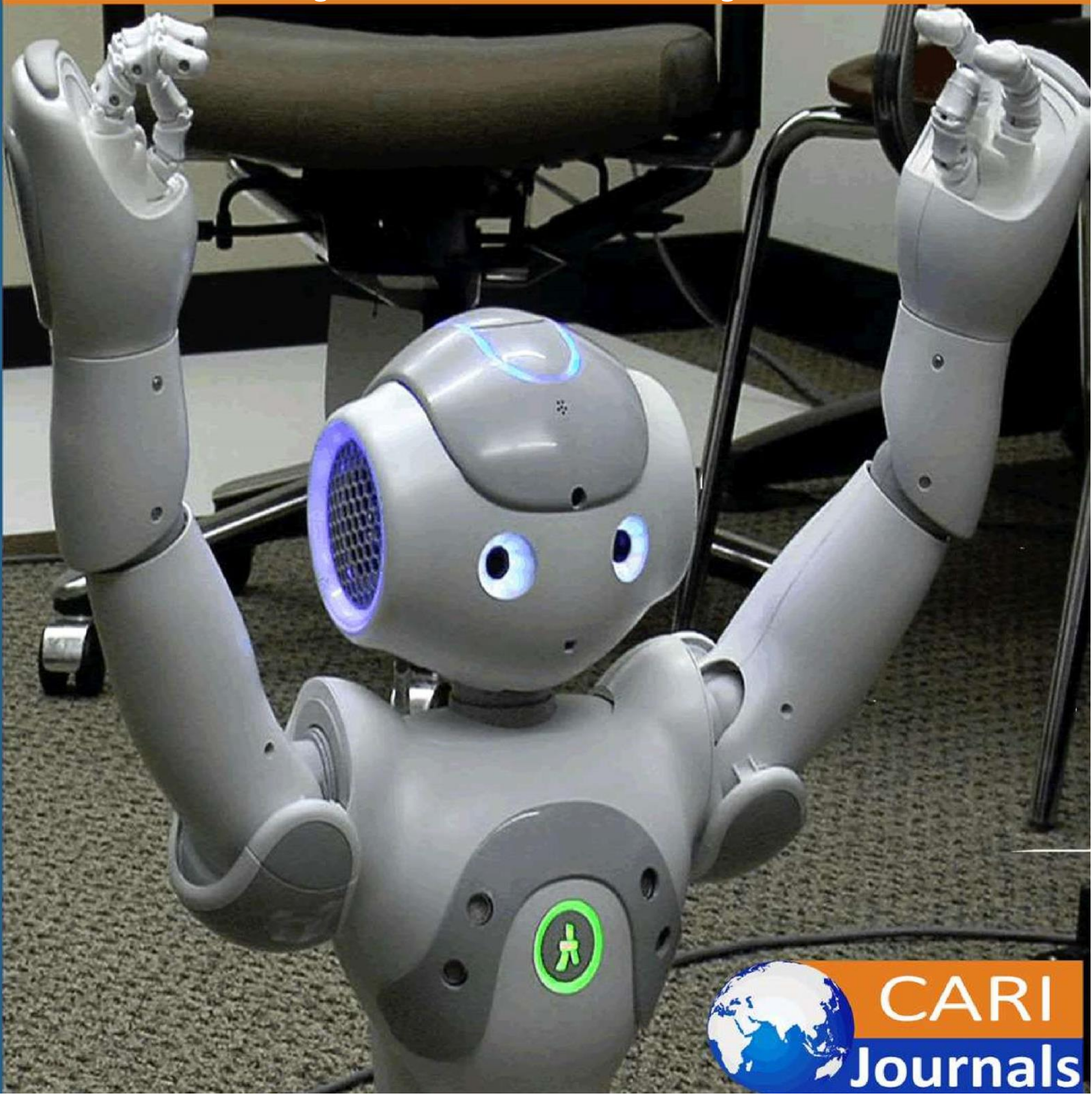
International Journal of

Computing and

Engineering

(IJCE)

Unveiling the AWS SAM Magic for Serverless Restful APIs: Architecting with ALB Path-Based Routing in AWS



International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 5, Issue No. 2, pp. 1 - 15, 2024
www.carijournals.org

Unveiling the AWS SAM Magic for Serverless Restful APIs: Architecting with ALB Path-Based Routing in AWS

Balasubrahmanya Balakrishna

https://orcid.org/0009-0000-1195-123X

Richmond, VA, USA

Accepted: 21st Feb 2024 Received in Revised Form: 5th Mar 2024 Published: 19th Mar 2024

Abstract

Purpose: This paper provides a thorough roadmap for developers, architects, and cloud enthusiasts who want to use the AWS Serverless Application Model (AWS SAM) to create a REST API and use the power of serverless computing. To handle HTTP requests effectively, the article focuses on deploying the API behind an Application Load Balancer (ALB) using path-based routing. The hands-on approach offers detailed instructions and valuable insights on planning, creating, and implementing serverless REST APIs. The focus is on the details of AWS SAM, examining its benefits and complexities. The paper makes the procedure easier to understand by providing thorough code excerpts, explanations, and pictures.

