Serverless URL Shortener using AWS Lambda, API Gateway, and DynamoDB

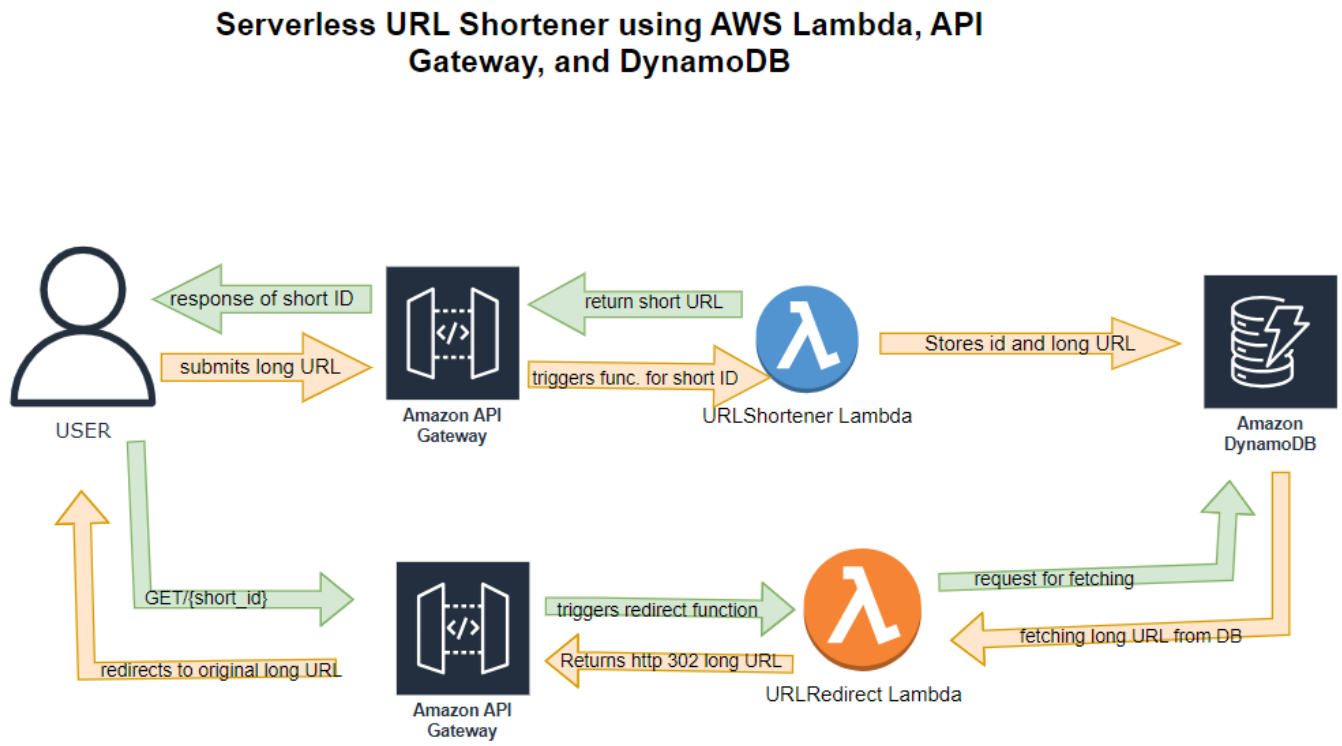
1. Project Overview

Create a serverless URL shortening application where users can:

- Submit a long URL and receive a shortened version.
- Use the short URL to redirect to the original long URL.

2. Architecture and Services Used

- **AWS API Gateway** — to expose RESTful endpoints for URL shortening and redirecting.
- **AWS Lambda** — functions to handle URL shortening logic and URL redirection.
- **DynamoDB** — to store the mapping between short IDs and long URLs.
- **IAM Roles and Policies** — to securely allow Lambda functions access to DynamoDB.
- **CloudWatch Logs** — to monitor and debug Lambda executions.



3. API Design

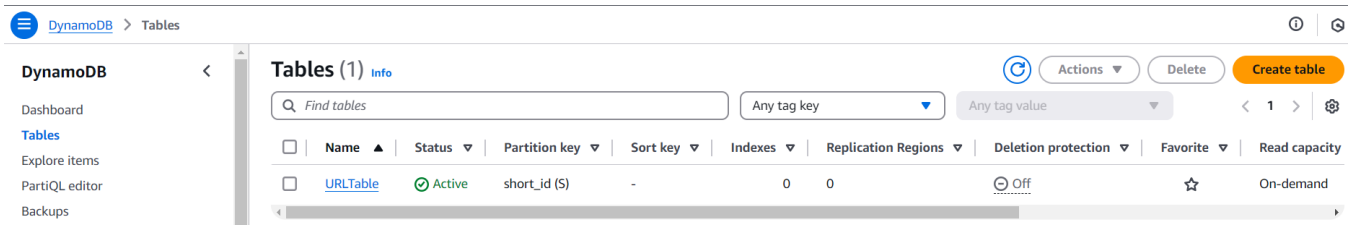
- **POST /shorten**
  - Request: JSON with a long_url field.
  - Response: JSON containing the generated short_url.
- **GET /{short_id}**
  - Request: Short ID path parameter.
  - Response: HTTP 302 redirect to the original long URL.

