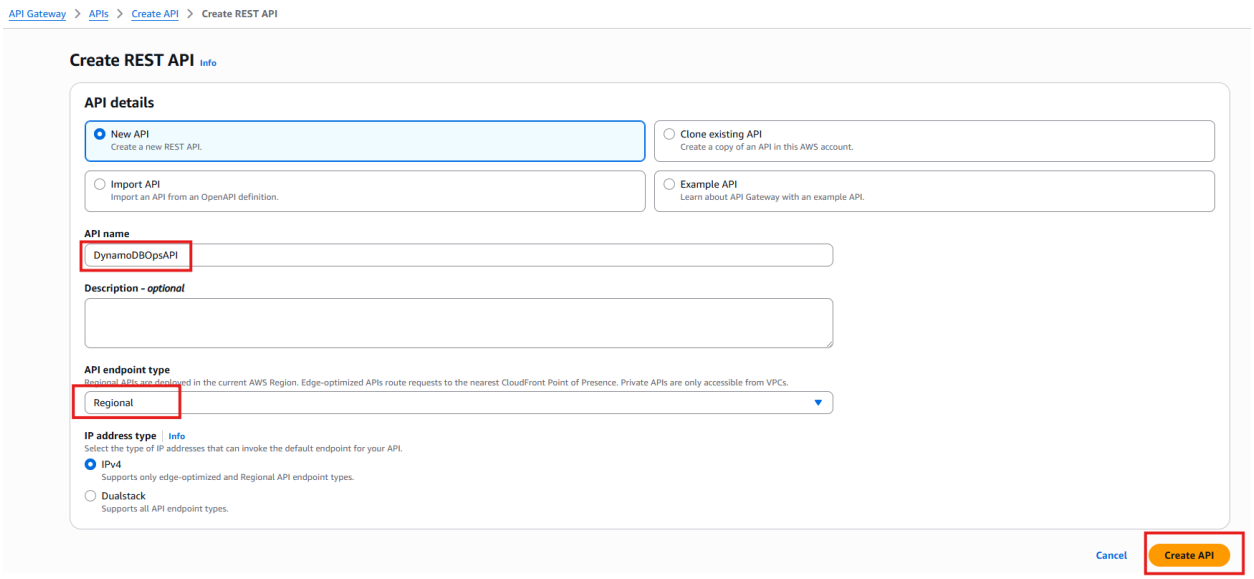

Test event Info
CloudWatch Logs Live Tail
Save
Test
To invoke your function without saving an event, configure the JSON event, then choose Test.
Test event action
Create new event
Edit saved event
Invocation type
Synchronous
Executes the Lambda function and blocks until receiving the function's response, with a maximum timeout of 15 minutes. Returns function output or error details directly to the calling application.
Asynchronous
Enqueues the Lambda function for execution and returns immediately with a request ID. Function processes independently, with results optionally sent to a configured destination like SQS, SNS, or EventBridge.
Event name
lambda echo test
Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.
Event sharing settings
Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more
Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more
Template - optional
Hello World
Event JSON
Format JSON
{
"operation": "echo",
"payload": {
"somekey1": "somevalue1",
"somekey2": "somevalue2"
}
}
Executing function: succeeded (logs)
Details
{
"somekey1": "somevalue1",
"somekey2": "somevalue2"
}
Summary
Code SHA-256
HAPq9EReJVEC5gLavtc/gyd5vZtd9eiUGF932t0jBxY=
Execution time
8 minutes ago
Function version
$LATEST
Request ID
1122eb70-e7b1-44e6-a5fb-31951eb00e9d
Duration
1.82 ms
Billed duration
273 ms
Resources configured
128 MB
Max memory used
65 MB
Init duration
270.70 ms
Log output
The area below shows the last 4 KB of the execution log. Click here to view the corresponding CloudWatch log group.
Loading function
START RequestId: 1122eb70-e7b1-44e6-a5fb-31951eb00e9d Version: $LATEST
END RequestId: 1122eb70-e7b1-44e6-a5fb-31951eb00e9d
REPORT RequestId: 1122eb70-e7b1-44e6-a5fb-31951eb00e9d Duration: 1.82 ms Billed Duration: 273 ms Memory Size: 128 MB Max Memory Used: 65 MB Init Duration: 270.70 ms