Methodology: The methodology covers local testing using the SAM CLI, allowing developers to validate the API's functionality before deployment.

Findings: The process also includes local testing with the SAM CLI, which enables developers to confirm the functioning of the API before deployment. To target the Lambda function, this paper will discuss AWS Lambda behind an ALB using a path-based listener rule on the ALB. The article’s conclusions cover essential topics like path-based routing, ALB integration, AWS SAM template structure, and recommended security and performance optimization practices.

Unique Contributor to Theory, Policy and Practice: Based on these findings, recommendations offer information on optimizing templates, ensuring secure deployment, and using local testing to speed up development. Finally, the article walks readers through deploying the built API to AWS via the SAM CLI, facilitating a seamless transfer from a local development environment to an environment in production. Ultimately, this paper provides readers with the know-how and abilities to successfully negotiate AWS SAM's complexities and build reliable serverless REST APIs.

Keywords: *AWS SAM CLI, AWS Lambda, ALB Path-Based Routing, AWS Powertools, AWS Observability*

INTRODUCTION

Thanks to its scalability, affordability, and ease of deployment, serverless computing has completely changed the application development industry. This paper is intended for cloud enthusiasts, developers, and architects eager to leverage the AWS Serverless Application Model[1] (AWS SAM) to realize the full potential of serverless architectures. The main focus of our investigation is developing and implementing a RESTful API behind an Application Load Balancer (ALB) that offers complex path-based routing for the best possible HTTP request processing.

This paper is based on the AWS SAM approach, well-known for simplifying the building of serverless applications on Amazon Web Services. We explore AWS SAM's nuances, revealing its benefits and complexities and offering practical takeaways via thorough code samples, explanations, and sample applications. Our exploration's hands-on methodology, which allows developers to create, build, and implement serverless REST APIs by following detailed instructions, is highly significant.

As above, comprehensive coverage of AWS Lambda, essential to the ALB, sets it apart from the others. We explain how the ALB's path-based listener rule[2] targets and calls the Lambda functions accurately. This crucial setup ensures that the serverless architecture integrates smoothly and enables the effective handling of various API calls behind an ALB.

The approach goes beyond development to include local testing with the SAM CLI, which enables developers to confirm the functioning of the API before deployment. Essential topics covered in the paper include the AWS SAM template layout, the ALB integration, the complexities of path-based routing, and the best practices for security and performance optimization. Based on these findings, recommendations have been made to provide developers with knowledge about secure deployment procedures, how to optimize templates, and how to employ local testing effectively for productive development cycles.

This paper gives a comprehensive overview of the deployment process as we wrap off our paper, walking them through the last stages of utilizing the SAM CLI to deploy the built API to AWS. This exploration aims to provide the necessary information and abilities to effectively navigate AWS SAM and build serverless REST APIs that are scalable and resilient.

UNVEILING ARCHITECTURAL PATTERN

We provide an architectural design in serverless computing that improves control and security by utilizing path-based routing at the Application Load Balancer (ALB) level. In contrast to traditional methods, this pattern presents a technique that allows REST calls that do not match the designated pathways to be intelligently prevented at the ALB, which stops the related AWS Lambda function from being called.

