

Lambda Performance Testing Strategy

Prepared by
Biswajit Dhar



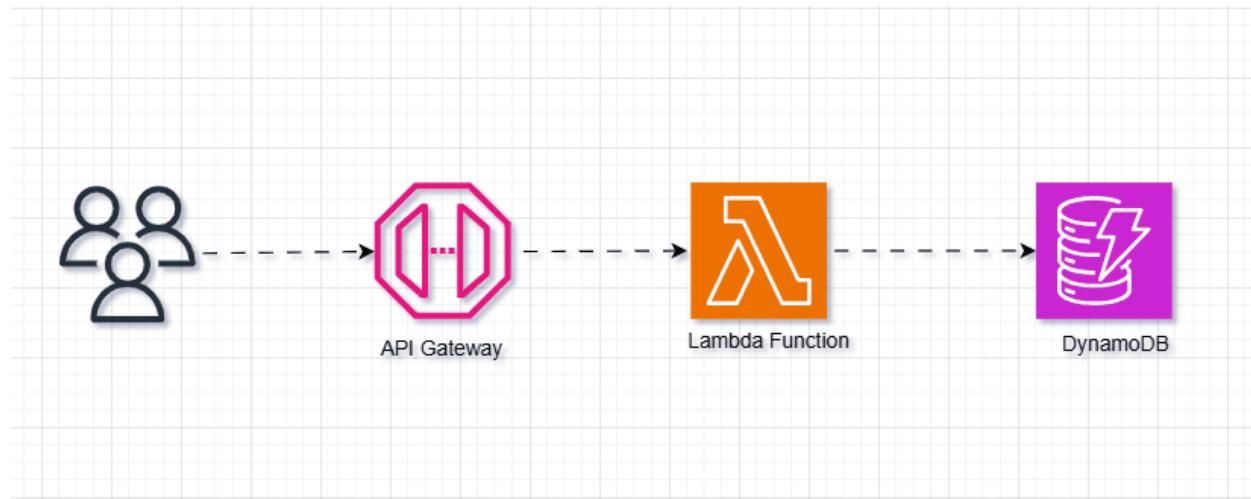
INTRODUCTION

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', with 'JSON' being the active tab. A sidebar on the right provides options for selecting existing policies or creating a new one.

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. | | | | | | | | | | | | | | | |
| <input type="text"/> Search | | | | | | | | | | | | | | | |
| Allow (2 of 450 services) <table border="1"> <thead> <tr> <th>Service</th> <th>Access level</th> <th>Resource</th> <th>Request condition</th> </tr> </thead> <tbody> <tr> <td>CloudWatch Logs</td> <td>Limited: Write</td> <td>All resources</td> <td>None</td> </tr> <tr> <td>DynamoDB</td> <td>Limited: Read, Write</td> <td>All resources</td> <td>None</td> </tr> </tbody> </table> | | | | Service | Access level | Resource | Request condition | CloudWatch Logs | Limited: Write | All resources | None | DynamoDB | Limited: Read, Write | All resources | None |
| 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 <input checked="" type="radio"/> AWS service <input type="radio"/> AWS account <input type="radio"/> SAML 2.0 federation <input type="radio"/> Custom trust policy | | | | | | | | | | | | | | | |
| Select trusted entity Info | | | | | | | | | | | | | | | |
| Trusted entity type <ul style="list-style-type: none"> <input checked="" type="radio"/> AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account. <input type="radio"/> AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account. <input type="radio"/> SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account. <input type="radio"/> 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 <input type="text"/> Lambda | | | | | | | | | | | | | | | |
| Choose a use case for the specified service. Use case <ul style="list-style-type: none"> <input checked="" type="radio"/> Lambda Allows Lambda functions to call AWS services on your behalf. | | | | | | | | | | | | | | | |

Add permissions Info

Permissions policies (1/1079) Info

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

Filter by Type

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

```
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 top navigation bar has tabs for 'Code', 'Test' (which is highlighted with a red box), 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below the tabs, there's a search bar with the placeholder 'lambda-over-https'. The left sidebar has icons for 'EXPLORER', 'LAMBDA-OVER-HTTPS' (which is expanded to show 'lambda_function.py'), and 'DEPLOY' (with 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)' buttons). The main area is titled 'Code source' with an 'Info' link. It shows the Python code for the 'lambda_function.py' file. The code defines a 'lambda_handler' function that prints the received event, extracts the 'operation' key, and then performs the specified operation on a DynamoDB table. A scroll bar is visible on the right side of the code editor.