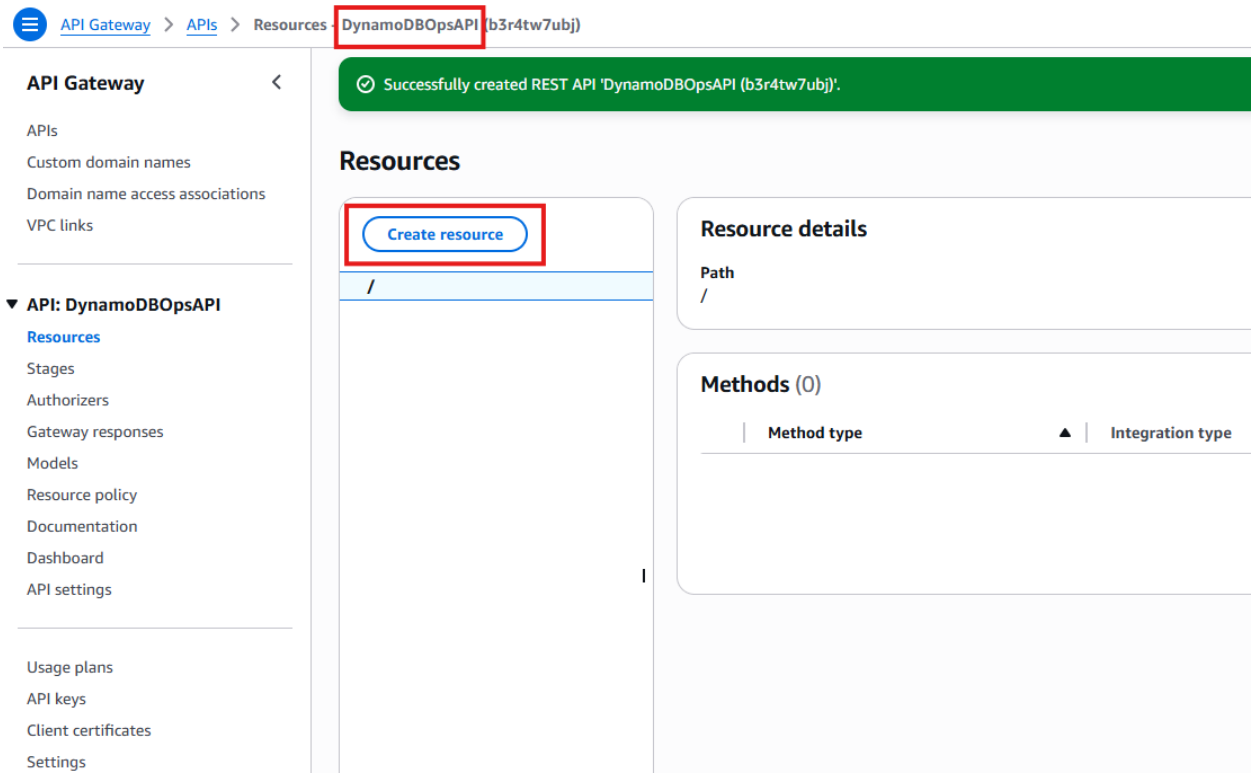
Setup DynamoDB Table

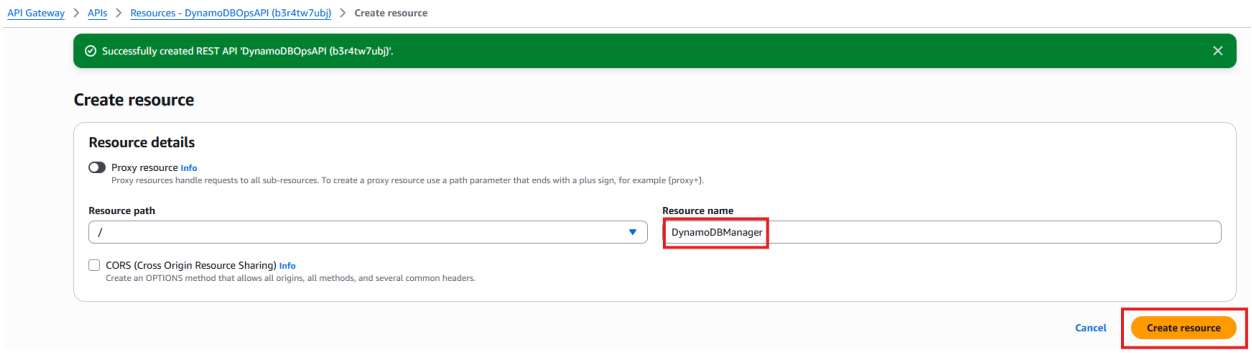


Configure API Gateway

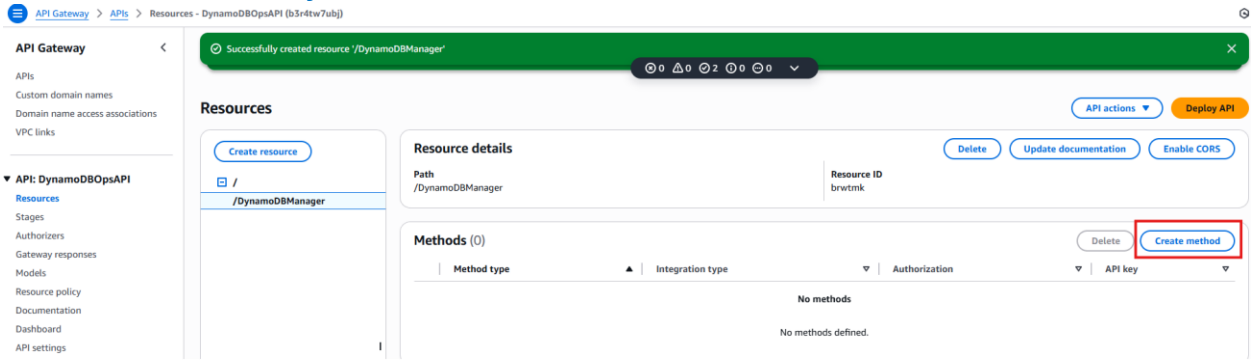


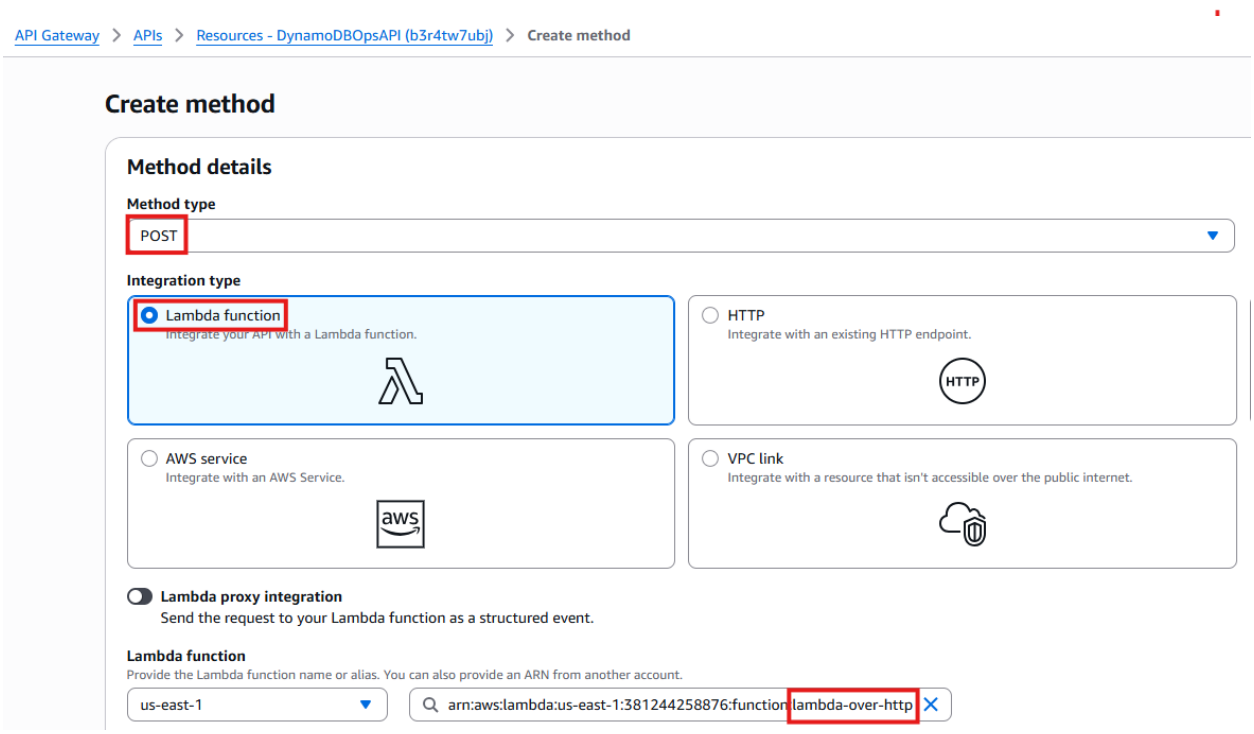

API Gateway > APIs > Create API > Create REST API
Create REST API Info
API details
New API
Create a new REST API.
Clone existing API
Create a copy of an API in this AWS account.
Import API
Import an API from an OpenAPI definition.
Example API
Learn about API Gateway with an example API.
API name
DynamoDBOpsAPI
Description - optional
API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.
Regional
IP address type | Info
Select the type of IP addresses that can invoke the default endpoint for your API.
IPv4
Supports only edge-optimized and Regional API endpoint types.
Dualstack
Supports all API endpoint types.
Cancel
Create API


API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj)
API Gateway
APIs
Custom domain names
Domain name access associations
VPC links
API: DynamoDBOpsAPI
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
API keys
Client certificates
Settings
Successfully created REST API 'DynamoDBOpsAPI (b3r4tw7ubj)'.
Resources
Create resource
/
Resource details
Path
/
Methods (0)
Method type
Integration type

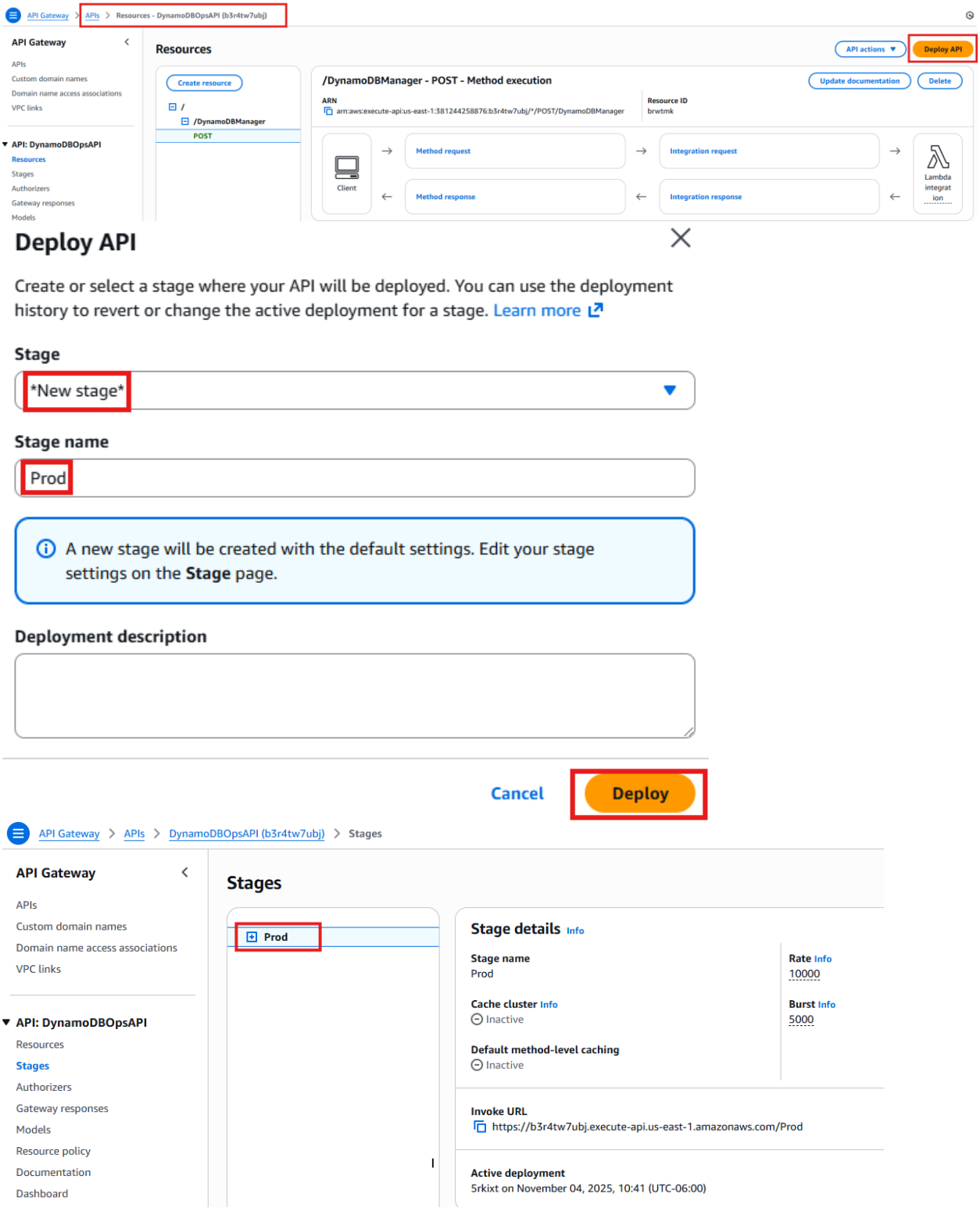


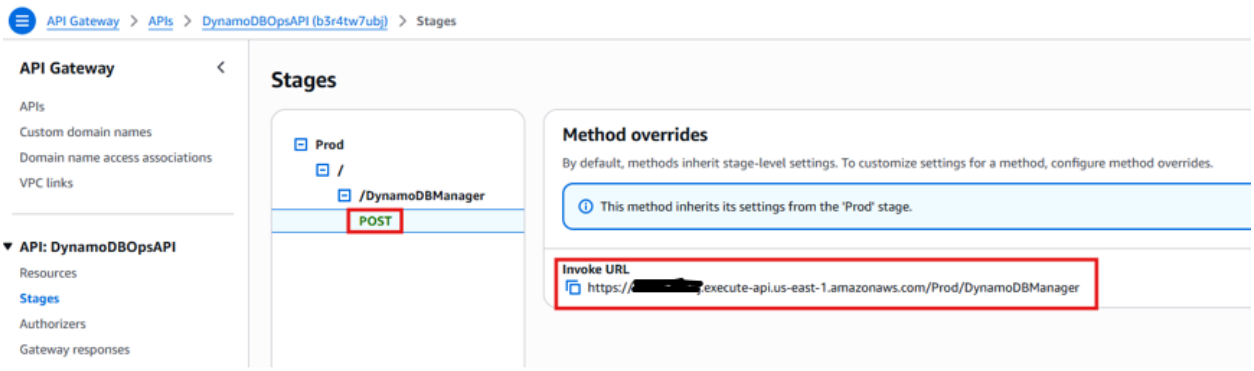
Create a POST Method for our API





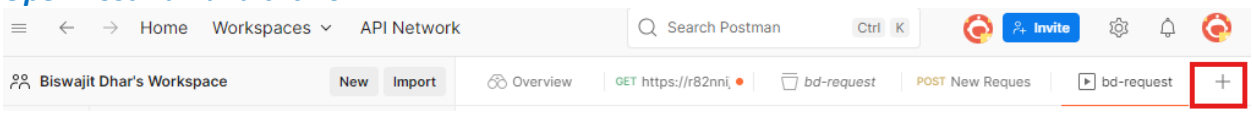
Deploy API


API Gateway
APIs
Resources - DynamoDBOpsAPI (b3r4tw7ubj)
Resources
API actions
Deploy API
Create resource
/DynamoDBManager - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:381244258876:b3r4tw7ubj/*/POST/DynamoDBManager
Resource ID
brwtmk
Method request
Integration request
Client
Method response
Integration response
Lambda integration
Deploy API
Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. Learn more
Stage
New stage
Stage name
Prod
A new stage will be created with the default settings. Edit your stage settings on the Stage page.
Deployment description
Cancel
Deploy
API Gateway
APIs
Custom domain names
Domain name access associations
VPC links
API: DynamoDBOpsAPI
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
Stages
Prod
Stage details Info
Stage name
Prod
Rate Info
10000
Cache cluster Info
Inactive
Burst Info
5000
Default method-level caching
Inactive
Invoke URL
https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod
Active deployment
5rkixt on November 04, 2025, 10:41 (UTC-06:00)