---

4. Implementation Steps

a. DynamoDB Table Setup

- Created a DynamoDB table named URLMapping.
- Primary key: short_id (string).
- Stores long_url against each short_id.

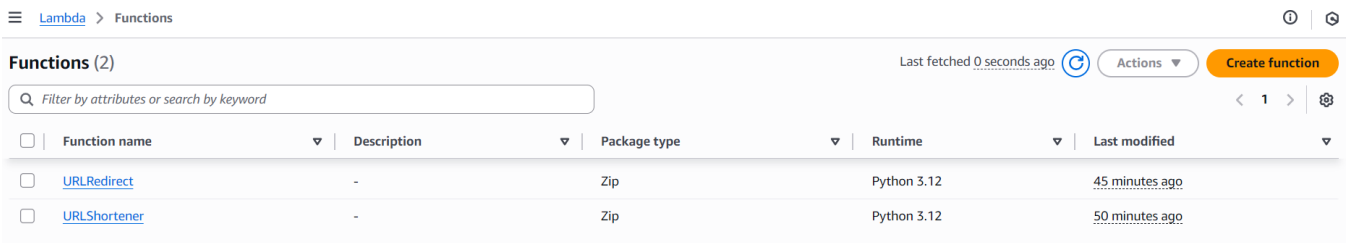


b. Lambda Function: URLShortener

- Receives POST request with long_url.
- Generates a unique short ID.
- Stores mapping (short_id → long_url) in DynamoDB.
- Returns JSON with the full short URL.