```

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

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 [
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
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

DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

lambda-apig

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

id

String

1 to 255 characters and case sensitive.

DynamoDB > Tables

The lambda-apig table was created successfully.

| Name | Status | Partition key | Sort key | Indexes | Replication Regions | Deletion protection | Favorite | Read capacity mode | Write capacity mode | Total size |
|-------------|--------|---------------|----------|---------|---------------------|---------------------|----------|--------------------|---------------------|------------|
| lambda-apig | Active | id (\$) | - | 0 | 0 | Off | ★ | On-demand | On-demand | 0 bytes |

Configure API Gateway

API Gateway > APIs > Create API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:
Lambda, HTTP backends

Import

Build

WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services

Build

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

Import

Build

API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj) > Create resource

Successfully created REST API 'DynamoDBOpsAPI (b3r4tw7ubj)'.

Create resource

Resource details

Proxy resource Info
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example [proxy+].

Resource path **Resource name**

CORS (Cross Origin Resource Sharing) Info
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Create resource Create resource

Create a POST Method for our API

API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj)

Successfully created resource '/DynamoDBManager'

Resources

| Resource details | | Actions |
|------------------|------------------|---|
| Path | /DynamoDBManager | Delete Update documentation Enable CORS |
| | | API actions Deploy API |

Methods (0)

| Method type | Integration type | Authorization | API key |
|---------------------|------------------|---------------|---------|
| No methods | | | |
| No methods defined. | | | |

Create method Create method

API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj) > Create method

Create method

Method details

Method type

POST

Integration type

Lambda function

Integrate your API with a Lambda function.



HTTP

Integrate with an existing HTTP endpoint.



AWS service

Integrate with an AWS Service.



VPC link

Integrate with a resource that isn't accessible over the public internet.



Lambda proxy integration

Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1

arn:aws:lambda:us-east-1:381244258876:function:lambda-over-http

Deploy API

API Gateway > APIs > Resources - DynamoDBOpsAPI (b3r4tw7ubj)

Resources

Create resource / /DynamoDBManager POST

ARN: arm:aws:execute-api:us-east-1:381244258876:b3r4tw7ubj/*POST/DynamoDBManager Resource ID: brwtkm

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

API actions Deploy API Update documentation Delete

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

Deployment description

A new stage will be created with the default settings. Edit your stage settings on the [Stage](#) page.

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 interface. On the left, there's a sidebar with 'APIs' and 'Stages' sections. Under 'Stages', 'Prod' is selected. Below it, there's a tree structure with '/' and '/DynamoDBManager'. Under '/DynamoDBManager', a 'POST' method is highlighted with a red box. To the right, a 'Method overrides' section is shown with a note: 'By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.' A tooltip says 'This method inherits its settings from the 'Prod' stage.' Below this is an 'Invoke URL' field containing a redacted URL.

Testing Integration Using Postman and Command-line Tools

[Open Postman and click on +](#)

The screenshot shows the Postman interface. At the top, there are tabs for Home, Workspaces, and API Network. Below that, a workspace named 'Biswajit Dhar's Workspace' is selected. A 'New' button is highlighted with a red box. The main area shows a request configuration with a 'POST' method and a redacted URL. A 'Send' button is also highlighted with a red box.

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.

