

Lambda Performance Testing Strategy

Prepared by
Biswajit Dhar



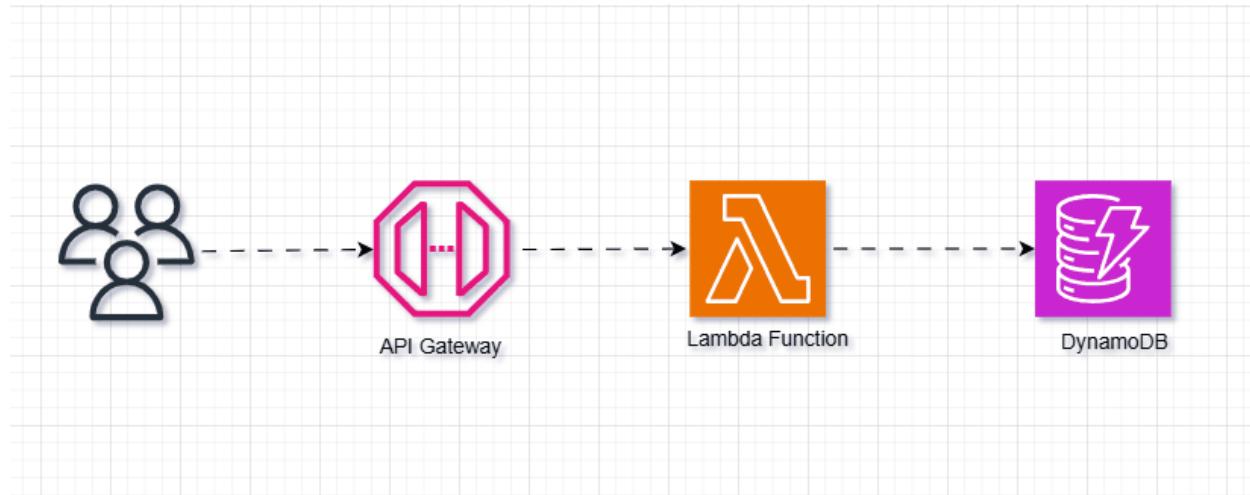
INTRODUCTION

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

The screenshot shows the AWS IAM 'Create policy' wizard. The 'Specify permissions' step is selected. The 'Policy editor' section displays the following JSON code:

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Sid": "Stmt1428341300017",
6        "Action": [
7          "dynamodb:DeleteItem",
8          "dynamodb:GetItem",
9          "dynamodb:PutItem",
10         "dynamodb:Query",
11         "dynamodb:Scan",
12         "dynamodb:UpdateItem"
13       ],
14       "Effect": "Allow",
15       "Resource": "*"
16     },
17     {
18       "Sid": "",
19       "Resource": "*",
20       "Action": [
21         "logs:CreateLogGroup",
22         "logs:CreateLogStream",
23         "logs:PutLogEvents"
24       ],
25       "Effect": "Allow"
26     }
27   ]
28 }
```

Below the editor is a button labeled '+ Add new statement'. To the right of the editor, there are tabs for 'Visual' and 'JSON' (which is highlighted with a red box). Further right are buttons for 'Edit statement', 'Select an existing policy', and a '+' icon.

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"  
    }  
}  
]
```

lambda-dynamodb-cloudwatch-policy Info

Policy details

Type Customer managed	Creation time November 04, 2025, 09:33 (UTC-06:00)	Edited time November 04, 2025, 09:33 (UTC-06:00)
--------------------------	---	---

Permissions Entities attached Tags Policy versions (1) Last Accessed

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Search

Allow (2 of 450 services)

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	All resources	None
DynamoDB	Limited: Read, Write	All resources	None

Step 1 **Select trusted entity**

Step 2 Add permissions

Step 3 Name, review, and create

Select trusted entity Info

Trusted entity type

AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
 Lambda

Choose a use case for the specified service.

Use case

Lambda
Allows Lambda functions to call AWS services on your behalf.