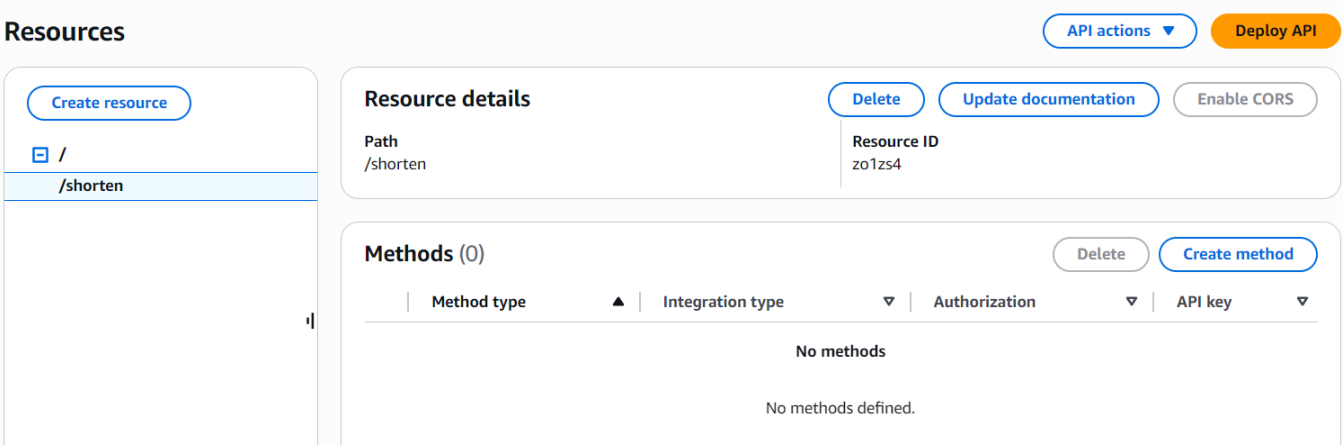
c. Lambda Function: URLRedirect

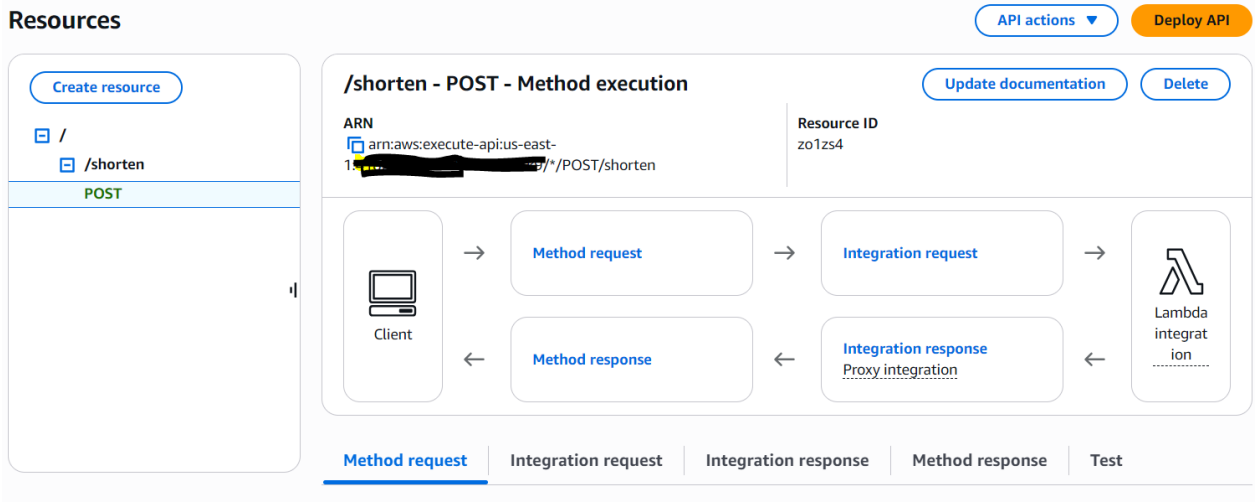
- Receives GET request with short_id.
- Queries DynamoDB for the original URL.
- Returns HTTP 302 redirect with Location header pointing to the long_url.