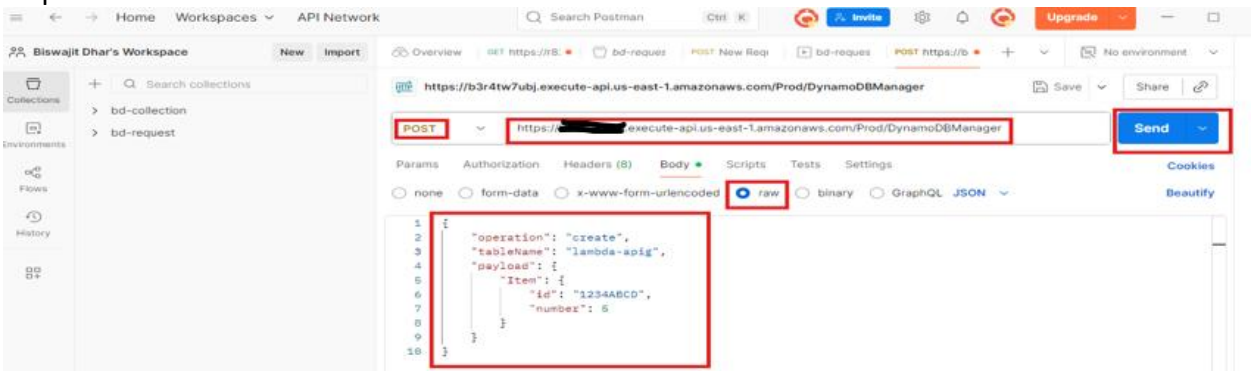


Testing Integration Using Postman and Command-line Tools

Open Postman and click on +

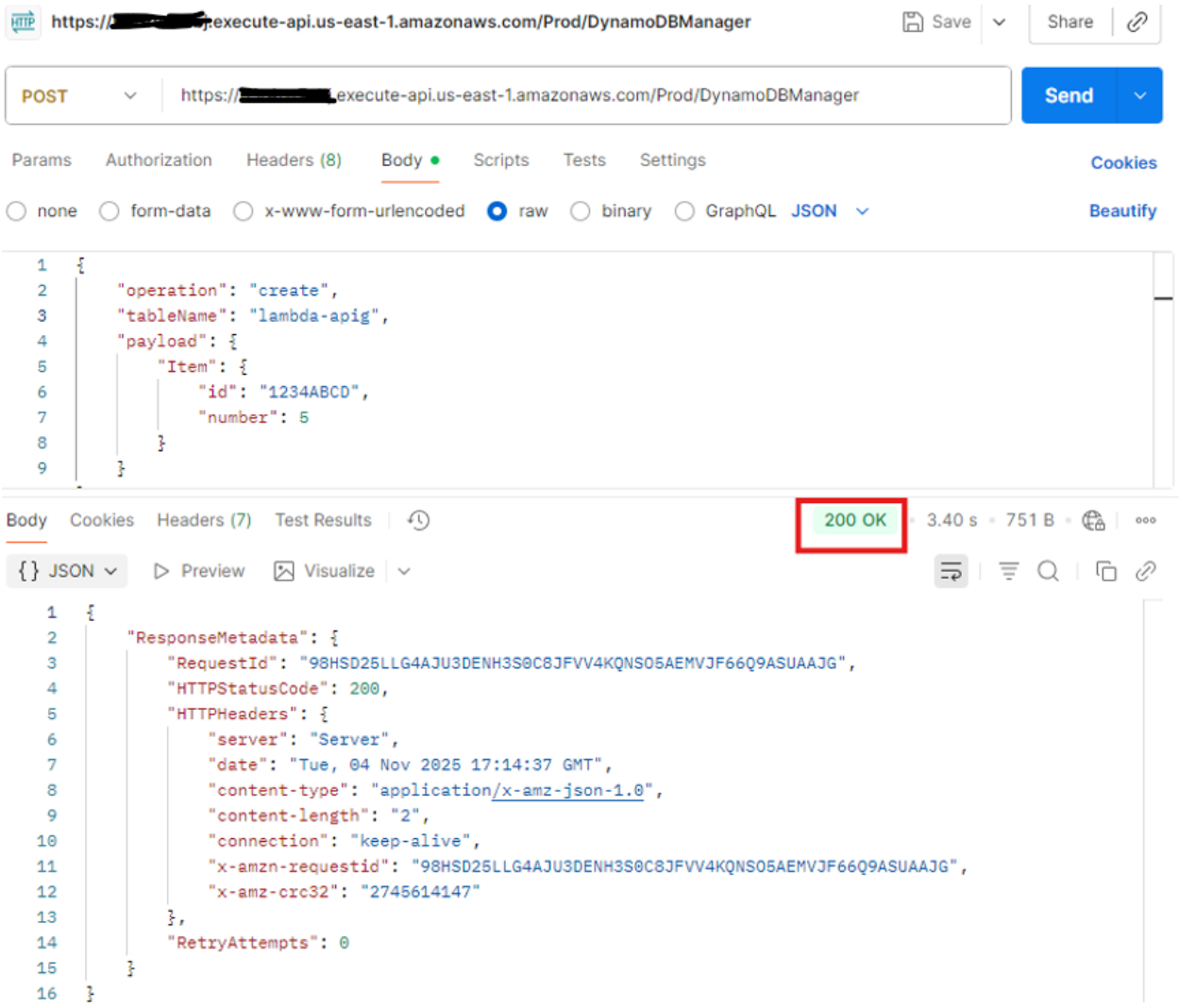


This will make API call using API Gateway endpoint, https://b3r***abc.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager, which will invoke lambda function, **lambda-over-http** to insert id and number payload into DynamoDB table **lambda-apig** created in previous steps.



Add following json content for Raw:

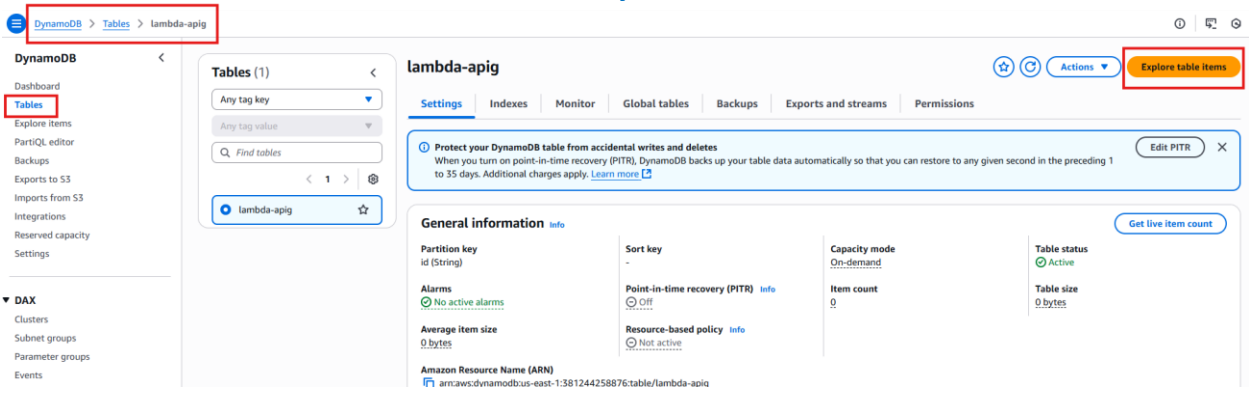
```
{
   "operation": "create",
   "tableName": "lambda-apigateway",
   "payload": {
      "Item": {
         "id": "1234ABCD",
         "number": 5
      }
   }
}
```



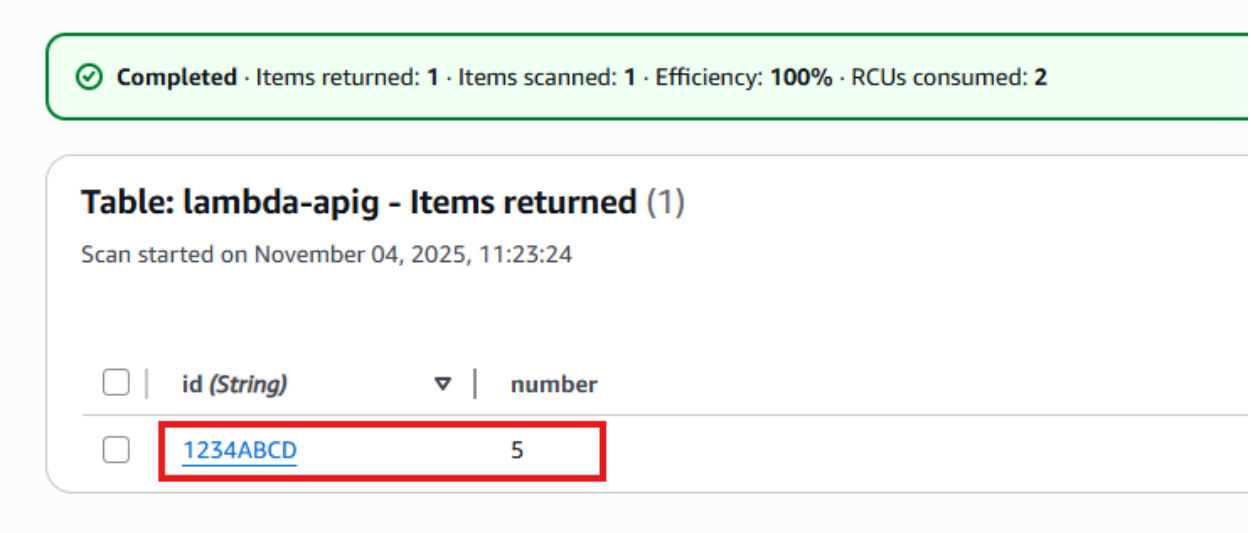
Alternatively, you can run this from command line