Add permissions

Permissions policies (1/1079)

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

Filter by Type All types

Policy name ▾

	Type
<input type="checkbox"/>	AWS managed
<input checked="" type="checkbox"/> lambda-dynamodb-cloudwatch-policy	Customer managed

▶ Set permissions boundary - optional

Role name
Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and '+-_.' characters.

Description
Add a short explanation for this role.

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

```
1  [
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "sts:AssumeRole"
8              ],
9              "Principal": {
10                  "Service": [
11                      "lambda.amazonaws.com"
12                  ]
13              }
14          }
15      ]
16 ]
```

Step 2: Add permissions

Permissions policy summary

Policy name ▾

	Type
lambda-dynamodb-cloudwatch-policy	Customer managed

Build the Lambda function to handle CRUD operations

Lambda > Functions > Create function

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The 'Basic information' section includes:

- Function name:** lambda-over-https
- Runtime:** Python 3.13
- Architecture:** x86_64
- Permissions:** A dropdown menu is open, showing "Change default execution role". It includes options for creating a new role, using an existing one, or creating from AWS policy templates. "Use an existing role" is selected, and "lambda-apig-role" is chosen.

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))')

```

The screenshot shows the AWS Lambda function configuration interface. The 'Test' tab is highlighted with a red box. The code source editor displays the Python function `lambda_function.py` with the provided code. The left sidebar shows the project structure under 'EXPLORER' with 'LAMBDA-OVER-HTTPS' expanded, showing 'lambda_function.py'. The bottom right has deployment buttons: 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)'.

Add following json code to test:

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

Test event Info

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

```

1 [object Object]
2   {
3     "operation": "echo",
4     "payload": {
5       "somekey1": "somevalue1",
6       "somekey2": "somevalue2"
7     }
8   }
9 
```

[Format JSON](#)

Executing function: succeeded [logs](#)

Details

```

{ "somekey1": "somevalue1",
  "somekey2": "somevalue2"
} 
```

Summary

Code SHA-256	Execution time
HAPq9EReJVECSgLavtc/gyd5vZtd9eiUGF932t0jBxY=	8 minutes ago
Function version	Request ID
\$LATEST	1122eb70-e7b1-44e6-a5fb-31951eb00e9d
Duration	Billed duration
1.82 ms	273 ms
Resources configured	Max memory used
128 MB	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 Duration: 1.82 ms Billed Duration: 273 ms Memory Size: 128 MB Max Memory Used: 65 MB Init Duration: 270.70 ms
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

```


Create REST API Info

API details

New API
Create a new REST API.

Import API
Import an API from an OpenAPI definition.

Clone existing API
Create a copy of an API in this AWS account.

Example API
Learn about API Gateway with an example API.

API name

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

Resources

/

Resource details

Path
/

Methods (0)

Method type	Integration type

Deploy API

API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj)

Resources

Create resource

ARN: arn:aws:execute-api:us-east-1:1381244258876:b3r4tw7ubj/*POST/DynamoDBManager

Resource ID: brwtkm

Method execution: /DynamoDBManager - POST

Client → Method request → Integration request → Lambda integration
← Method response ← Integration response

API actions: Update documentation, Delete, Deploy API

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 > DynamoDBOpsAPI (b3r4tw7ubj) > Stages

Stages

+ Prod

Stage details

Stage name: Prod

Cache cluster: Inactive

Default method-level caching: Inactive

Rate Info: 10000

Burst Info: 5000

Invoke URL: https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod

Active deployment: Started on November 04, 2025, 10:41 (UTC-06:00)

The screenshot shows the AWS API Gateway 'Stages' page. On the left, there's a sidebar with 'APIs', 'Custom domain names', 'Domain name access associations', and 'VPC links'. Below that is a section for 'API: DynamoDBOpsAPI' with 'Resources', 'Stages' (which is selected), 'Authorizers', and 'Gateway responses'. The main area is titled 'Stages' and shows a tree structure: Prod > / > /DynamoDBManager > POST. The 'Invoke URL' field is highlighted with a red box and contains the URL [https://\[REDACTED\].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager](https://[REDACTED].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager).