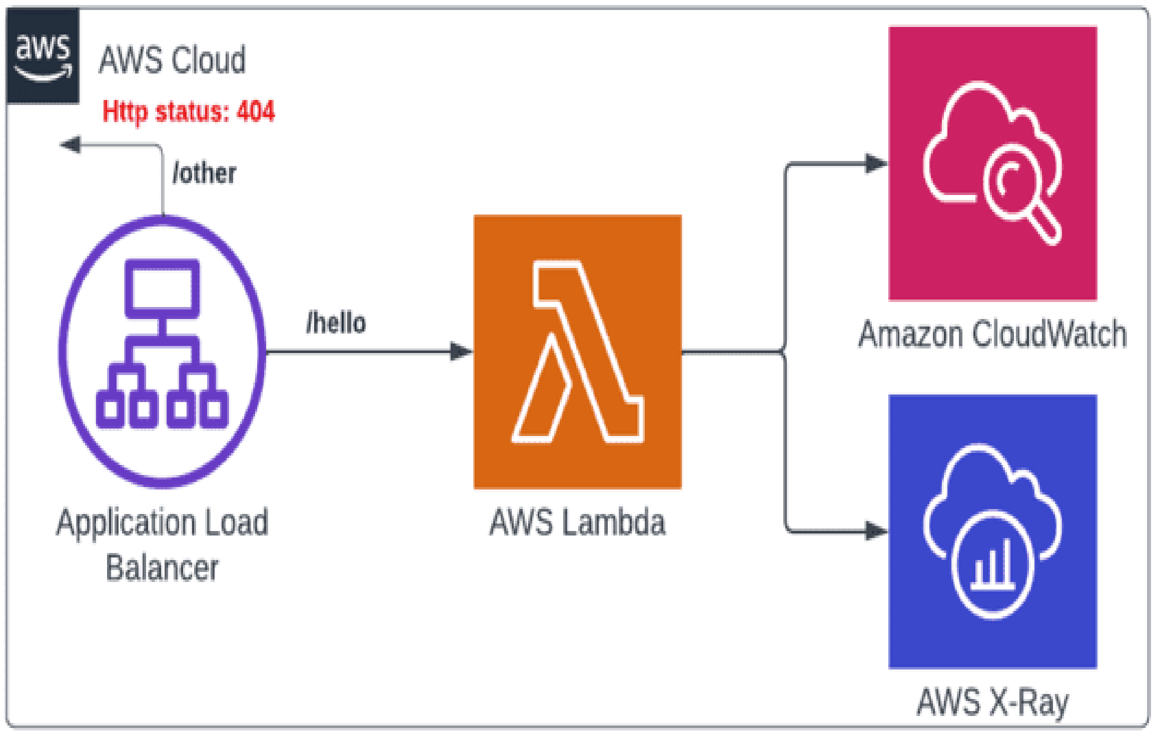


Figure 1. ALB fronted AWS Lambda

A. *The Significance of Path-Based Routing*

Path-based routing is an essential component of our architectural pattern, which enables the separation of API endpoints according to their pathways. This method makes it easier to regulate incoming HTTP requests more precisely by ensuring each request is sent to the correct Lambda function according to its specified path.

B. *Blocking Unwanted Requests at the ALB Level*

The capacity of the ALB to serve as a gatekeeper and stop REST calls that do not match any designated pathways from getting to the Lambda functions is what this architectural pattern is all about. We improve the serverless API's security posture by applying this rule at the ALB level, which reduces the attack surface.

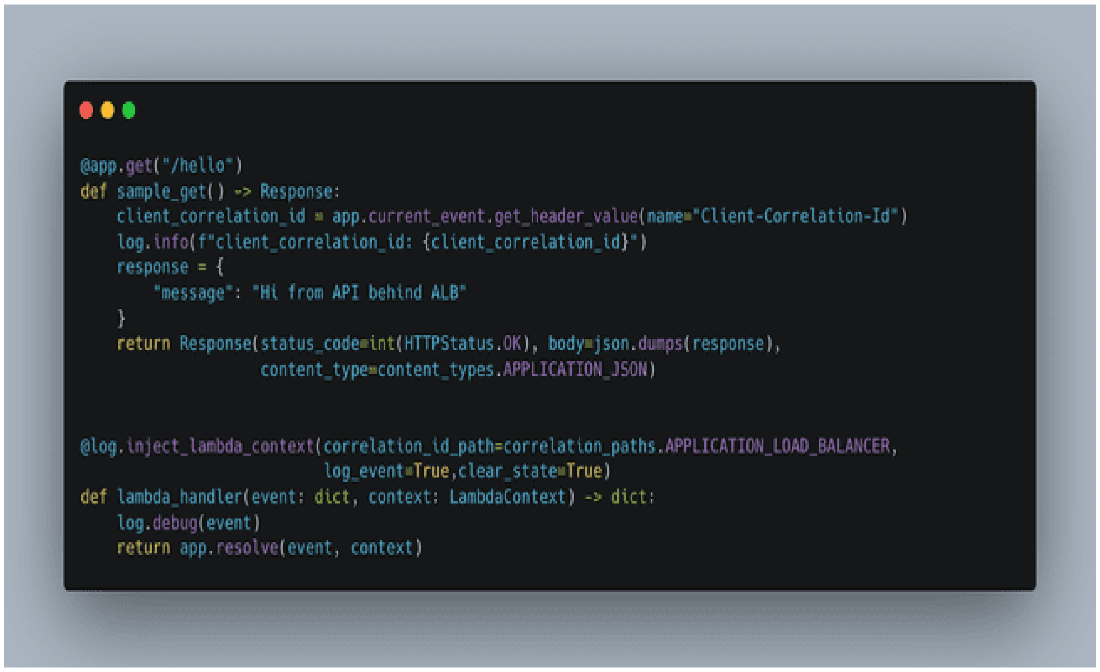
C. *Ensuring Efficient Resource Utilization*

In addition to improving security, blocking undesired requests at the ALB level also helps to maximize resource usage. Unnecessary computation and related expenses are reduced by blocking illegal or irrelevant requests from reaching the Lambda functions, leading to a more efficient and economical serverless architecture.

Practical Implementation and Configuration

The section includes practical insights into the ALB implementation and configuration of path-based routing. Code snippets, configuration settings, and illustrative examples guide readers through defining and enforcing path rules to achieve this architectural pattern.