$ *curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-apig\",\"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}" https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager*

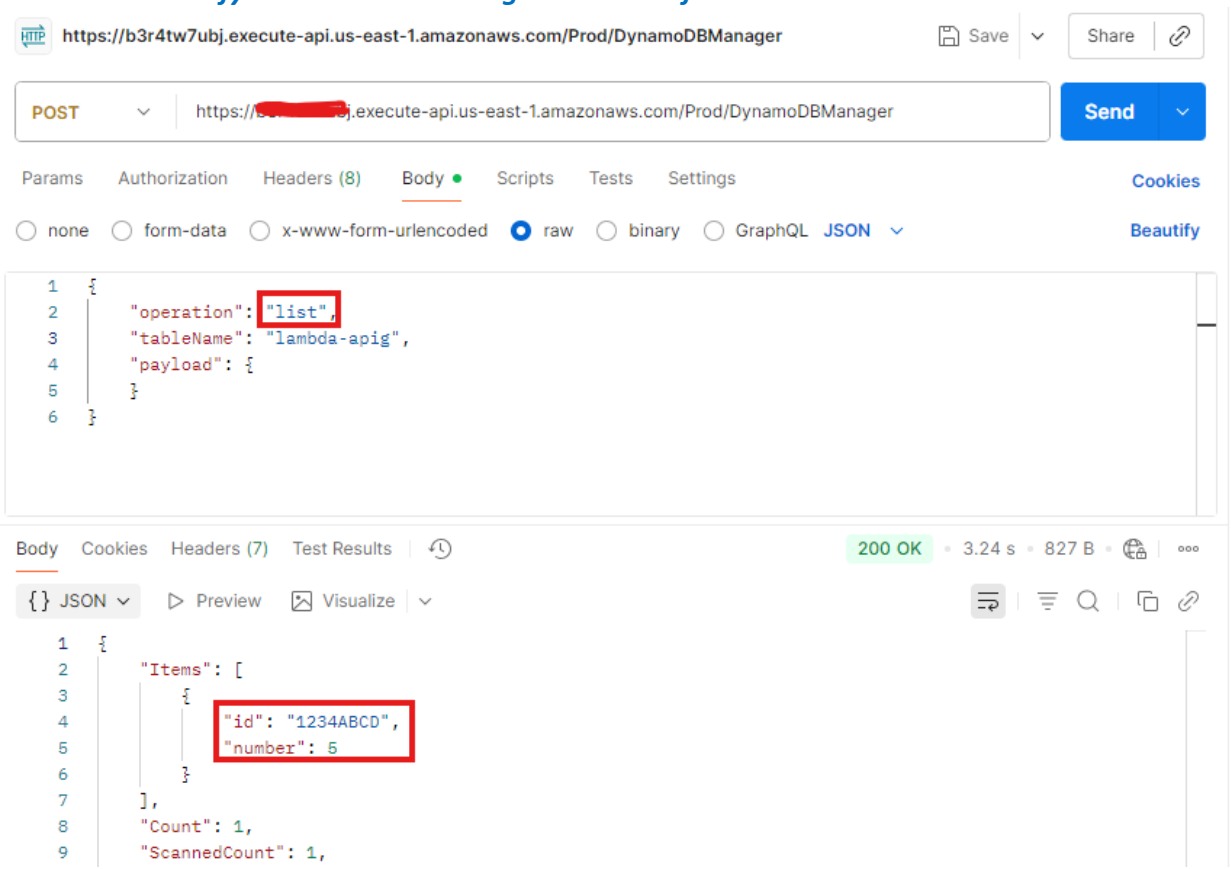
Validate that the item is indeed inserted into DynamoDB table



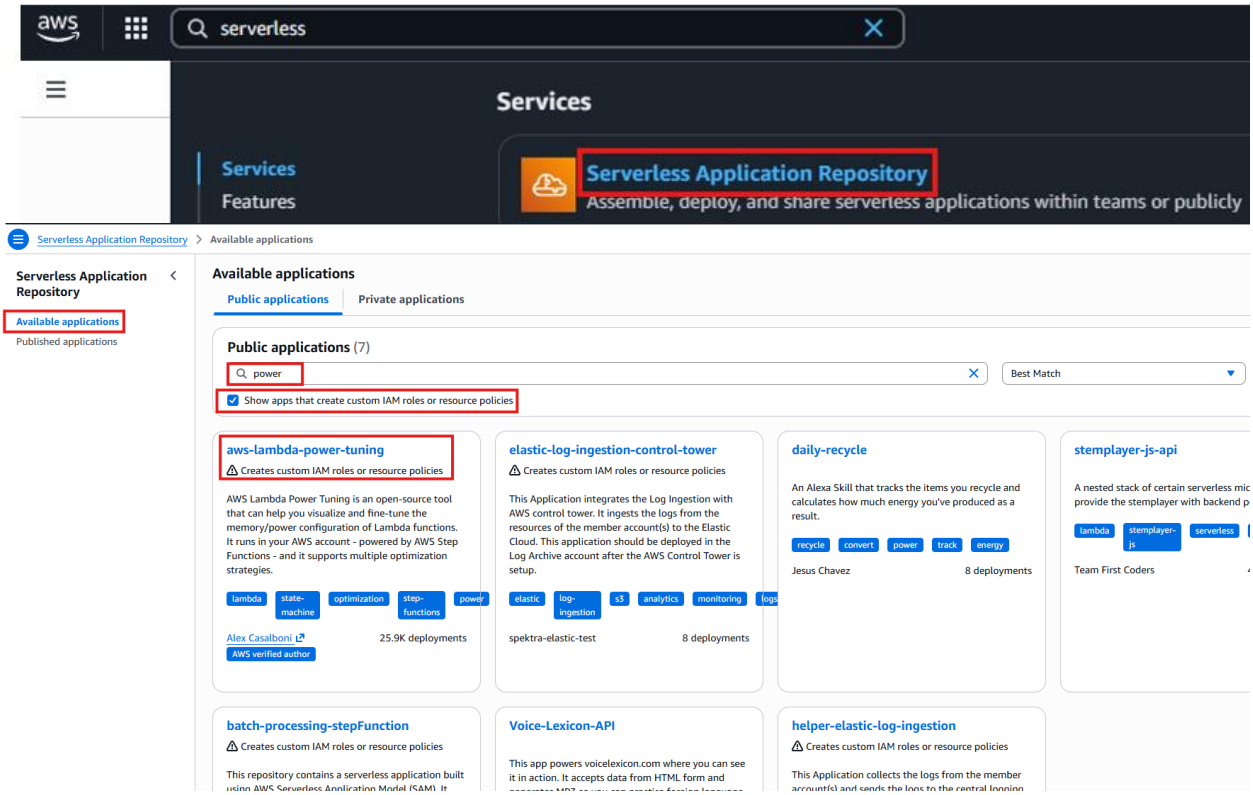
This is the payload for id and number were sent through Postman API call and showed up on DynamoDB table below.