Testing Integration Using Postman and Command-line Tools

Open Postman and click on +

The screenshot shows the Postman interface. At the top, it says 'Biswajit Dhar's Workspace'. Below that, there are buttons for 'New', 'Import', 'Overview', and a search bar. A request is being set up with the method 'POST' and the URL [https://\[REDACTED\].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager](https://[REDACTED].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager). The 'Body' tab is selected, and the 'raw' radio button is checked. The raw JSON payload is shown in a code editor:

This will make API call using API Gateway endpoint, [https://\[REDACTED\].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager](https://[REDACTED].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.

The screenshot shows the Postman interface with a request setup. The method is 'POST' and the URL is [https://\[REDACTED\].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager](https://[REDACTED].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager). The 'Body' tab is selected, and the 'raw' radio button is checked. The raw JSON payload is:

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

Add following json content for Raw:

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

The screenshot shows the Postman interface with a POST request to `https://[REDACTED].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager`. The request body contains JSON data for creating a new item in a table named `lambda-apig`.

```

1  {
2      "operation": "create",
3      "tableName": "lambda-apig",
4      "payload": {
5          "Item": {
6              "id": "1234ABCD",
7              "number": 5
8          }
9      }

```

The response status is 200 OK, with a response time of 3.40 s and a body size of 751 B. The response JSON includes metadata like RequestId, HttpStatusCode, and HTTPHeaders.