The screenshot shows the Postman interface with a request configuration. The method is 'POST' and the URL is a redacted version of the API endpoint. The 'Body' tab is selected, and the 'raw' option is chosen. A red box highlights the 'raw' button. Below it, a red box highlights the JSON payload area, which contains the following code:

```

1 "operation": "create",
2   "tableName": "lambda-apig",
3   "payload": {
4     "Item": {
5       "id": "1234ABCD",
6       "number": 5
7     }
8   }
9 }
10
11
12
13
14
15
16
17
18
  
```

Add following json content for Raw:

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


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

✓ Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 2

Table: lambda-apig - Items returned (1)

Scan started on November 04, 2025, 11:23:24

| | id (String) | number |
|--------------------------|--------------------------|--------|
| <input type="checkbox"/> | 1234ABCD | 5 |

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

HTTP <https://b3r4tw7ubj.execute-api.us-east-1.amazonaws.com/Prod/DynamoDBManager> Save Share

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

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

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

Body Cookies Headers (7) Test Results

200 OK 3.24 s 827 B

{ } JSON Preview Visualize

```
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 'serverless' typed in. Below it, a banner for the 'Serverless Application Repository' is visible, stating 'Assemble, deploy, and share serverless applications within teams or publicly'. The main area is titled 'Available applications' under 'Public applications'. A search bar here has 'power' typed in. A checkbox for 'Show apps that create custom IAM roles or resource policies' is checked. The results list several applications, with 'aws-lambda-power-tuning' highlighted by a red box. This application is described as an open-source tool for visualizing and fine-tuning Lambda functions. It has 25.9K deployments and is authored by Alex Casalboni, who is an AWS verified author. Other applications shown include 'elastic-log-ingestion-control-tower', 'daily-recycle', 'stemplayer-js-api', 'batch-processing-stepFunction', 'Voice-Lexicon-API', and 'helper-elastic-log-ingestion'. The 'aws-lambda-power-tuning' page itself has a header 'Review, configure and deploy' with a red box around it. Below this, the application name 'aws-lambda-power-tuning — version 4.3.7' is displayed. The 'Application details' section shows the author as 'Alex Casalboni' (AWS verified author) and the source code URL as <https://github.com/alexcasalboni/aws-lambda-power-tuning>. There are sections for 'Template', 'Permissions', and 'License'. At the bottom, a 'Readme file' section links to the AWS Serverless Application Repository site.

Lambda > Applications > Review, configure and deploy

aws-lambda-power-tuning — version 4.3.7

Review, configure and deploy

Application details

Author
Alex Casalboni
AWS verified author

Source code URL
<https://github.com/alexcasalboni/aws-lambda-power-tuning>

Template

Permissions

License

Readme file

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

visualizationURL
Stats visualization URL
`https://lambda-power-tuning.show/`

▼ powerTuningStateMachine

stateMachineNamePrefix
Prefix to the name of the StateMachine. The StackId will be appended to this value (optional).
`powerTuningStateMachine`

totalExecutionTimeout
Maximum invocation timeout (in seconds) for the Executor step, after which you get a States.Timeout error
`300`

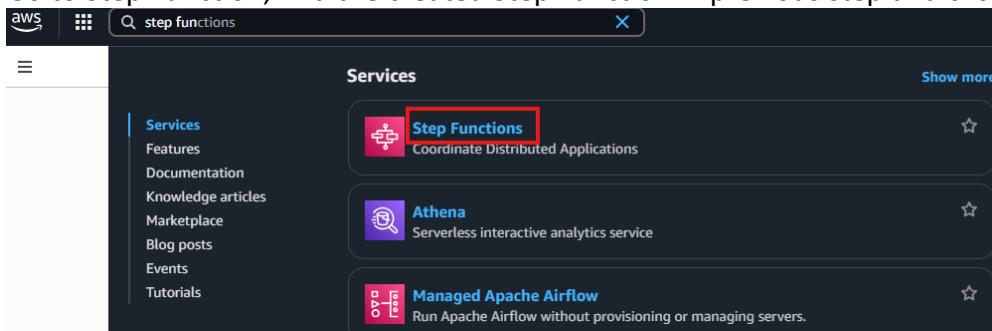
I acknowledge that this app creates custom IAM roles.[Info](#)

Cancel **Deploy**

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-2kgjHUrWLcd | 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 | arm: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-2kgjHUrWLcd | 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



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 }
```