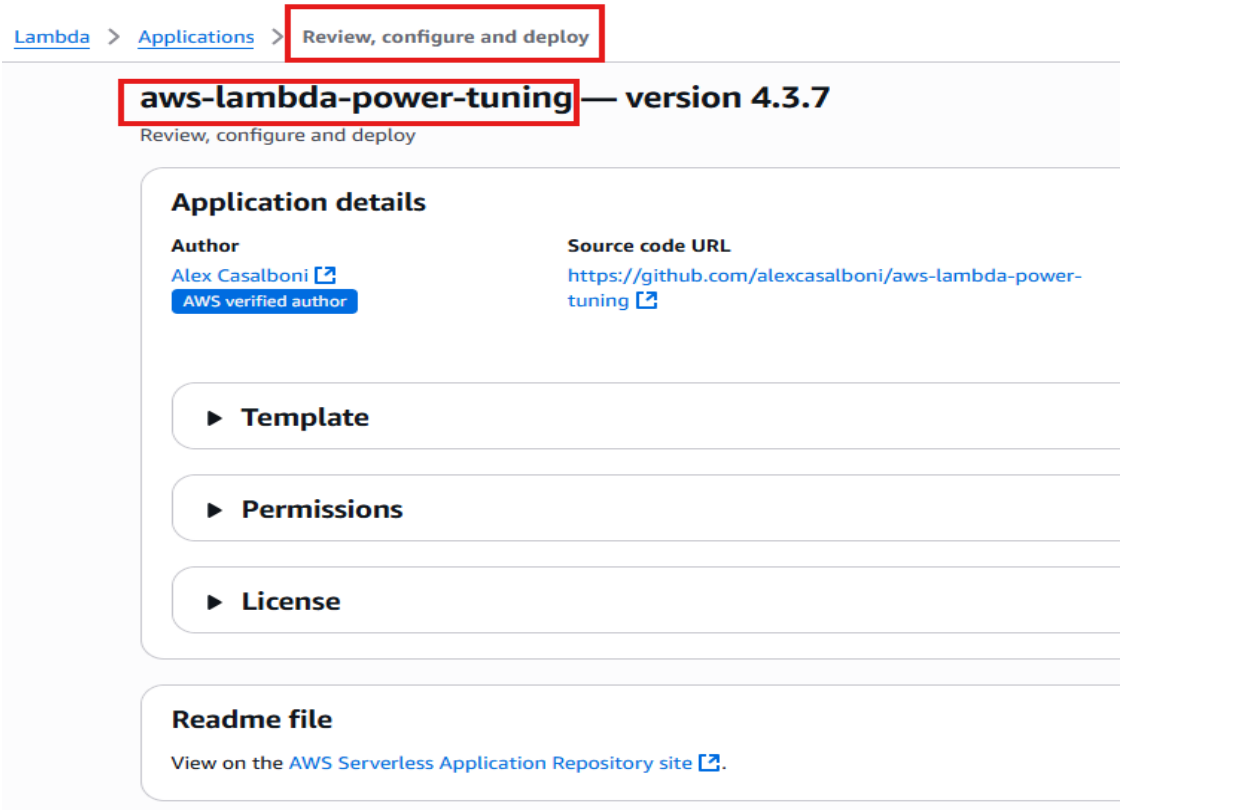


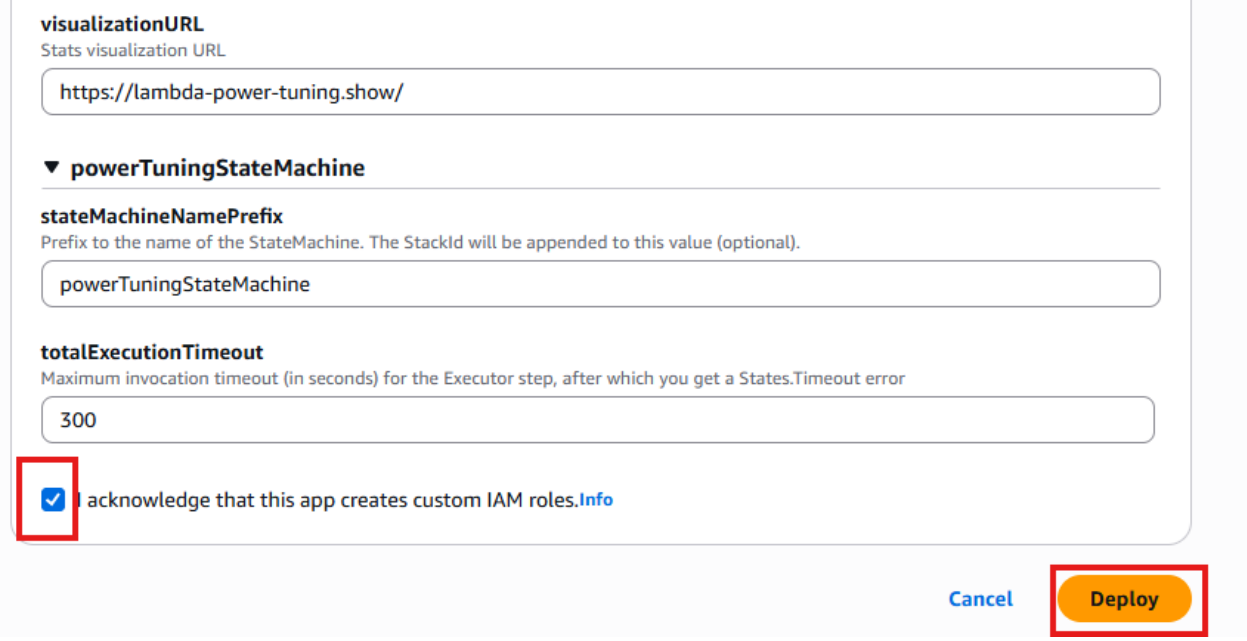
You can also verify inserted items using List API call from Postman



LAMBDA POWER TUNING



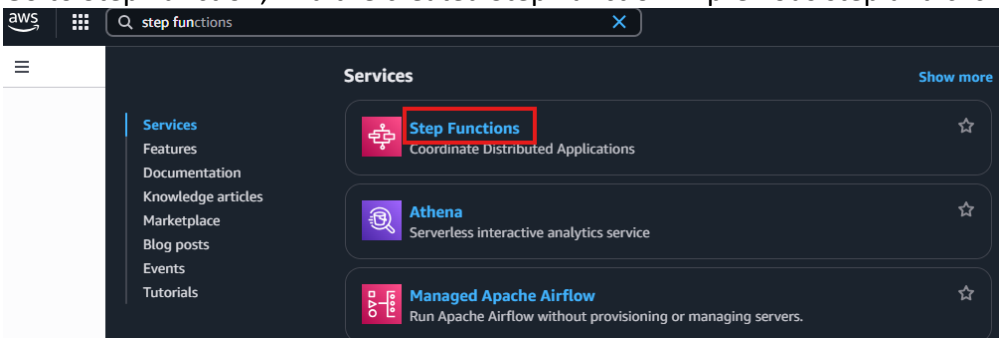


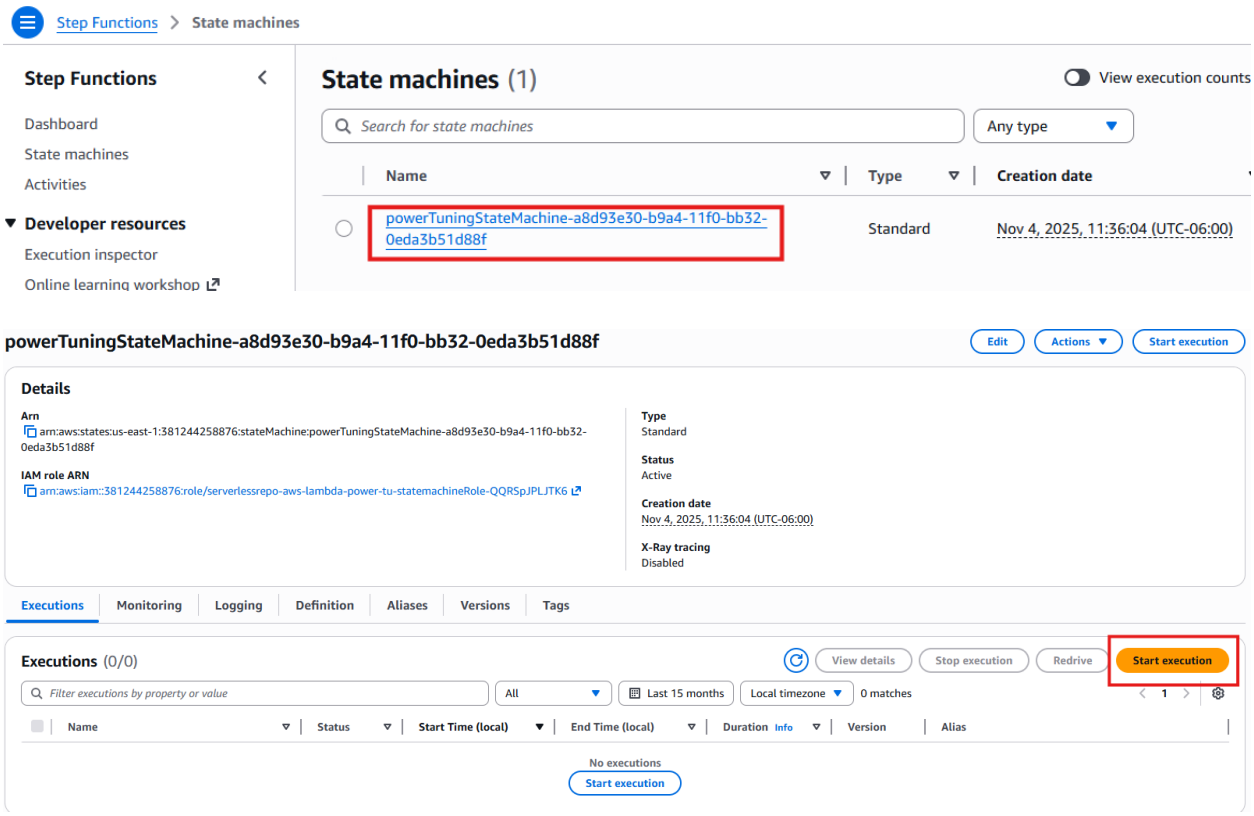


All these lambda functions, step function, IAM role and log groups are created by aws-lambda-power-tuning

Logical ID	Physical ID	Type
statemachineRole	serverlessrepo-aws-lambda-power-tu-statemachineRole-QQRSpJPLJTK6	IAM Role
analyzer	serverlessrepo-aws-lambda-power-tuning-analyzer-sHltMmXKAYbb	Lambda Function
cleaner	serverlessrepo-aws-lambda-power-tuning-cleaner-2kgIHUrlWLcd	Lambda Function
executor	serverlessrepo-aws-lambda-power-tuning-executor-9ex8V61rtghT	Lambda Function
initializer	serverlessrepo-aws-lambda-power-tuning-initializer-vABkYt6QIDHJ	Lambda Function
optimizer	serverlessrepo-aws-lambda-power-tuning-optimizer-VQ9bvu4d3REs	Lambda Function
publisher	serverlessrepo-aws-lambda-power-tuning-publisher-CZovYVC906OX	Lambda Function
SDKlayerb74fb6ab48	arn:aws:lambda:us-east-1:381244258876:layer:AWS-SDK-v3:1	Lambda LayerVersion
analyzerLogGroup	/aws/lambda/serverlessrepo-aws-lambda-power-tuning-analyzer-sHltMmXKAYbb	Logs LogGroup
cleanerLogGroup	/aws/lambda/serverlessrepo-aws-lambda-power-tuning-cleaner-2kgIHUrlWLcd	Logs LogGroup
executorLogGroup	/aws/lambda/serverlessrepo-aws-lambda-power-tuning-executor-9ex8V61rtghT	Logs LogGroup
initializerLogGroup	/aws/lambda/serverlessrepo-aws-lambda-power-tuning-initializer-vABkYt6QIDHJ	Logs LogGroup
optimizerLogGroup	/aws/lambda/serverlessrepo-aws-lambda-power-tuning-optimizer-VQ9bvu4d3REs	Logs LogGroup
powerTuningStateMachine	arn:aws:states:us-east-1:381244258876:stateMachine:powerTuningStateMachine-a8d93e30-b9a4-11f0-bb32-0eda3b51d88f	StepFunctions StateMachine

Go to Step Function, find the created Step Function in previous step and click on it





Enter the following with ARN for your lambda function and your DynamoDB table name:

```
{
 "lambdaARN": "arn:aws:lambda:us-east-1:38*****8876:function:lambda-over-https",
 "powerValues": [
  128,
  256,
  512,
  1024
 ],
 "num": 10,
 "payload": {
  "operation": "list",
  "tableName": "lambda-***db",
  "payload": {}
 },
 "parallelInvocation": true,
 "strategy": "cost"
}
```

Start execution

Name

1fa967aa-0bbb-4761-81a3-3fa2cc1f686f

Must be 1-80 characters. Can use alphanumeric characters, dashes, or underscores.