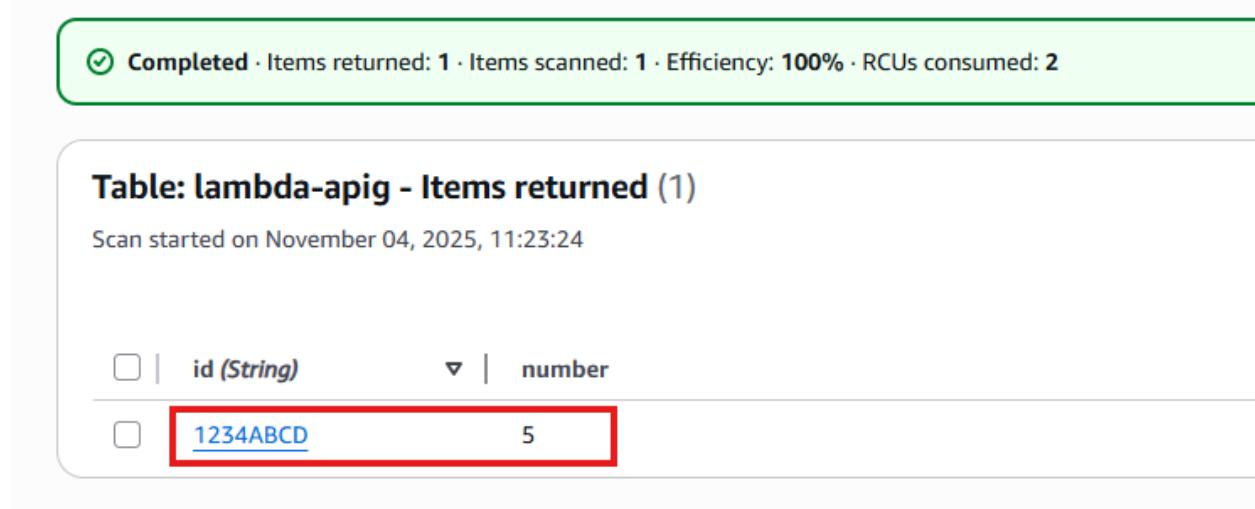
Alternatively, you can run this from command line

```
$ 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

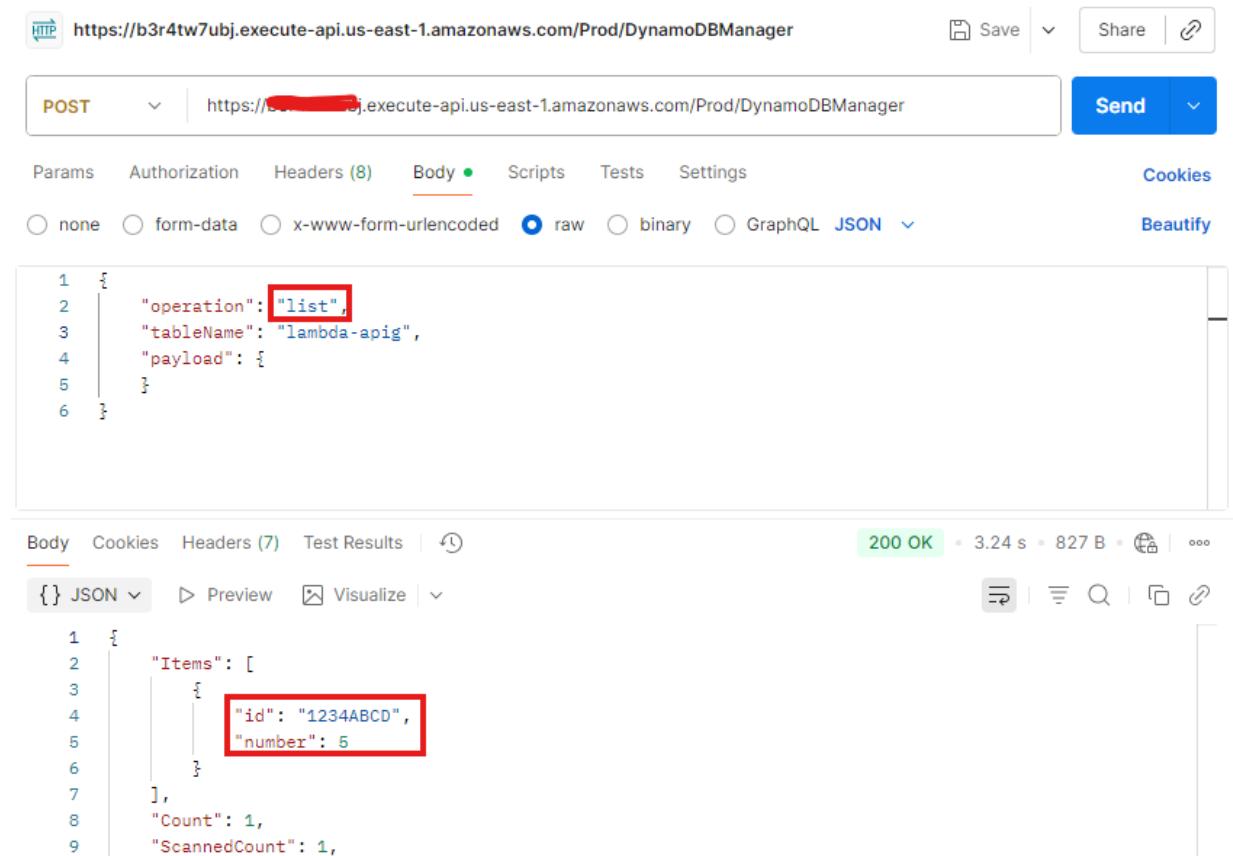
The screenshot shows the AWS DynamoDB console. On the left, the navigation bar has 'Tables' selected. In the main pane, the table `lambda-apig` is listed under 'Tables (1)'. On the right, the table's configuration page is shown. A yellow box highlights the 'Actions' dropdown and the 'Explore table items' button. Another yellow box highlights the 'Get live item count' button at the bottom right of the table configuration section.

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



The screenshot shows the AWS DynamoDB console. At the top, a green bar indicates "Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 2". Below this, a table titled "Table: lambda-apig - Items returned (1)" is shown. A note says "Scan started on November 04, 2025, 11:23:24". The table has two columns: "id (String)" and "number". There is one item listed: "1234ABCD" under "id" and "5" under "number".

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



The screenshot shows the Postman interface. The URL is `https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager`. The method is POST. The request body is:

```
1 {
2     "operation": "list",
3     "tableName": "lambda-apig",
4     "payload": {}
5 }
```

The response status is 200 OK, with a response time of 3.24 s and a response size of 827 B. The JSON response is:

```
1 {
2     "Items": [
3         {
4             "id": "1234ABCD",
5             "number": 5
6         }
7     ],
8     "Count": 1,
9     "ScannedCount": 1,
```

LAMBDA POWER TUNING

The screenshot shows the AWS Serverless Application Repository interface. At the top, there's a search bar with the text 'serverless'. Below it, a banner for 'Serverless Application Repository' says 'Assemble, deploy, and share serverless applications within teams or publicly'. The main area shows a list of 'Available applications' under the 'Public applications' tab. One application, 'aws-lambda-power-tuning', is highlighted with a red box. The application card for 'aws-lambda-power-tuning' includes a description, author information (Alex Casalboni), deployment count (25.9K), and tags like lambda, state-machine, optimization, step-functions, power, elastic, log-ingestion, s3, analytics, monitoring, and logs. Below this, there are cards for 'elastic-log-ingestion-control-tower', 'daily-recycle', 'stemplayer-js-api', 'batch-processing-stepFunction', 'Voice-Lexicon-API', and 'helper-elastic-log-ingestion'. The navigation bar at the bottom shows 'Lambda > Applications > Review, configure and deploy', with 'Review, configure and deploy' also highlighted with a red box.

aws-lambda-power-tuning — version 4.3.7

Application details

Author	Source code URL
Alex Casalboni	https://github.com/alexcasalboni/aws-lambda-power-tuning

Template

Permissions

License

Readme file

[View on the AWS Serverless Application Repository site](#)

Step Functions < State machines

State machines (1)

Search for state machines | Any type

Name	Type	Creation date
powerTuningStateMachine-a8d93e30-b9a4-11f0-bb32-0eda3b51d88f	Standard	Nov 4, 2025, 11:36:04 (UTC-06:00)

powerTuningStateMachine-a8d93e30-b9a4-11f0-bb32-0eda3b51d88f

Details

Arn: arn:aws:states:us-east-1:381244258876:stateMachine:powerTuningStateMachine-a8d93e30-b9a4-11f0-bb32-0eda3b51d88f

IAM role ARN: arn:iam:iam::381244258876:role/serverlessrepo-aws-lambda-power-tu-statemachineRole-QQRSpJPLTK6

Type: Standard
Status: Active
Creation date: Nov 4, 2025, 11:36:04 (UTC-06:00)
X-Ray tracing: Disabled

Executions | Monitoring | Logging | Definition | Aliases | Versions | Tags

Executions (0/0)

No executions

Start execution

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

```
1 {  
2   "lambdaARN": "arn:aws:lambda:us-east-1:[REDACTED]:function:lambda-over-https",  
3   "powerValues": [  
4     128,  
5     256,  
6     512,  
7     1024  
8   ],  
9   "num": 10,  
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

```
8 ],  
9   "num": 10,  
10  "payload": {  
11    "operation": "list",  
12    "tableName": "[REDACTED]",  
13    "payload": {}  
14  },  
15  "parallelInvocation": true,  
16  "strategy": "cost"  
17 }
```


This collection is empty.
Add a request to start working.

Load test

Make things easier for your teammates with a complete collection description.

POST https://[REDACTED].execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager

```

1 {
2   "operation": "list",
3   "tableName": "lambda-apig",
4   "payload": {}
5 }
6
7
  