Step Functions > State machines > powerTuningStateMachine-a8d93e30-b9a4-11f0-bb52-0eda5b51d88f > Execution: 1fa967aa-0bbb-4761-81a3-3fa2cc1f686f

Execution started successfully

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

Execution input and output

State input

```

1 *
2   "lambdaARN": "arn:aws:lambda:us-east-1:381244258876:function:lambda-over-https",
3   "powerValues": [
4     128,
5     256,
6     512,
7     1024
8   ],
9   "num": 10,
10  "payload": {
11    "operation": "list",
12    "tableName": "lambda-apig",
13    "payload": {}
14  },
15  "parallelInvocation": true,

```

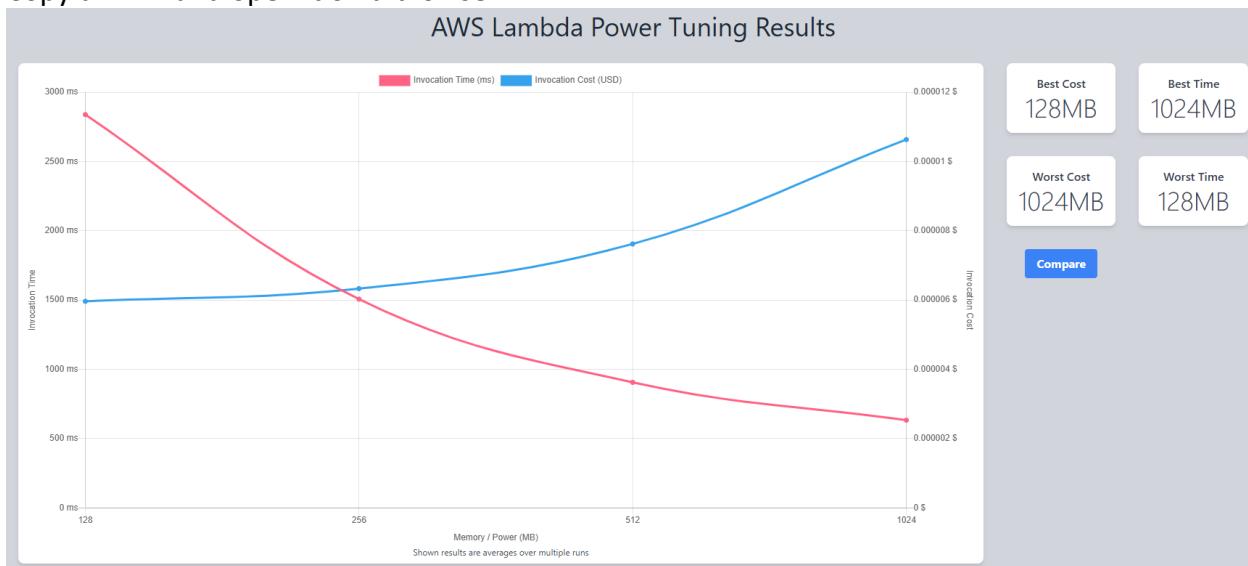
State output

```

1 *
2   {
3     "power": 1024,
4     "cost": 9.744800000000001e-7,
5     "duration": 56.91833333333333,
6     "stateMachine": {
7       "executionCost": 0.00045,
8       "lambdaCost": 0.00015933380000000001,
9       "tuning": {
10         "url": "https://lambda-power-tuning-show/#gAAAAACAAQ=jkbFRLURyEdtV1tdYKxjQg--;uoZelNqM4jVrKfkIIc1CnQ--"
11       }
12     }
13   }
14
15