I. Figure 2 illustrates a straightforward representation of a RESTful API constructed using the AWS Powertools for Lambda library in Python. The simplicity of the design is evident, showcasing the library's ease of use and effectiveness. The API responds with a clear and concise success message when an HTTP GET method request is made to the resource path /hello. This uncomplicated setup highlights the power and user-friendly nature of the AWS Powertools for Lambda library in Python, offering developers an efficient framework for building robust serverless applications.



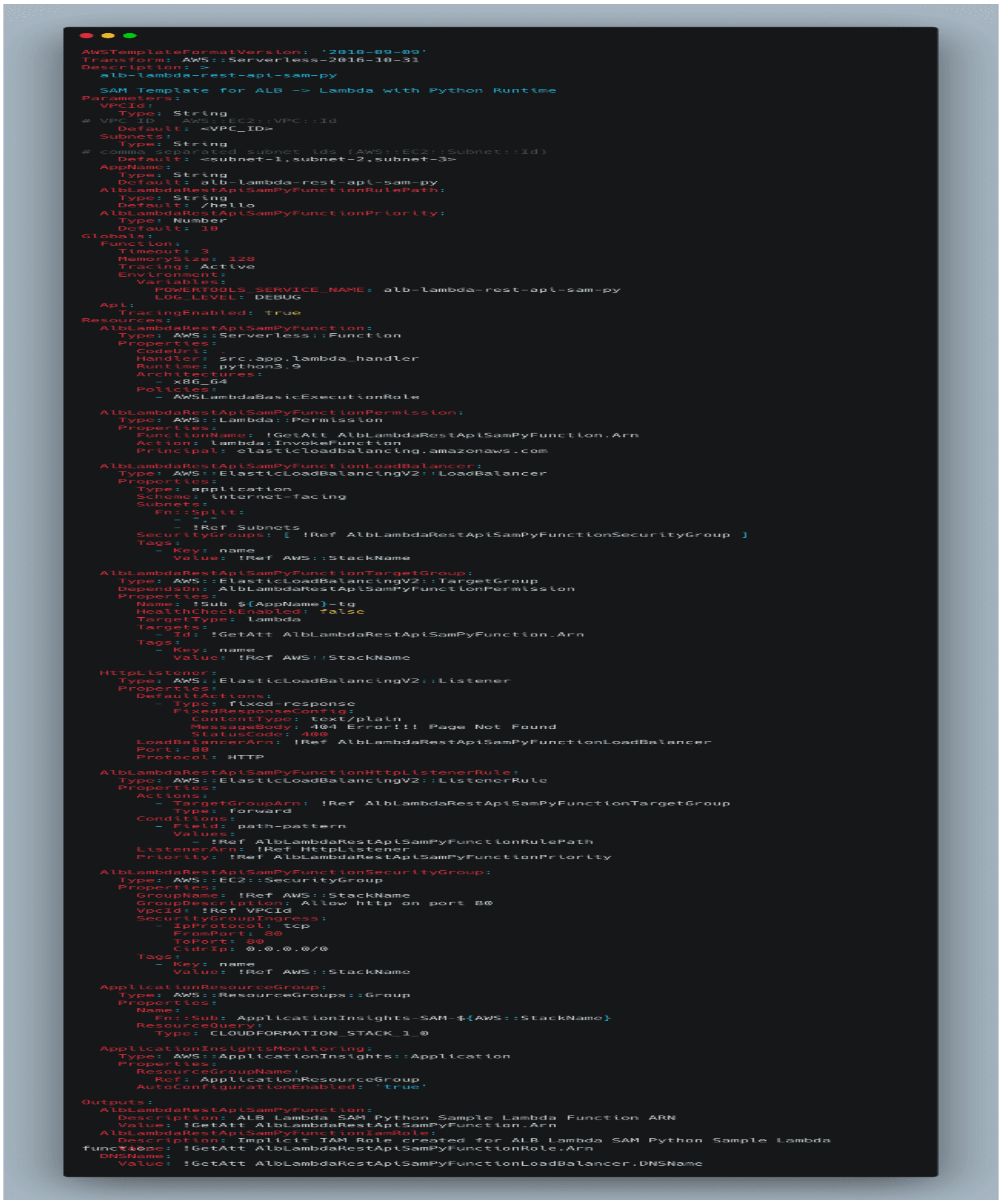
```python
@app.get("/hello")
def sample_get() -> Response:
    client_correlation_id = app.current_event.get_header_value(name="Client-Correlation-Id")
    log.info(f"client_correlation_id: {client_correlation_id}")
    response = {
        "message": "Hi from API behind ALB"
    }
    return Response(status_code=int(HTTPStatus.OK), body=json.dumps(response),
                    content_type=content_types.APPLICATION_JSON)


@log.inject_lambda_context(correlation_id_path=correlation_paths.APPLICATION_LOAD_BALANCER,
                           log_event=True,clear_state=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    log.debug(event)
    return app.resolve(event, context)
```

Figure 2. Simple REST API wilt AWS Lambda Powertools

II. To streamline local API testing, we utilize the SAM CLI with its tailored emulation image for Python, faithfully replicating the Lambda environment for realistic local development. Figure 3 illustrates the SAM template explicitly crafted for this purpose, showcasing the orchestrated use of AWS SAM to establish a seamless local testing environment.