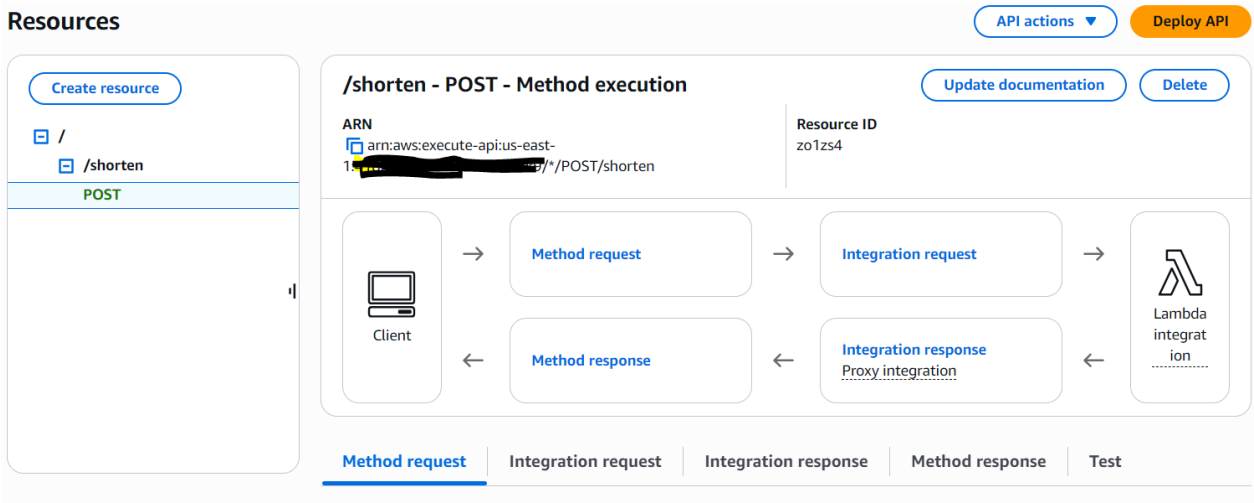


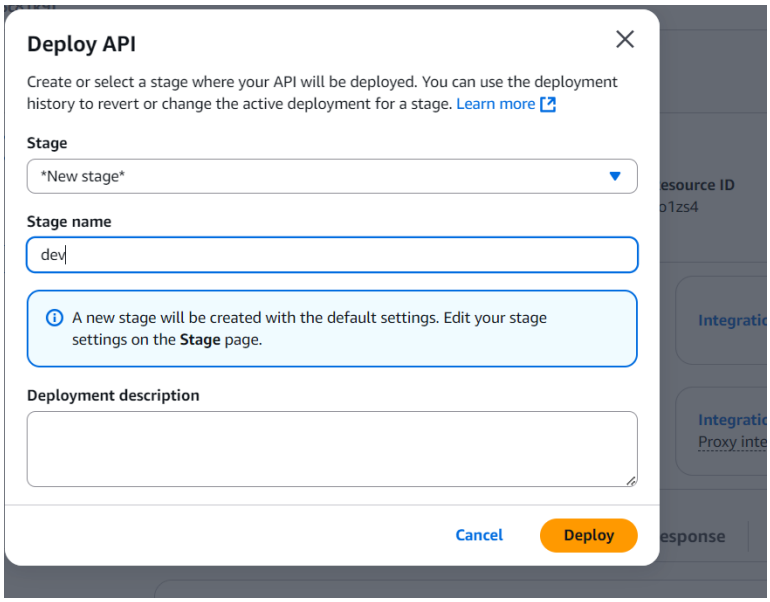
d. API Gateway Configuration

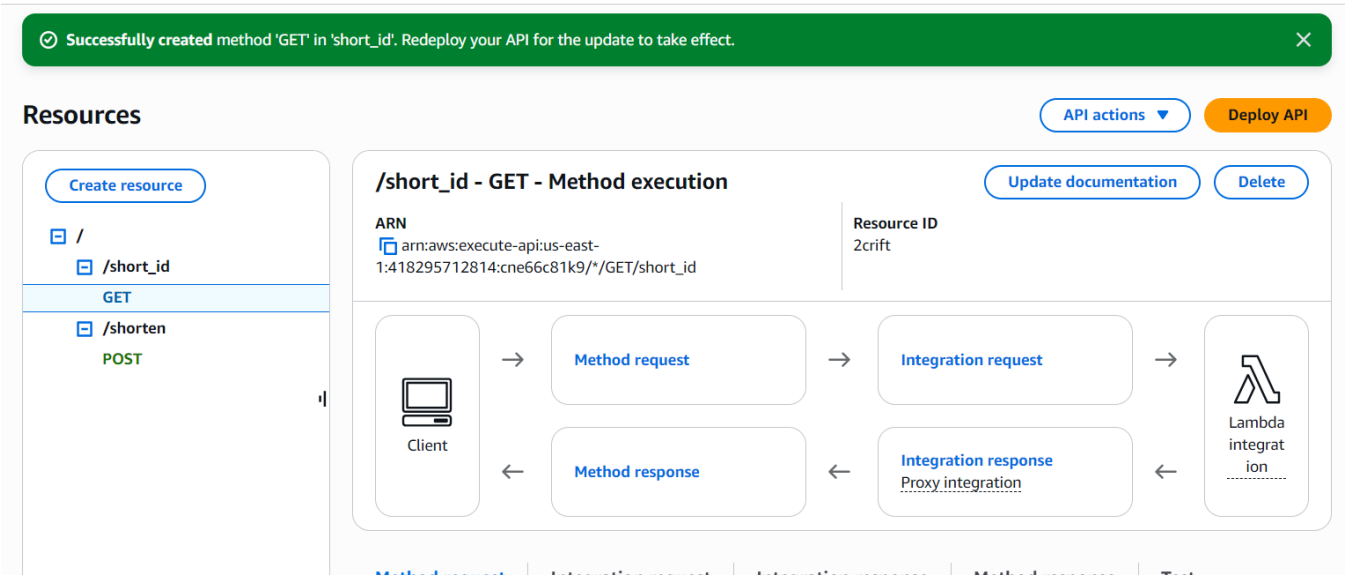
- Created a REST API.
- Configured POST /shorten method integrated with the URLShortener Lambda.
- Configured GET /{short_id} method integrated with the URLRedirect Lambda.
- Set up method request and integration request mappings.
- Enabled logging and tracing for debugging.





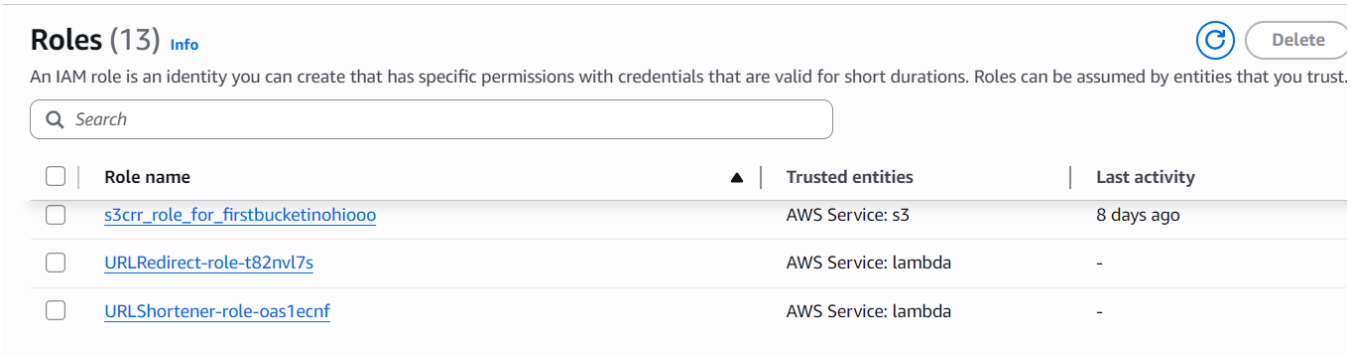

Resources
API actions
Deploy API
Create resource
/
/shorten
POST
/shorten - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-
/*/POST/shorten
Resource ID
zo1zs4
Client
Method request
Integration request
Method response
Integration response
Proxy integration
Lambda
integrat
ion
Method request
Integration request
Integration response
Method response
Test


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
dev
A new stage will be created with the default settings. Edit your stage settings on the Stage page.
Deployment description
Cancel
Deploy


Successfully created method 'GET' in 'short_id'. Redeploy your API for the update to take effect.
Resources
API actions
Deploy API
Create resource
/
/short_id
GET
/shorten
POST
/short_id - GET - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-
1:418295712814:cne66c81k9/*/GET/short_id
Resource ID
2crift
Client
Method request
Integration request
Method response
Integration response
Proxy integration
Lambda
integrat
ion

5. IAM Roles and Permissions

- Created IAM roles granting Lambda functions permission to read/write the DynamoDB table.
- Assigned roles to respective Lambda functions.