```

Copy url link and open it on a browser



POSTMAN LOAD TESTING SIMULATION

Create a Blank Collection

Copy and paste api endpoint url

Collections

Search collections

bd-collection

bd-request

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

Params Authorization Headers (8) Body Scripts Tests Settings

Body (raw JSON)

```

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

```

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

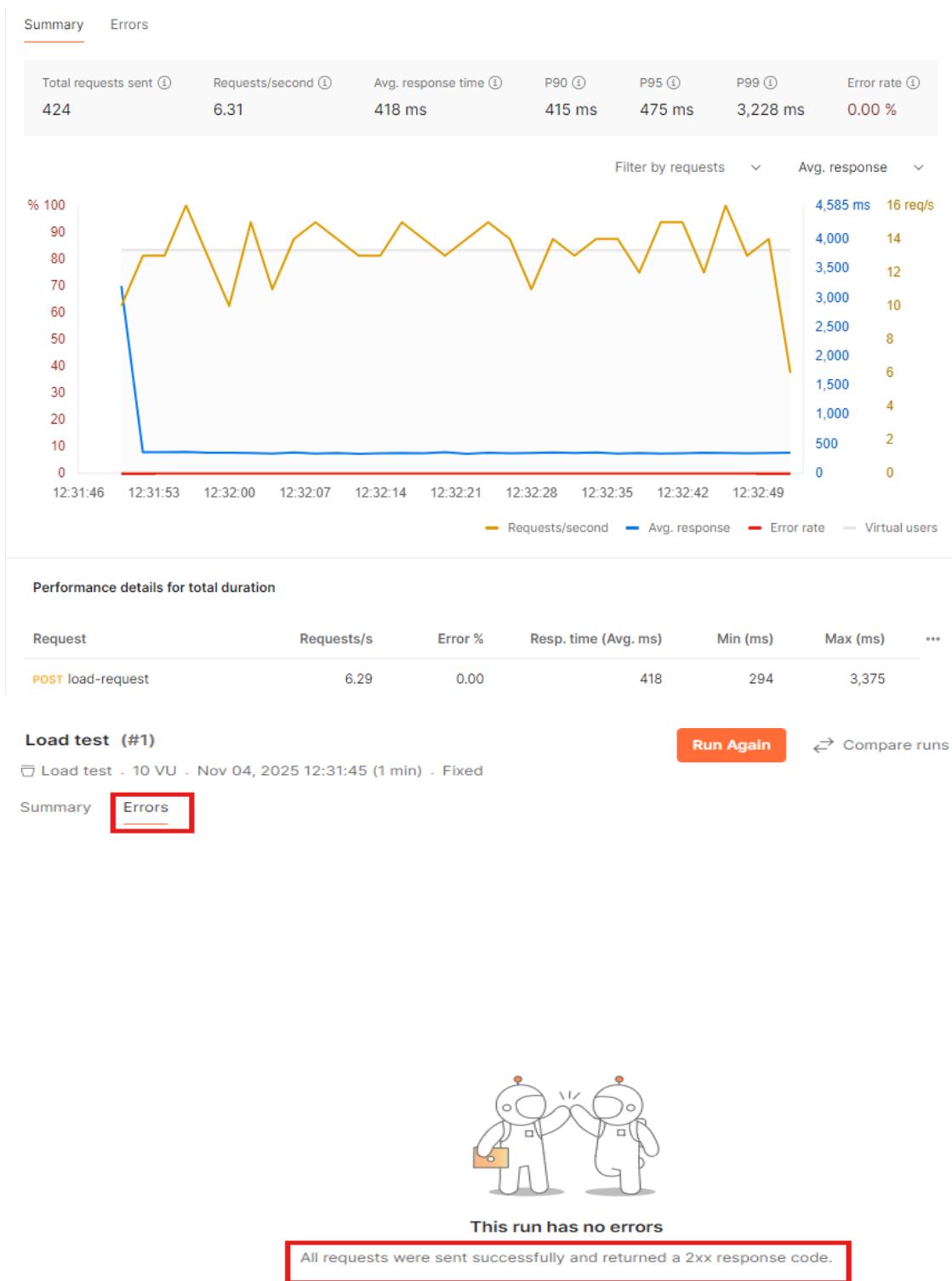
```

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

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, Environments, Flows, History.