The SAM CLI seamlessly extends the capabilities of the AWS CLI, incorporating essential features for building and validating Lambda applications. Powered by Docker, it orchestrates function execution within an Amazon Linux environment meticulously aligned with Lambda runtime specifications, sourced from sam/build-python3.9. This emulation ensures a precise replication of the application's build environment and API, enhancing the accuracy of local testing processes.



```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  alb-lambda-rest-api-sam-py

  SAM Template for ALB -> Lambda with Python Runtime
Parameters:
  VPCId:
    Type: String
# VPC ID - AWS::EC2::VPC::Id
    Default: <VPC_ID>
  Subnets:
    Type: String
# comma separated subnet ids (AWS::EC2::Subnet::Id)
    Default: <subnet-1,subnet-2,subnet-3>
  AppName:
    Type: String
    Default: alb-lambda-rest-api-sam-py
  AlbLambdaRestApiSamPyFunctionRulePath:
    Type: String
    Default: /hello
  AlbLambdaRestApiSamPyFunctionPriority:
    Type: Number
    Default: 10
Globals:
  Function:
    Timeout: 3
    MemorySize: 128
    Tracing: Active
    Environment:
      Variables:
        POWERTOOLS_SERVICE_NAME: alb-lambda-rest-api-sam-py
        LOG_LEVEL: DEBUG
  Api:
    TracingEnabled: true
Resources:
  AlbLambdaRestApiSamPyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Handler: src.app.lambda_handler
      Runtime: python3.9
      Architectures:
        - x86_64
      Policies:
        - AWSLambdaBasicExecutionRole

  AlbLambdaRestApiSamPyFunctionPermission:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !GetAtt AlbLambdaRestApiSamPyFunction.Arn
      Action: lambda:InvokeFunction
      Principal: elasticloadbalancing.amazonaws.com

  AlbLambdaRestApiSamPyFunctionLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Type: application
      Scheme: internet-facing
      Subnets:
        Fn::Split:
          - ","
          - !Ref Subnets
      SecurityGroups: [ !Ref AlbLambdaRestApiSamPyFunctionSecurityGroup ]
      Tags:
        - Key: name
          Value: !Ref AWS::StackName

  AlbLambdaRestApiSamPyFunctionTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    DependsOn: AlbLambdaRestApiSamPyFunctionPermission
    Properties:
      Name: !Sub ${AppName}-tg
      HealthCheckEnabled: false
      TargetType: lambda
      Targets:
        - Id: !GetAtt AlbLambdaRestApiSamPyFunction.Arn
      Tags:
        - Key: name
          Value: !Ref AWS::StackName

  HttpListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: fixed-response
          FixedResponseConfig:
            ContentType: text/plain
            MessageBody: 404 Error!!! Page Not Found
            StatusCode: 400
      LoadBalancerArn: !Ref AlbLambdaRestApiSamPyFunctionLoadBalancer
      Port: 80
      Protocol: HTTP

  AlbLambdaRestApiSamPyFunctionHttpListenerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
        - TargetGroupArn: !Ref AlbLambdaRestApiSamPyFunctionTargetGroup
          Type: forward
      Conditions:
        - Field: path-pattern
          Values:
            - !Ref AlbLambdaRestApiSamPyFunctionRulePath
      ListenerArn: !Ref HttpListener
      Priority: !Ref AlbLambdaRestApiSamPyFunctionPriority

  AlbLambdaRestApiSamPyFunctionSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Ref AWS::StackName
      GroupDescription: Allow http on port 80
      VpcId: !Ref VPCId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
      Tags:
        - Key: name
          Value: !Ref AWS::StackName

  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Sub: ApplicationInsights-SAM-${AWS::StackName}
      ResourceQuery:
        Type: CLOUDFORMATION_STACK_1_0

  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Ref: ApplicationResourceGroup
      AutoConfigurationEnabled: 'true'

Outputs:
  AlbLambdaRestApiSamPyFunction:
    Description: ALB Lambda SAM Python Sample Lambda Function ARN
    Value: !GetAtt AlbLambdaRestApiSamPyFunction.Arn
  AlbLambdaRestApiSamPyFunctionIamRole:
    Description: Implicit IAM Role created for ALB Lambda SAM Python Sample Lambda
function
    Value: !GetAtt AlbLambdaRestApiSamPyFunctionRole.Arn
  DNSName:
    Value: !GetAtt AlbLambdaRestApiSamPyFunctionLoadBalancer.DNSName
```

1. *Parameters*:

- `VPCId`: Specifies the VPC ID where the resources will be deployed.