```

Click on three dots here and Run

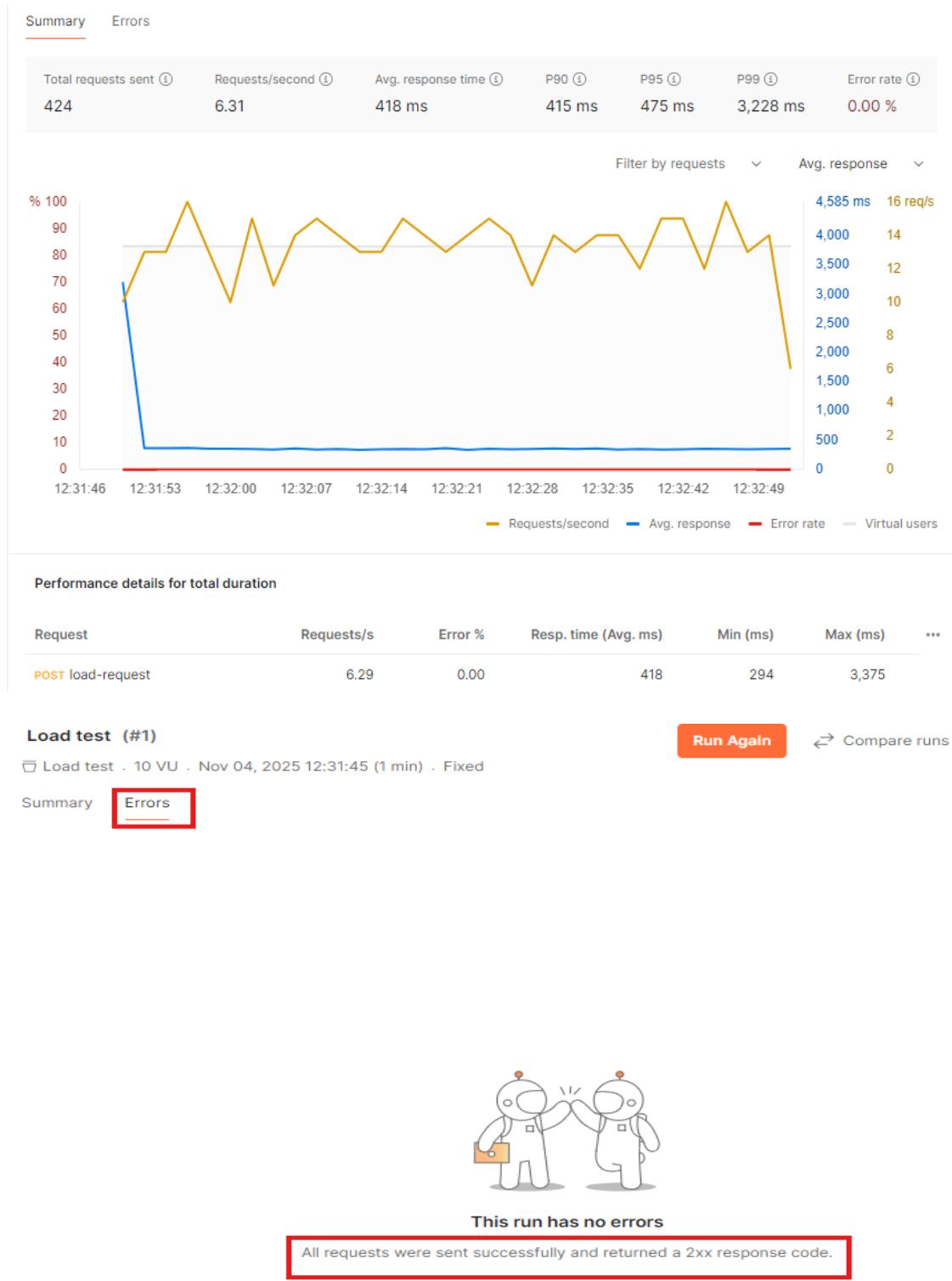
POST https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager

```

1 {
2   "operation": "list",
3   "tableName": "lambda-apig",
4   "payload": {}
5 }
6
7
  
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (bd-collection, bd-request), 'Environments' (with a 'Load test' entry highlighted with a red box), 'Flows', and 'History'. The main workspace has tabs at the top: 'Ovi' (disabled), 'GET https' (disabled), 'POST New', 'bd-re...', 'POST http...', 'Load i...', 'POST load...', and 'POST load...'. Below these tabs, the 'Performance' tab is selected (highlighted with a red box). A 'Run Sequence' section shows a single step: '1 POST load-request'. To the right, a