6. Testing

- Used the **API Gateway Console** to test both API methods directly:
  - **POST /shorten**

    Input: JSON body with long_url, e.g.:

    {

    "long_url": "https://www.example.com"

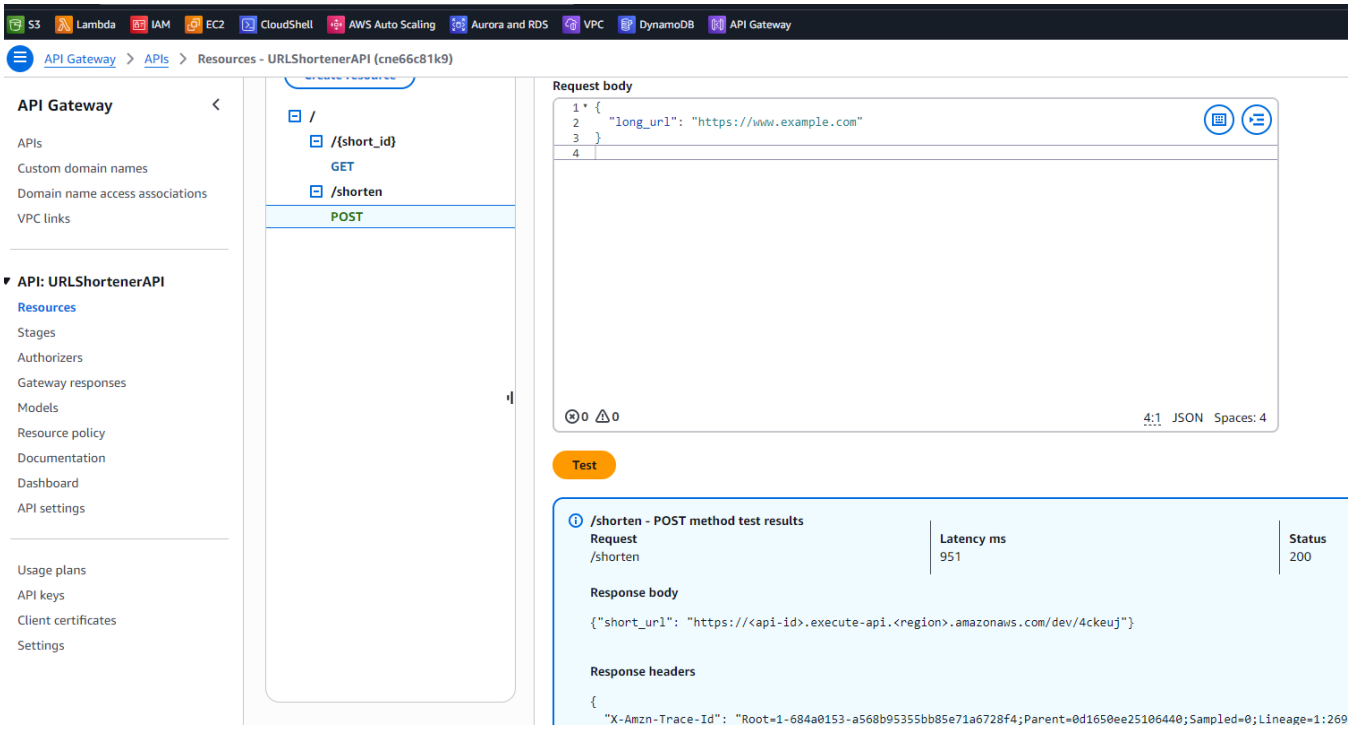
    }

    Output: JSON response with the generated short URL, for example:

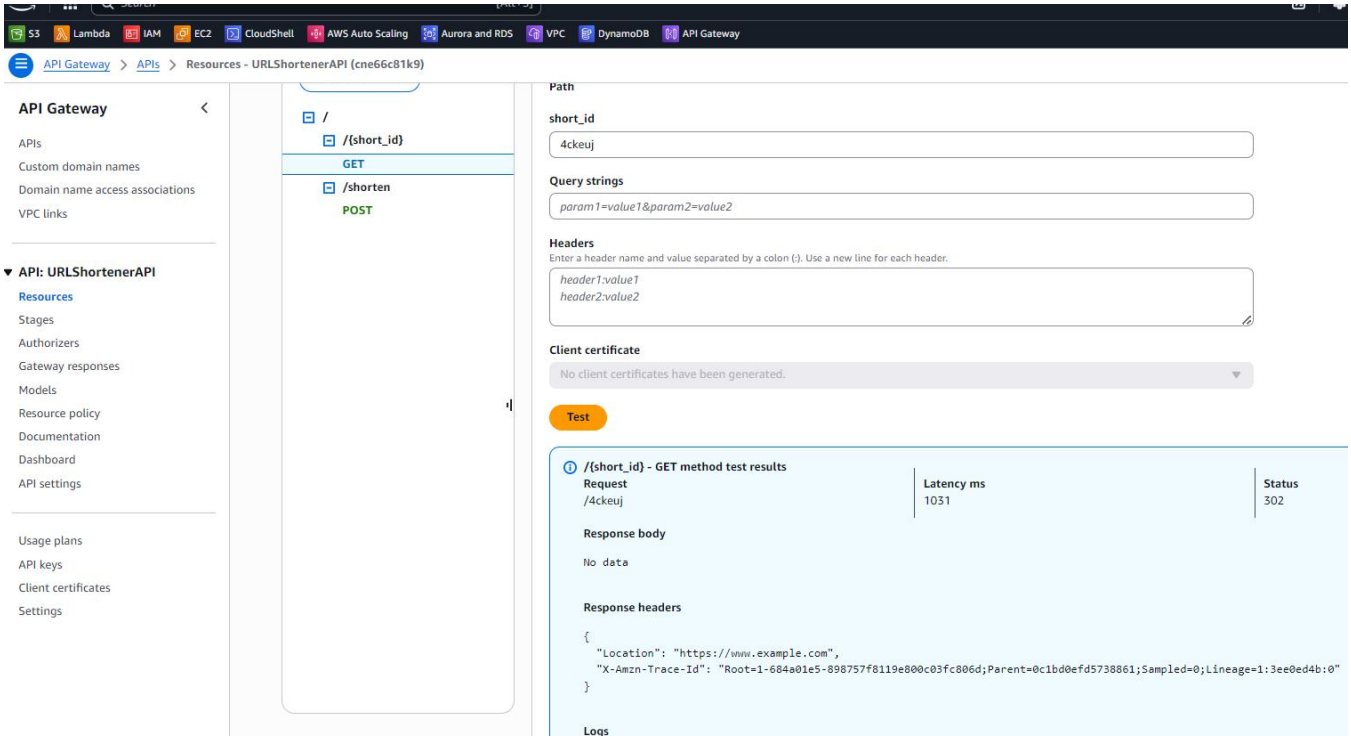
    {

```
{"short_url": "https://<api-id>.execute-api.<region>.amazonaws.com/dev/4ckeuj"}
```