- `Subnets`: Specifies the comma-separated subnet IDs where the ALB and Lambda function will be deployed.

- `AppName`: Specifies the name of the application.

- `AlbLambdaRestApiSamPyFunctionRulePath`: Specifies the path pattern for the ALB listener rule.

- `AlbLambdaRestApiSamPyFunctionPriority`: Specifies the priority of the ALB listener rule.

2. *Globals*:

- Sets global configurations for Lambda functions (`Timeout,` `MemorySize,` `Tracing,` and `Environment` variables).

3. *Resources*:

- Lambda Function (`AlbLambdaRestApiSamPyFunction`):

  - Specifies the AWS Lambda function named `AlbLambdaRestApiSamPyFunction.`

  - Configures the function with Python 3.9 runtime, a specific handler (`src.app.lambda_handler`), and other settings.

  - Defines environment variables and assigns IAM role (`AWSLambdaBasicExecutionRole`).

- Lambda Function Permission (`AlbLambdaRestApiSamPyFunctionPermission`):

  - Grants permission to the ALB to invoke the Lambda function.

- *Elastic Load Balancer (`AlbLambdaRestApiSamPyFunctionLoadBalancer`):*

  - Creates an ALB of type `application` and `internet-facing.`

  - Associates the ALB with specified subnets and security groups.

  - Adds tags to the ALB for identification.

- *Elastic Load Balancer Target Group (`AlbLambdaRestApiSamPyFunctionTargetGroup`):*

  - Creates a target group for the Lambda function in the ALB.

- *ALB Listener (`HttpListener`):*

  - Configures an ALB listener on port 80 with a default fixed response for 404 errors.

- *ALB Listener Rule (`AlbLambdaRestApiSamPyFunctionHttpListenerRule`):*
  - Associates the ALB listener rule with the Lambda function's target group based on the specified path pattern.
- *Security Group (`AlbLambdaRestApiSamPyFunctionSecurityGroup`):*
  - Creates a security group allowing HTTP traffic on port 80 from any IP (`0.0.0.0/0`).
- *Resource Group (`ApplicationResourceGroup`):*
  - Creates an AWS Resource Group for monitoring purposes.
- *Application Insights Monitoring (`ApplicationInsightsMonitoring`):*
  - Sets up Application Insights monitoring for the application.

4. *Outputs*:

- Provides outputs for the Lambda function ARN, Lambda function IAM role ARN, and the DNS name of the ALB.

Figure 3. SAM Template (template.yaml)

This template creates a serverless architecture where an ALB forwards HTTP requests to a Lambda function, and the application's performance is monitored using AWS Application Insights. The parameters allow customized VPC settings, subnets, and application details.

III. When hosting AWS Lambda behind an Application Load Balancer (ALB). AWS SAM simplifies generating ALB events for local testing using the command *sam local generate-event alb request*. This command can be easily modified to simulate a GET request method, a step in testing our Lambda API running locally.

The commands below need to be executed to run the Lambda function locally.

a. First, ensure SAM CLI has the dependencies specified in the *requirements.txt* file. Using pipenv can be conveniently achieved with the command:

pipenv lock -r > requirements.txt

b. Next, execute the command below:

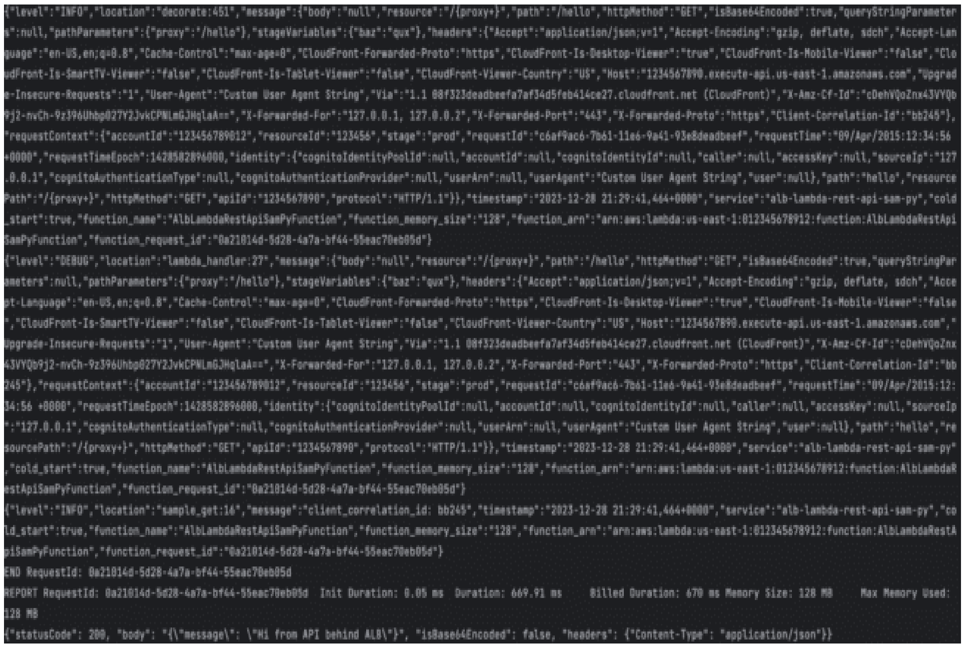
sam build.