large box titled 'Test how your APIs perform under load' contains instructions about performance testing. Below this is a 'Set up your performance test' section with fields for 'Load profile' (set to 'Fixed'), 'Virtual users' (set to '10', highlighted with a red box), and 'Test duration' (set to '1 mins'). A preview area shows '10 VUs' and a timeline from '0' to '1 min'. A note below states: '10 virtual users run for 1 minute, each executing all requests sequentially.' Further down, there's a 'Data file' section with a 'Select file' button, and a 'Pass test if' section with a 'Run' button (highlighted with a red box).

Load Test Output for 10VU over 1min



Click here to save to pdf



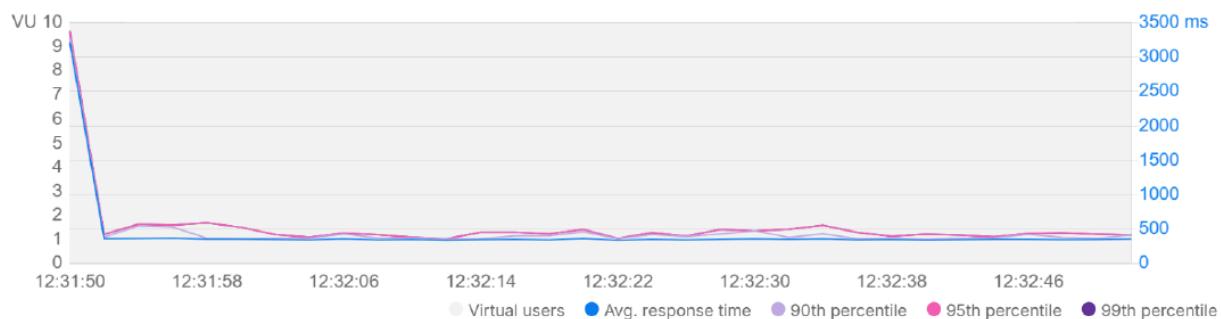
Summary Errors