Input - *optional*

Enter input values for this execution in JSON format

Format JSON | Export | Import

```
{
  "lambdaARN": "arn:aws:lambda:us-east-1:[redacted]:function:lambda-over-https",
  "powerValues": [
    128,
    256,
    512,
    1024
  ],
  "num": 10,
  "payload": {
```

Start execution

Name

1fa967aa-0bbb-4761-81a3-3fa2cc1f686f

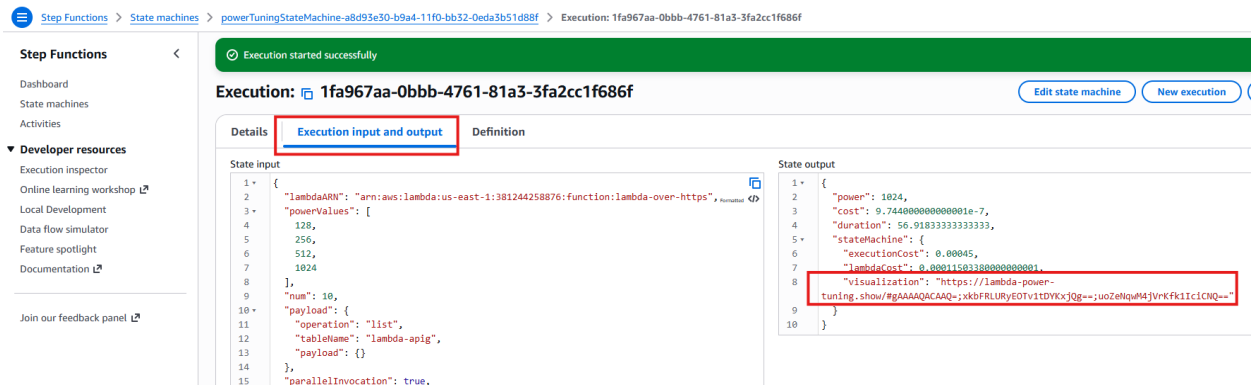
Must be 1-80 characters. Can use alphanumeric characters, dashes, or underscores.

Input - *optional*

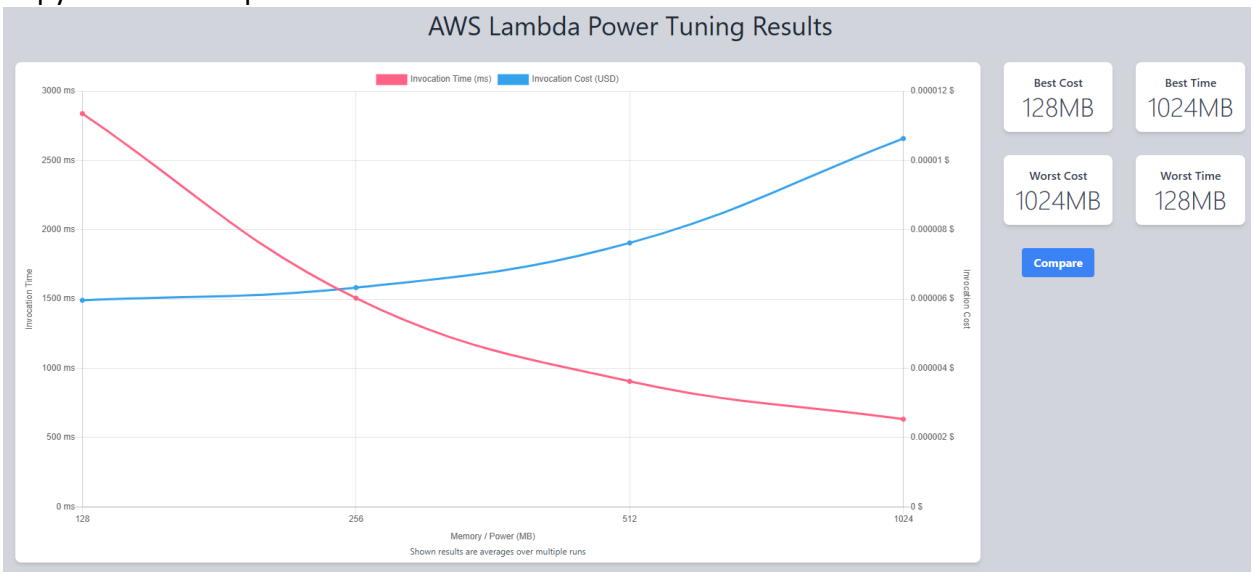
Enter input values for this execution in JSON format

Format JSON | Export | Import

```
  ],
  "num": 10,
  "payload": {
    "operation": "list",
    "tableName": "[redacted]",
    "payload": {}
  },
 "parallelInvocation": true,
  "strategy": "cost"
}
```