This command automates the application preparation for deployment and ensures that dependencies are accurately installed, and the code is efficiently packaged for subsequent deployment to AWS Lambda. By default, the *sam build* command searches for a *template.yaml* file in the current directory. However, the --*template* or -*t* option can specify an alternative template file.

c. Finally, execute the command below to run and invoke the lambda function:

sam local invoke -e events/event.json.

Where *event.json* is the ALB event generated by executing *the sam generate-event* command as discussed above. During the execution of the *sam invoke* command, the SAM emulation image is automatically downloaded. This emulation image plays a pivotal role in simulating the Lambda environment locally, enabling the seamless execution of the Lambda function. Subsequently, an event is passed to the Lambda function, allowing developers to observe and validate the behavior of their serverless application in a local context. This iterative process, made efficient by SAM emulation[8], enhances the local testing experience, providing valuable insights into the functionality of the Lambda function before deployment to AWS.



```
{"level":"INFO","location":"decorate:451","message":{"body":"null","resource":"/{proxy+}","path":"/hello","httpMethod":"GET","isBase64Encoded":true,"queryStringParameter
s":null,"pathParameters":{"proxy":"/hello"},"stageVariables":{"baz":"qux"},"headers":{"Accept":"application/json;v=1","Accept-Encoding":"gzip, deflate, sdch","Accept-Lan
guage":"en-US,en;q=0.8","Cache-Control":"max-age=0","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","Clo
udFront-Is-SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-Country":"US","Host":"1234567890.execute-api.us-east-1.amazonaws.com","Upgrad
e-Insecure-Requests":"1","User-Agent":"Custom User Agent String","Via":"1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"cDehVQoZnx43VYQb
9j2-nvCh-9z396Uhbp027Y2JvkCPNLmGJHqlaA==","X-Forwarded-For":"127.0.0.1, 127.0.0.2","X-Forwarded-Port":"443","X-Forwarded-Proto":"https","Client-Correlation-Id":"bb245"},
"requestContext":{"accountId":"123456789012","resourceId":"123456","stage":"prod","requestId":"c6af9ac6-7b61-11e6-9a41-93e8deadbeef","requestTime":"09/Apr/2015:12:34:56
+0000","requestTimeEpoch":1428582896000,"identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"accessKey":null,"sourceIp":"127
.0.0.1","cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,"userAgent":"Custom User Agent String","user":null},"path":"hello","resource
Path":"/{proxy+}","httpMethod":"GET","apiId":"1234567890","protocol":"HTTP/1.1"}},"timestamp":"2023-12-28 21:29:41,464+0000","service":"alb-lambda-rest-api-sam-py","cold
_start":true,"function_name":"AlbLambdaRestApiSamPyFunction","function_memory_size":"128","function_arn":"arn:aws:lambda:us-east-1:012345678912:function:AlbLambdaRestApi
SamPyFunction","function_request_id":"0a21014d-5d28-4a7a-bf44-55eac70eb05d"}
{"level":"DEBUG","location":"lambda_handler:27","message":{"body":"null","resource":"/{proxy+}","path":"/hello","httpMethod":"GET","isBase64Encoded":true,"queryStringPar
ameters":null,"pathParameters":{"proxy":"/hello"},"stageVariables":{"baz":"qux"},"headers":{"Accept":"application/json;v=1","Accept-Encoding":"gzip, deflate, sdch","Acce
pt-Language":"en-US,en;q=0.8","Cache-Control":"max-age=0","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-Viewer":"true","CloudFront-Is-Mobile-Viewer":"false
","CloudFront-Is-SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-Country":"US","Host":"1234567890.execute-api.us-east-1.amazonaws.com","
Upgrade-Insecure-Requests":"1","User-Agent":"Custom User Agent String","Via":"1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"cDehVQoZnx
43VYQb9j2-nvCh-9z396Uhbp027Y2JvkCPNLmGJHqlaA==","X-Forwarded-For":"127.0.0.1, 127.0.0.2","X-Forwarded-Port":"443","X-Forwarded-Proto":"https","Client-Correlation-Id":"bb
245"},"requestContext":{"accountId":"123456789012","resourceId":"123456","stage":"prod","requestId":"c6af9ac6-7b61-11e6-9a41-93e8deadbeef","requestTime":"09/Apr/2015:12:
34:56 +0000","requestTimeEpoch":1428582896000,"identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"accessKey":null,"sourceIp
":"127.0.0.1","cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,"userAgent":"Custom User Agent String","user":null},"path":"hello","re
sourcePath":"/{proxy+}","httpMethod":"GET","apiId":"1234567890","protocol":"HTTP/1.1"}},"timestamp":"2023-12-28 21:29:41,464+0000","service":"alb-lambda-rest-api-sam-py"
,"cold_start":true,"function_name":"AlbLambdaRestApiSamPyFunction","function_memory_size":"128","function_arn":"arn:aws:lambda:us-east-1:012345678912:function:AlbLambdaR
estApiSamPyFunction","function_request_id":"0a21014d-5d28-4a7a-bf44-55eac70eb05d"}
{"level":"INFO","location":"sample_get:16","message":"client_correlation_id: bb245","timestamp":"2023-12-28 21:29:41,464+0000","service":"alb-lambda-rest-api-sam-py","co
ld_start":true,"function_name":"AlbLambdaRestApiSamPyFunction","function_memory_size":"128","function_arn":"arn:aws:lambda:us-east-1:012345678912:function:AlbLambdaRestA
piSamPyFunction","function_request_id":"0a21014d-5d28-4a7a-bf44-55eac70eb05d"}
END RequestId: 0a21014d-5d28-4a7a-bf44-55eac70eb05d
REPORT RequestId: 0a21014d-5d28-4a7a-bf44-55eac70eb05d  Init Duration: 0.05 ms  Duration: 669.91 ms     Billed Duration: 670 ms Memory Size: 128 MB     Max Memory Used:
128 MB
{"statusCode": 200, "body": "{\"message\": \"Hi from API behind ALB\"}", "isBase64Encoded": false, "headers": {"Content-Type": "application/json"}}
```