- Top Bar:** Biswajit Dhar's Workspace, New, Import, various tabs like Ovi, GET https, POST New, bd-request, POST http, Load, POST load, and a dropdown for No environment.
- Run Sequence:** A list of requests:
 - 1 POST load-request
- Performance Tab:** Selected (highlighted with a red box).
- Description:** "Test how your APIs perform under load. Simulate real-world traffic from your local machine and observe the performance of your APIs. Learn more about [performance testing](#)."
- Setup:**
 - Load profile: Fixed (dropdown), Virtual users: 10 (highlighted with a red box), Test duration: 1 mins.
 - 10 VUs
 - 0 to 1 min timeline.
 - Text: "10 virtual users run for 1 minute, each executing all requests sequentially."
 - Data file: Select file.
 - Pass test if:
 - Run button (highlighted with a red box).

Load Test Output for 10VU over 1min



Click here to save to pdf



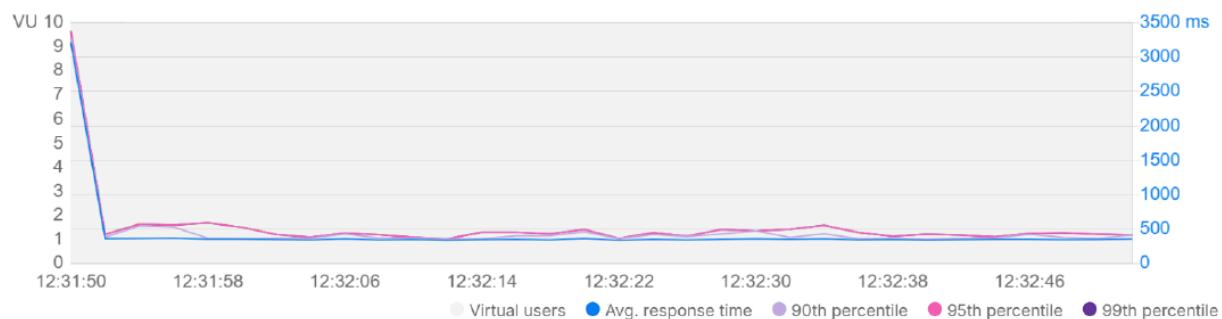
Summary Errors

| 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 % |



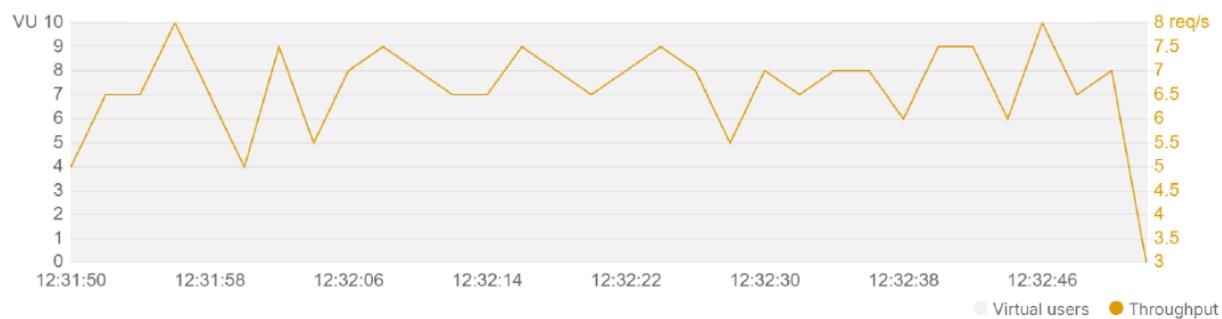
1.1 Response time

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 |

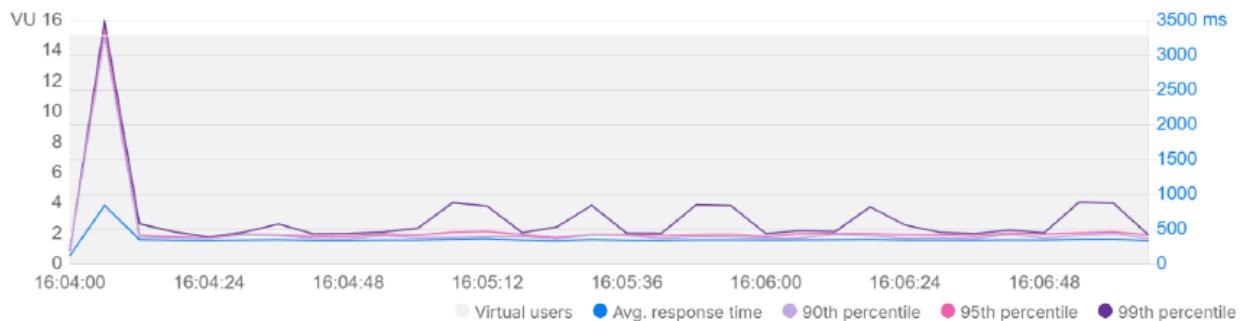
For 15VU over 3min of load test

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

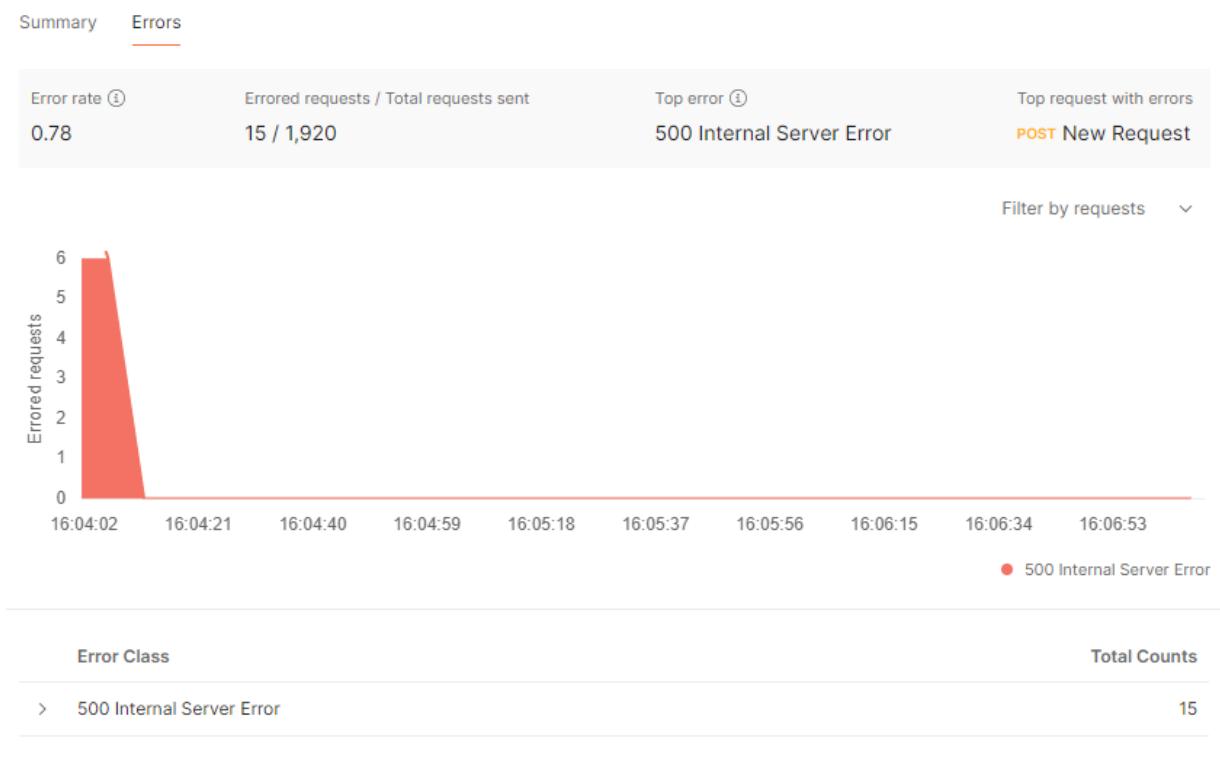
Top 5 requests with the most errors, along with the most frequently occurring errors for each request.

| 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 |