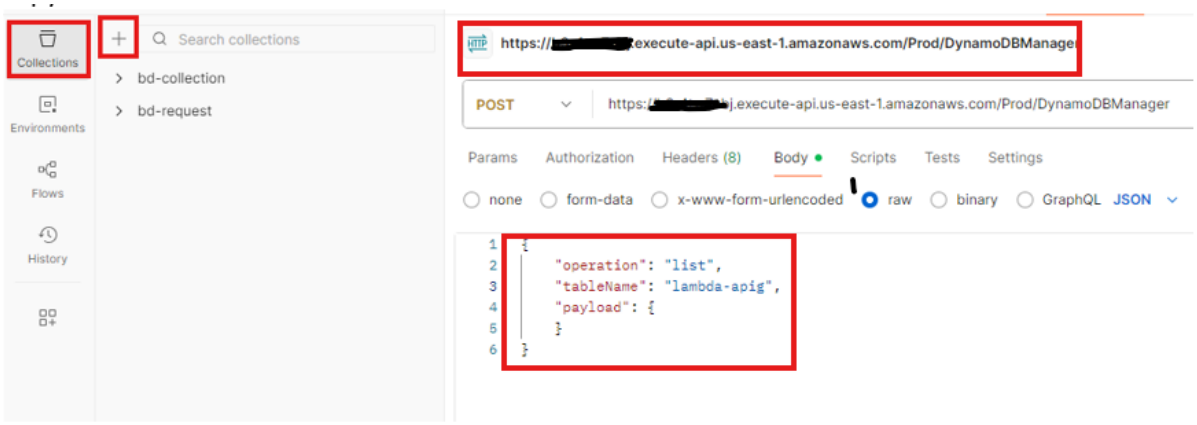
Copy url link and open it on a browser

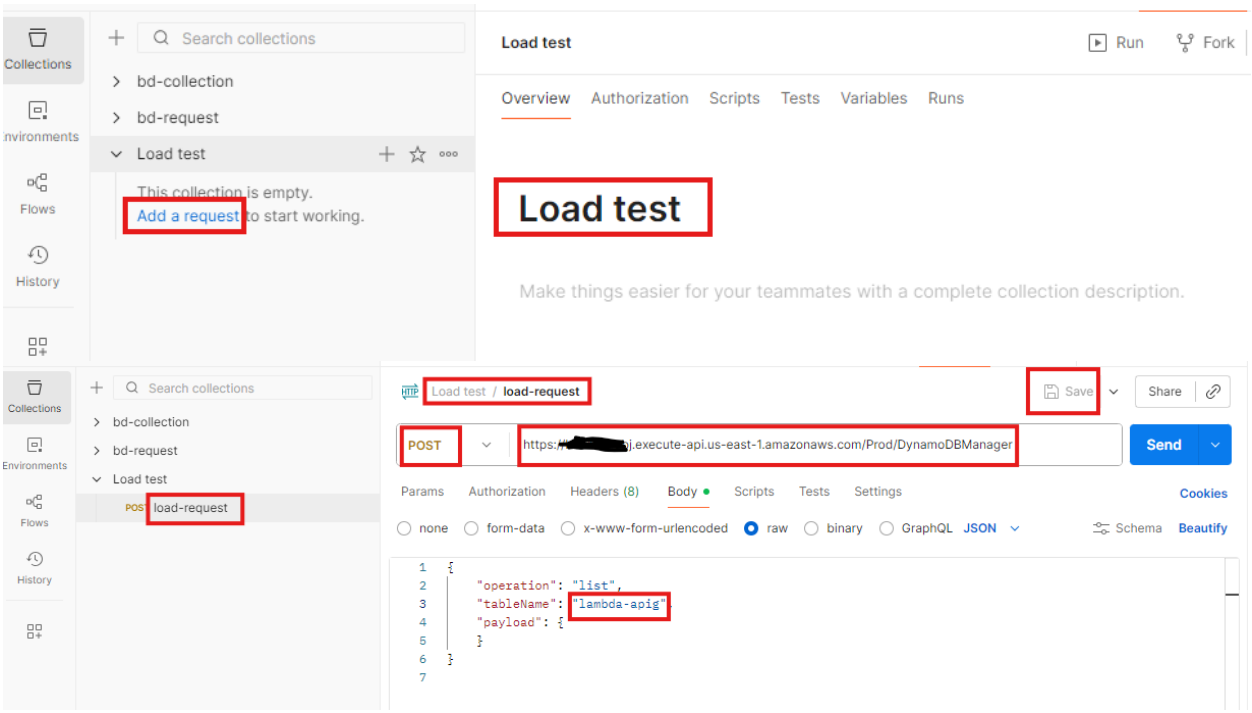


POSTMAN LOAD TESTING SIMULATION

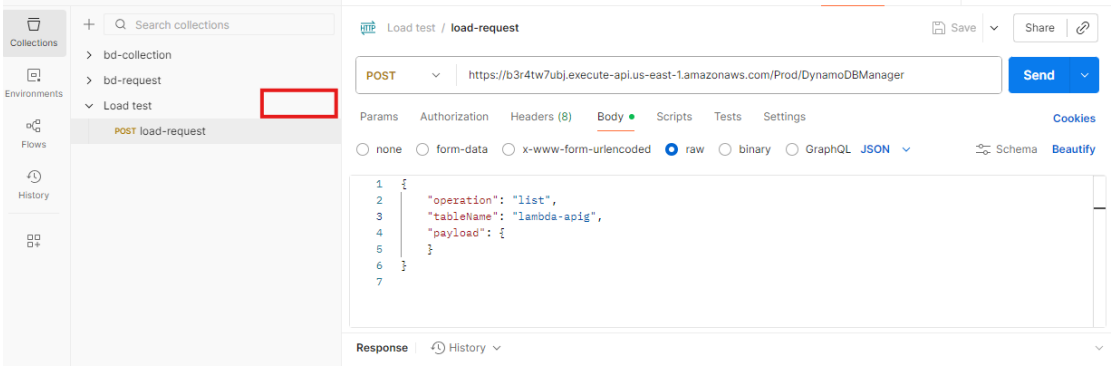
Create a Blank Collection

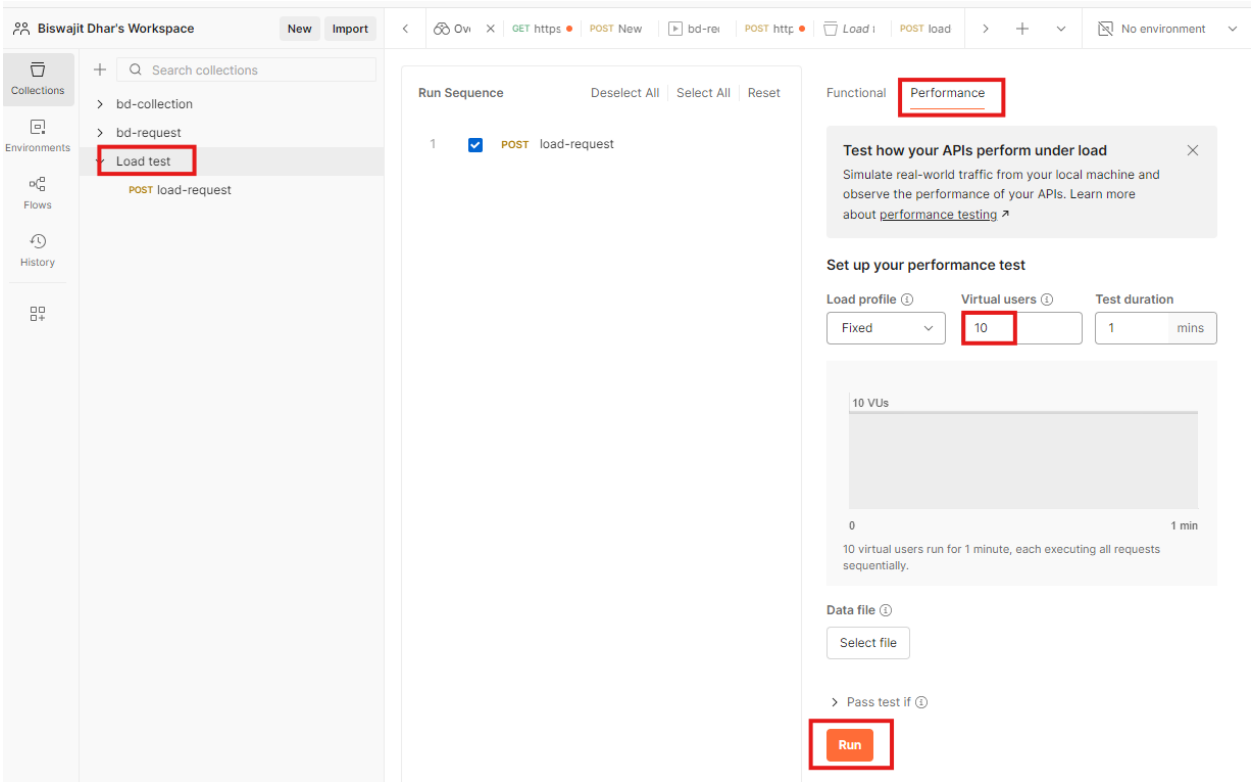
Copy and paste api endpoint url



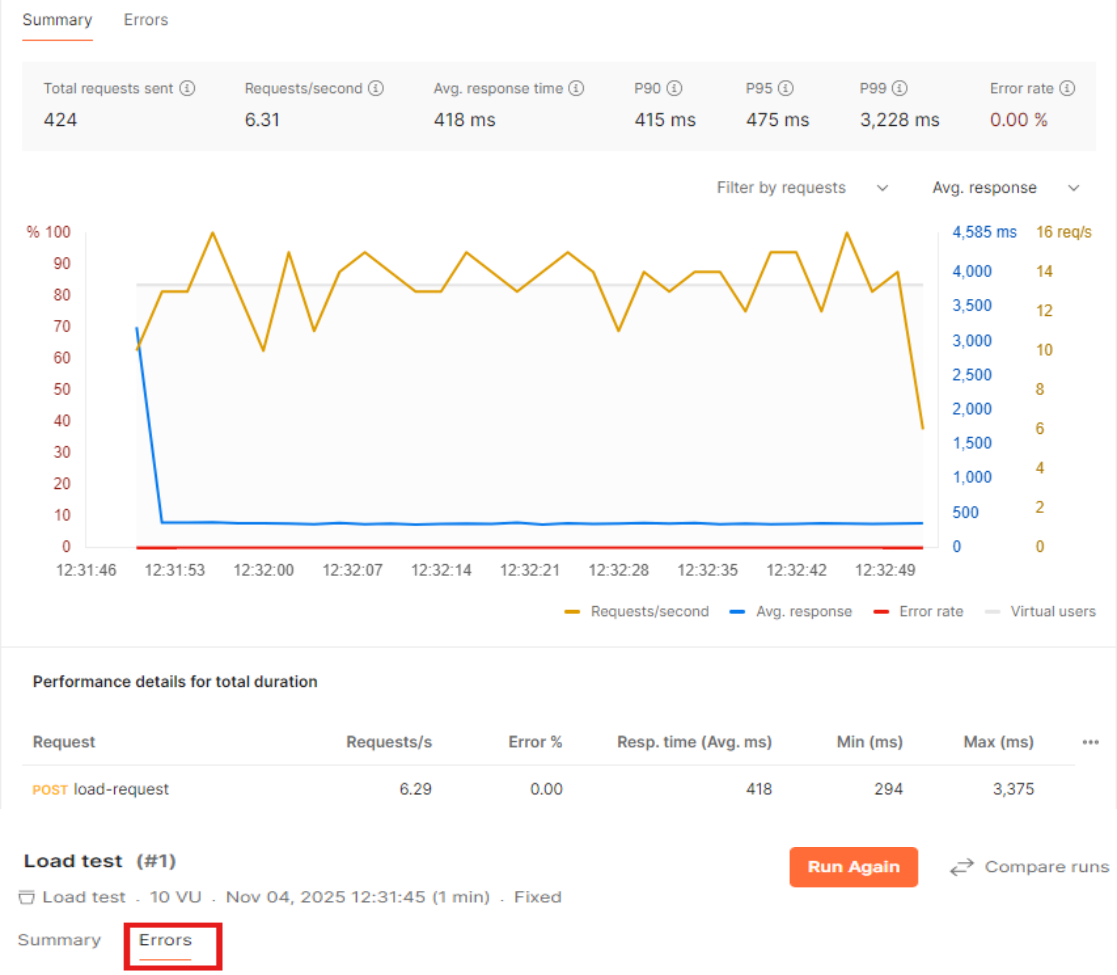


Click on three dots here and Run

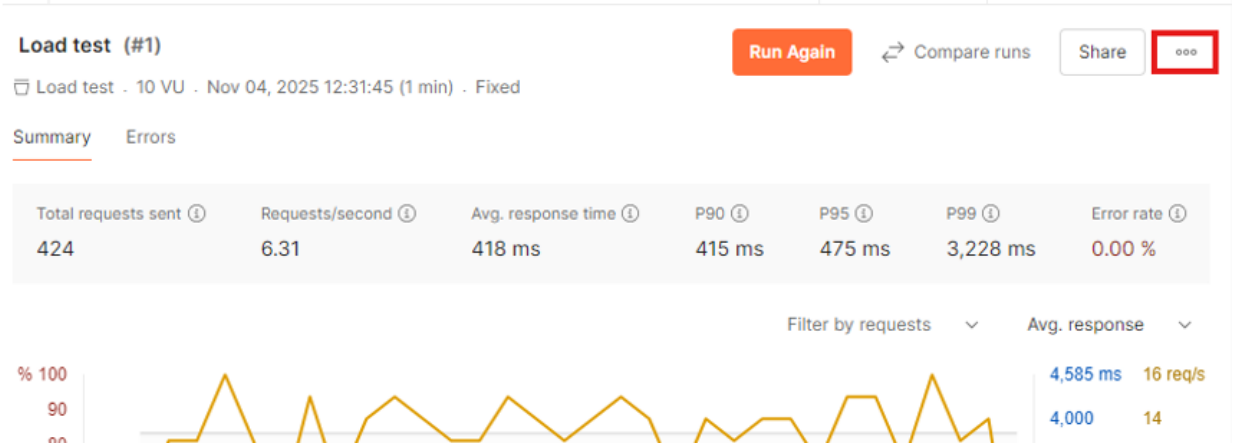




Load Test Output for 10VU over 1min

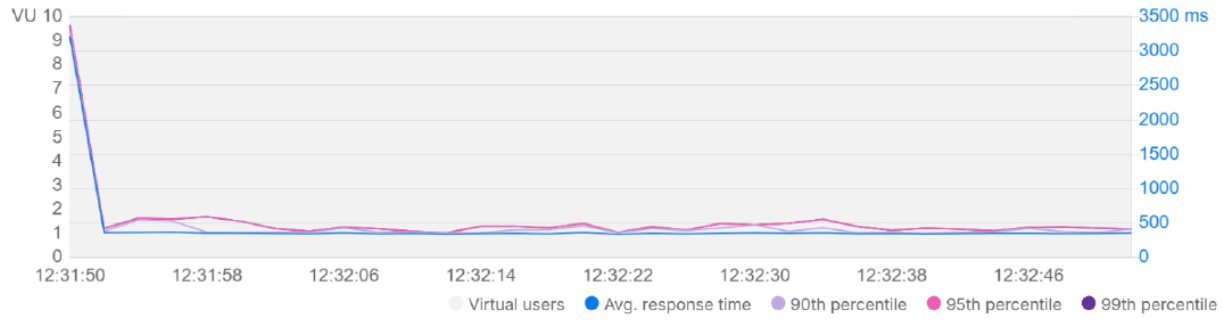


Click here to save to pdf



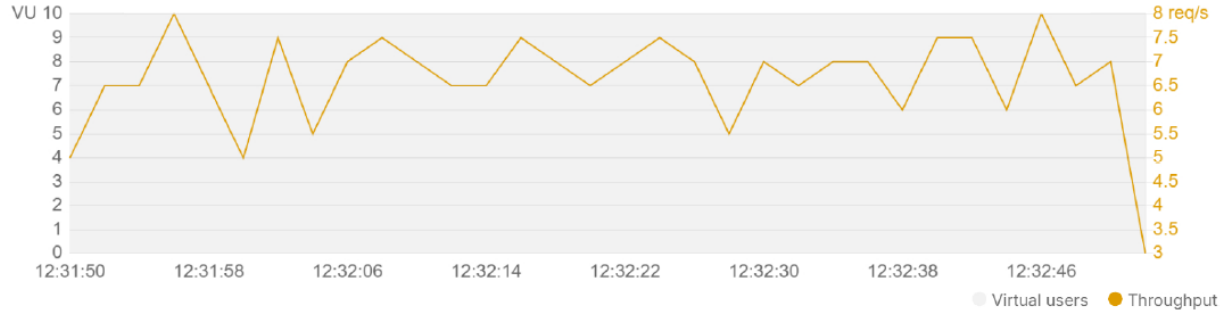
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
POST load-request https://[redacted]execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager	418	415	475	3,228	294	3,375