Total requests sent ⓘ	Requests/second ⓘ	Avg. response time ⓘ	P90 ⓘ	P95 ⓘ	P99 ⓘ	Error rate ⓘ
424	6.31	418 ms	415 ms	475 ms	3,228 ms	0.00 %



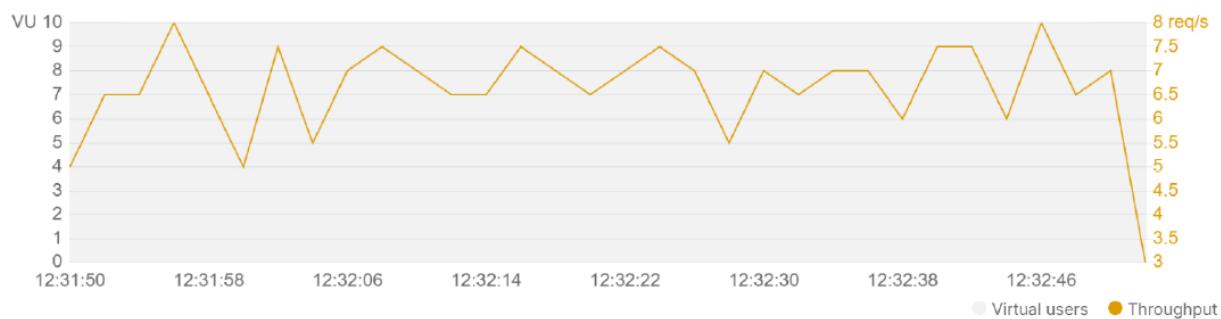
1.1 Response time

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://████████.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://████████.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager	424	6.31	294	418	415	3,375	0

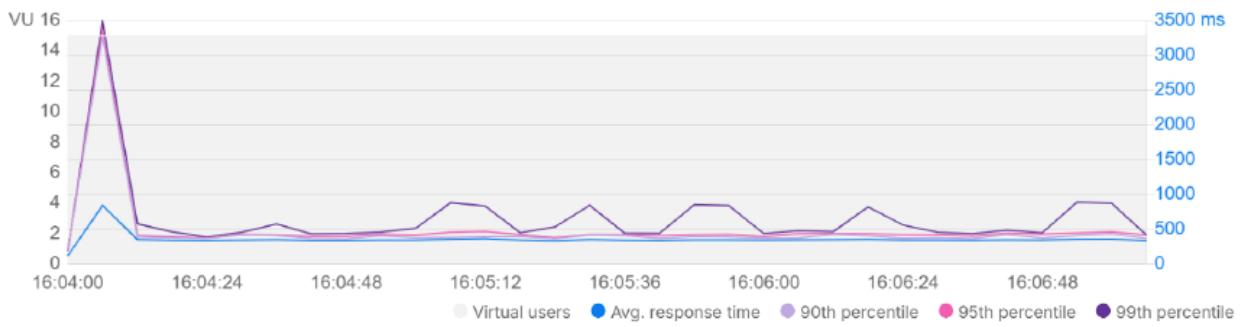
For 15VU over 3min of load test

1. Summary

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

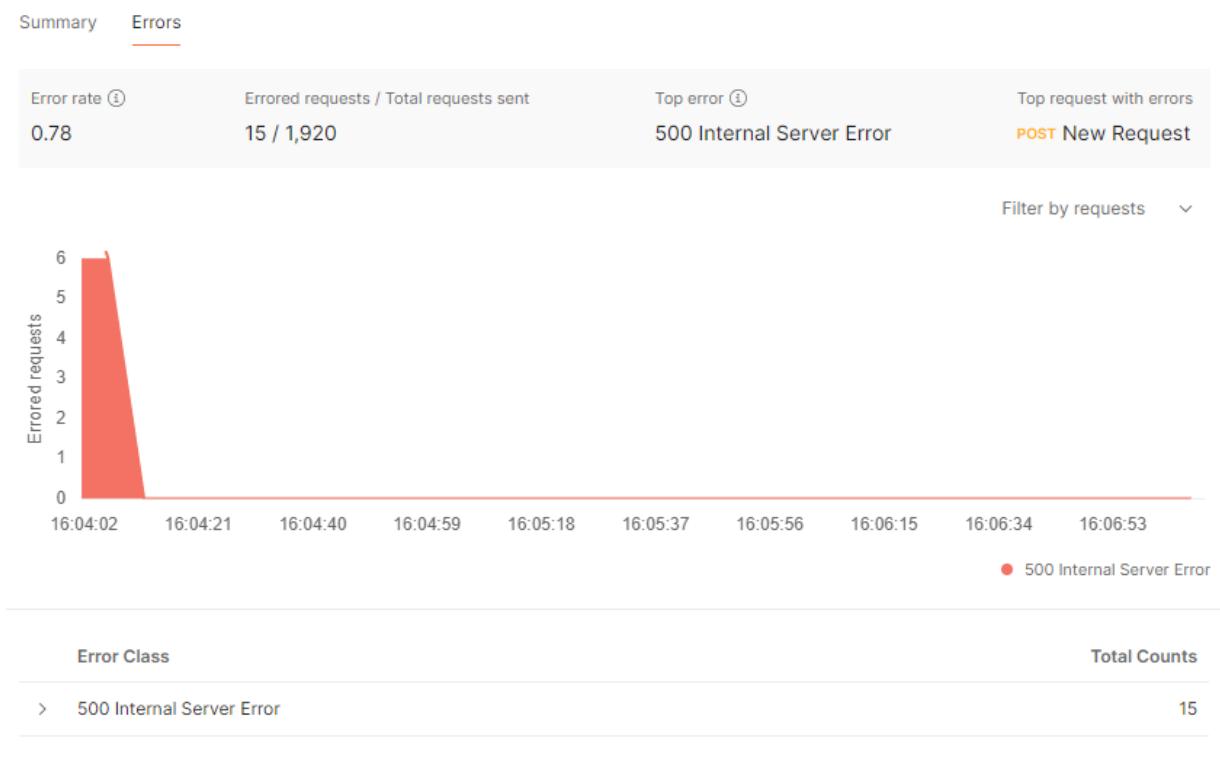
Top 5 requests with the most errors, along with the most frequently occurring errors for each request.

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