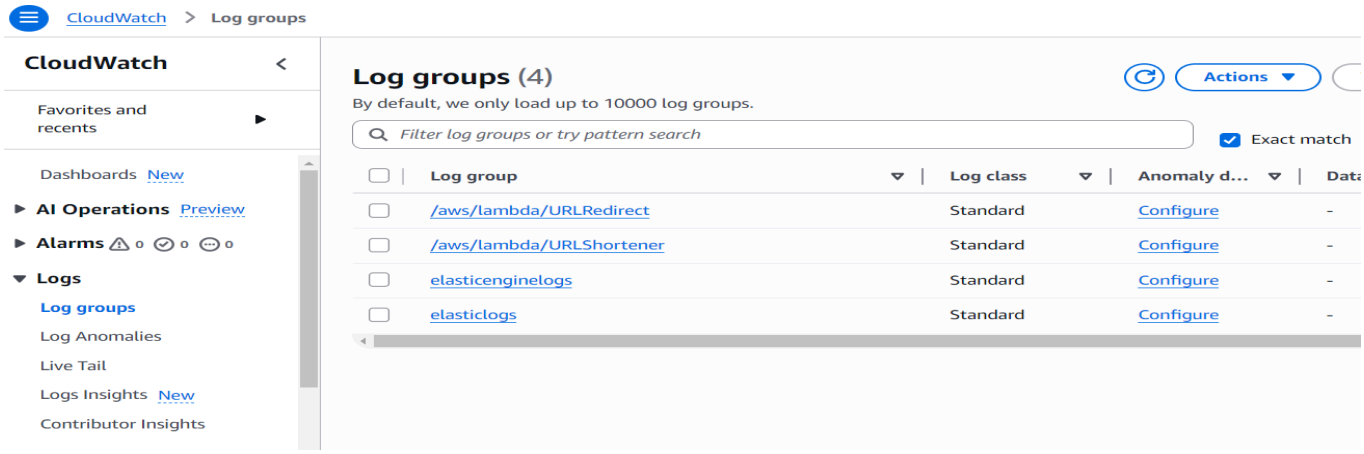
    }

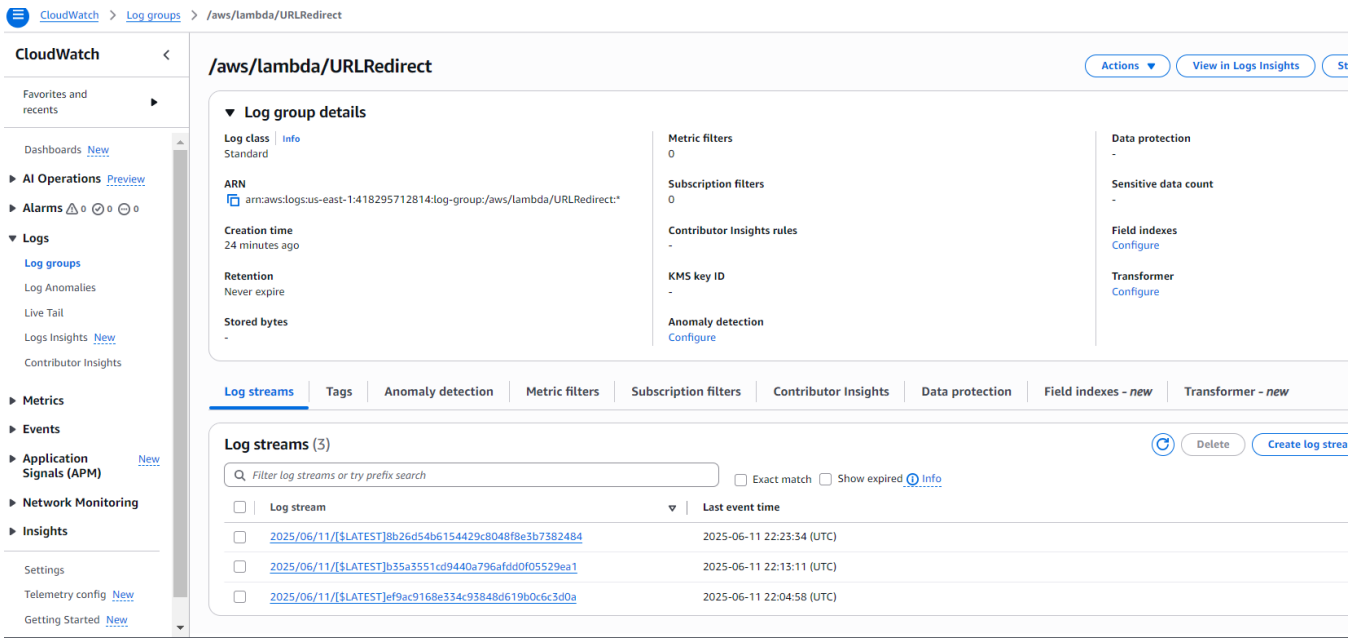


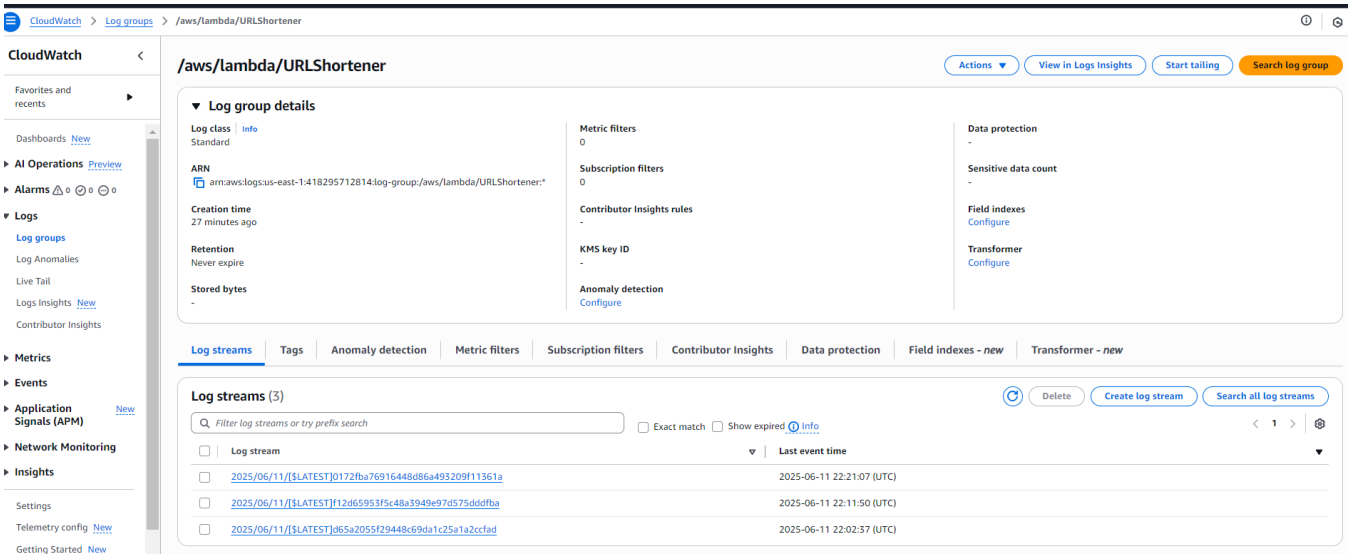
- **GET /{short_id}**
  - Tested by entering a short ID (e.g., 4ckeuj) in the path parameter field in the API Gateway Console test.
  - The API Gateway skipped authorization and directly invoked the Lambda function.
  - Response returned HTTP status **302** with a Location header redirecting to the original URL (https://www.example.com).
  - Example response headers from the test:
  - Status: 302
  - Location: https://www.example.com



Verified logs in CloudWatch confirming Lambda executions and integration latencies.