2 Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
POST load-request https://[redacted].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager	424	6.31	294	418	415	3,375	0

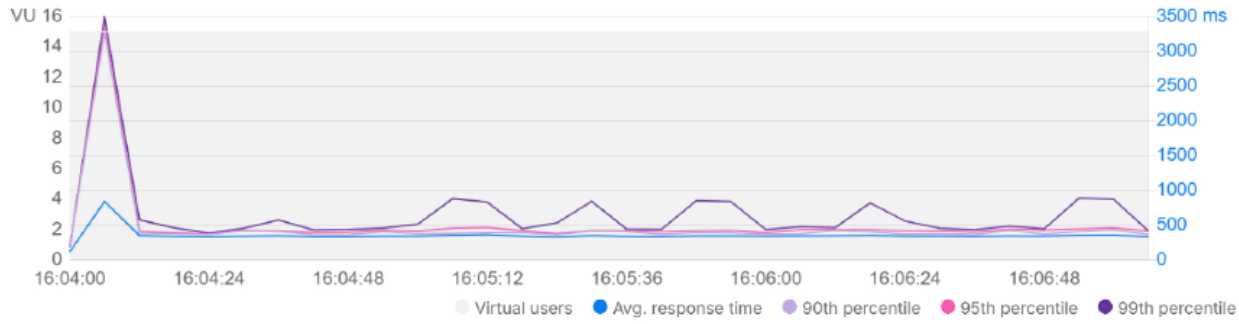
For 15VU over 3min of load test

1. Summary

Total requests sent	Throughput	Average response time	Error rate
1,920	10.30 requests/second	362 ms	0.78 %

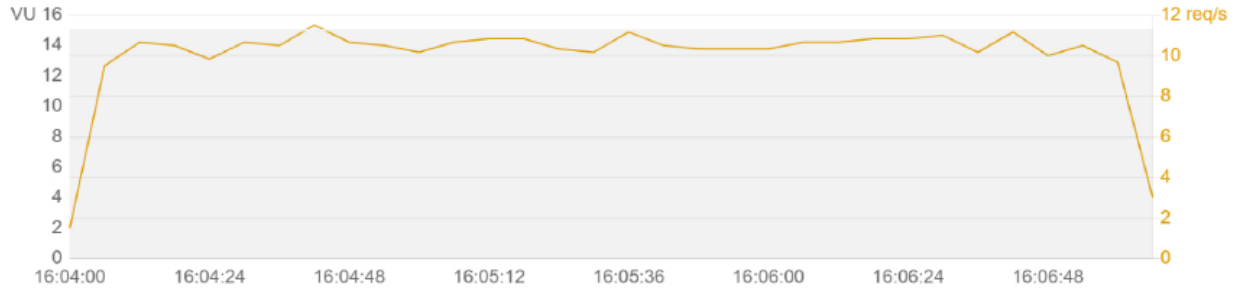
1.1 Response time

Response time trends during the test duration.



1.2 Throughput

Rate of requests sent per second during the test duration.



1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
POST New Request https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager	362	395	434	825	54	3,488

1.4 Requests with most errors

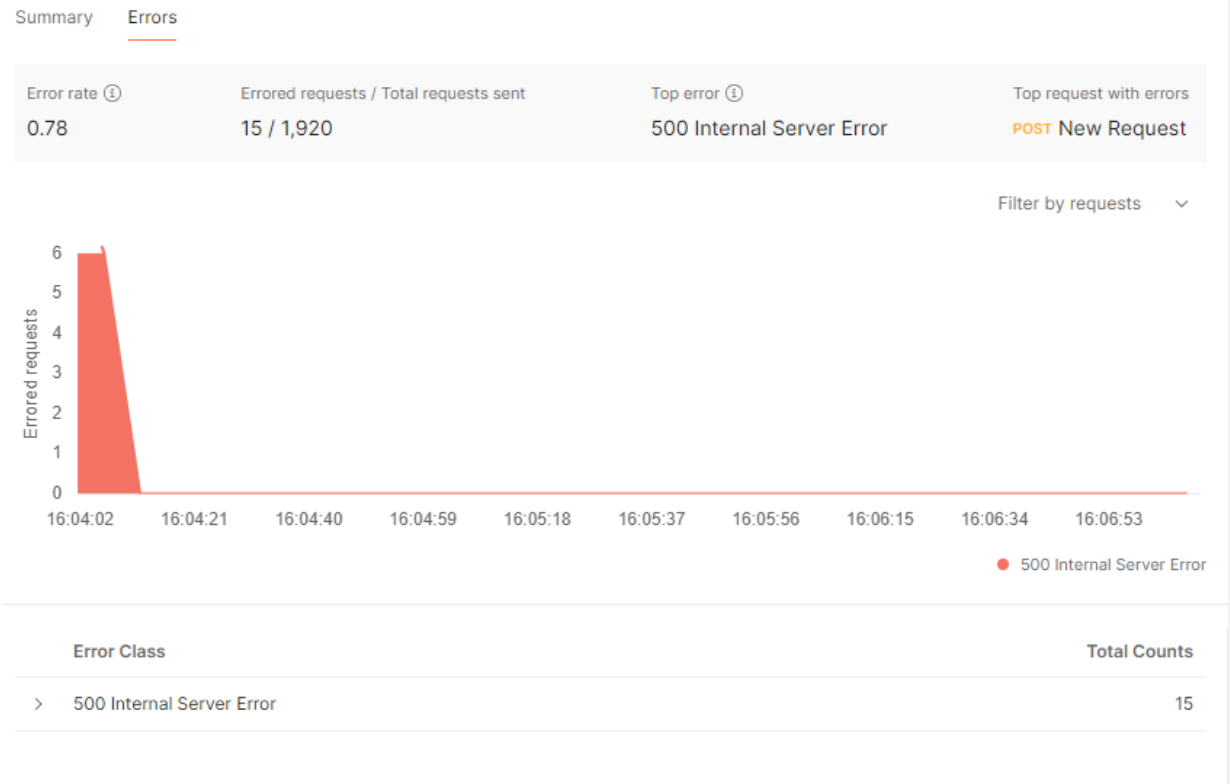
Top 5 requests with the most errors, along with the most frequently occurring errors for each request.

Request	Total error count	Error 1	Error 2	Other errors
POST New Request https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager	15	500 Internal Server Error (15)	-	0

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
POST New Request	1,920	10.30	54	362	395	3,488	0.78



KEY FINDINGS & RECOMMENDATIONS

Following are the synthesized results from the AWS Lambda Power Tuning framework and the Postman Load Testing simulation for the ***lambda-over-https*** function, providing actionable insights for optimizing performance, cost, and reliability.

I tested four memory configurations: **128 MB, 256 MB, 512 MB, and 1024 MB**, using the cost strategy with parallel invocations. Here's what the results reveal:

Performance Trends

- **Execution Time** decreased significantly as memory increased:
  - 128 MB showed the longest duration and highest latency.
  - 512 MB and 1024 MB offered much faster execution, with diminishing returns beyond 512 MB.
- **Cost Efficiency** peaked around **256 MB to 512 MB**, where execution time dropped without a steep cost increase.
- **Cold Start Impact** was more noticeable at lower memory settings, especially 128 MB, which had slower initialization.

Optimal Memory Setting

- For CRUD operations on DynamoDB with moderate payloads, **512 MB** strikes the best balance between speed and cost.
- **256 MB** may be acceptable for low-throughput or latency-tolerant workloads.

Key Findings from Postman Load Testing

Two load profiles were executed-

- 10 Virtual Users for 1 Minute
- 15 Virtual Users for 3 Minutes

Observations

- Average Response Time was stable under 10 VU but degraded slightly at 15 VU.
- Latency Percentiles (P90, P95, P99) showed spikes, likely due to cold starts or backend throttling.
- Throughput was consistent but not maximized, indicating room for concurrency tuning.
- Error Rate reached a critical 78% under 15 VU for 3 minutes.

This high failure rate signals serious performance bottlenecks under moderate concurrency and likely Causes:

- Lambda timeouts or throttling due to insufficient memory or concurrency limits
- API Gateway rate limiting or misconfigured integration responses
- DynamoDB throughput saturation, especially if using provisioned capacity
- Uncaught exceptions or malformed payloads under stress

To mitigate 78% error rate following targeted optimizations are recommended

- Enable Provisioned Concurrency to eliminate cold starts for predictable workloads.
- Increase Lambda memory to 512–1024 MB for better throughput and lower latency.
- Implement retry logic and structured error handling to reduce client-side failures.
- Switch DynamoDB to on-demand mode or increase provisioned throughput for write-heavy operations.
- Gradually ramp up virtual users in future load tests to identify failure thresholds.
- Monitor CloudWatch metrics for Lambda duration, error count, and DynamoDB throttling to pinpoint bottlenecks.

Use Case Recommendations

Use Case	Memory Setting	Concurrency Strategy	Rationale
Low-volume Internal API	256mb	Default (1000)	Cost-efficient, tolerates latency
Moderate Traffic	512mb	Provisioned concurrency (5–10)	Reduces cold starts, balances cost
High-throughput ingestion	1024mb	Reserved concurrency (50+)	Prioritize speed, scale aggressively
Latency-sensitive workloads	512mb-1024mb	Warm-up strategy + provisioned	Pre-warm Lambdas to avoid spikes