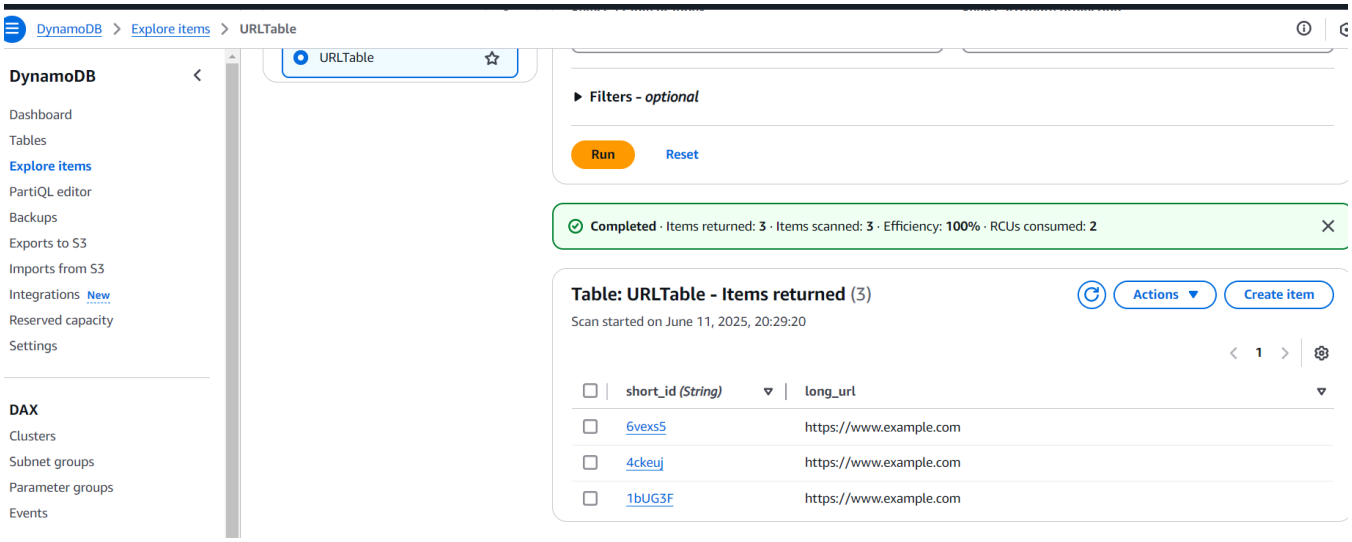






- Confirmed end-to-end functionality: submitting a long URL produces a short URL, and using the short URL correctly redirects to the original site.

7. DynamoDB key-value storage:



8. **Challenge Faced & Resolution**

While configuring the **GET method** in API Gateway to retrieve the original long URL using a dynamic path parameter (e.g., /short_id), I encountered a key challenge: entering {short_id} as the **resource path** in the console was mistakenly interpreted as a static string instead of a dynamic path variable. This resulted in API Gateway treating /short_id literally, rather than allowing unique IDs to be passed dynamically. After several failed tests and configuration attempts, the issue was resolved with the help of AWS CloudShell. By using a command like:

aws apigateway create-resource --rest-api-id cne66c81k9 --parent-id yob4bjpcsj --path-part '{short_id}' --region us-east-1

creating GET method :

aws apigateway put-method --rest-api-id cne66c81k9 --resource-id abc123 --http-method GET --authorization-type NONE --region us-east-1

Integrate the GET method with Lambda function

aws apigateway put-integration --rest-api-id cne66c81k9 --resource-id abc123 --http-method GET --type AWS_PROXY --integration-http-method POST --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/<lambda-arn>/invocations --region us-east-1

Add the permission and deploy:

aws apigateway create-deployment --rest-api-id cne66c81k9 --stage-name dev --region us-east-1

I confirmed that the API Gateway could correctly interpret and route dynamic paths once the resource was properly configured. This validation through CloudShell clarified the importance of precise

resource path syntax and dynamic variable setup within API Gateway. The method began working as intended once the path parameter was correctly defined and tested outside the console interface.

---

9. Conclusion

This project demonstrates building a simple, scalable, serverless URL shortener using AWS managed services. It provides hands-on experience with API Gateway, Lambda, DynamoDB, and CloudWatch, showcasing how to design REST APIs and implement redirect logic.