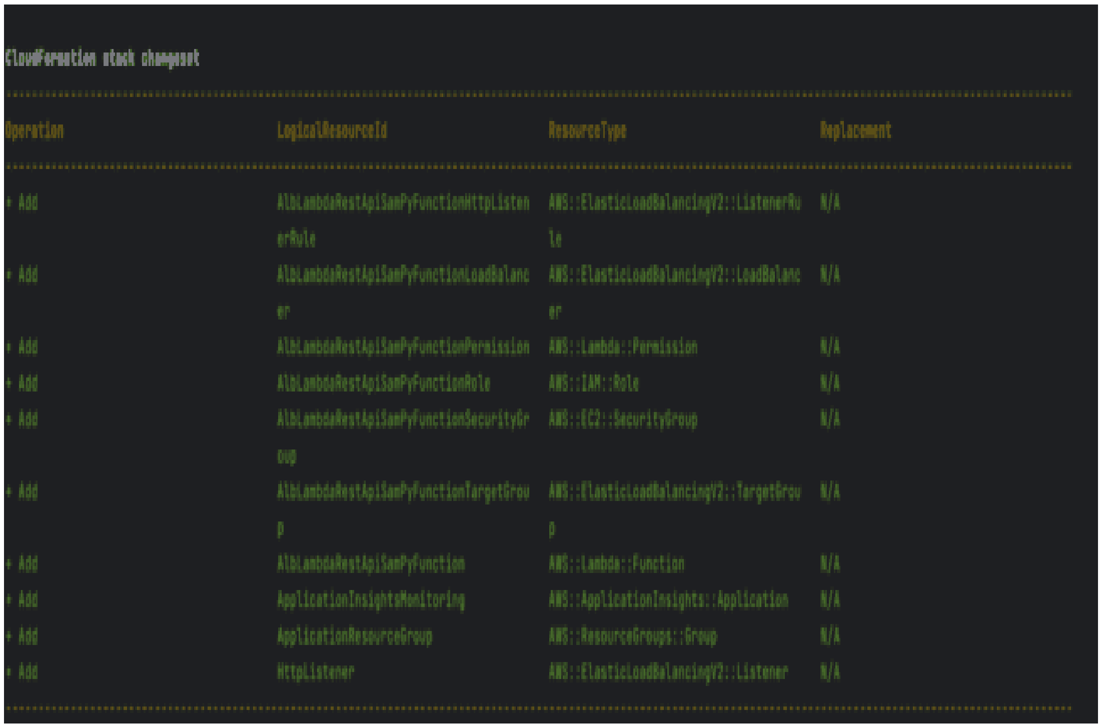
Figure 4. Successful Lamda invocation in local

IV. Ultimately, to bring our serverless application to AWS, we can initiate the deployment using the SAM CLI command:

sam deploy --guided

This command not only deploys the application but also guides users through a configuration process, ensuring a tailored and seamless deployment experience. Once deployed, thorough testing of the full functionality, including the intricacies of ALB path-based routing, can be performed. This final step ensures that the serverless application operates as expected in the AWS environment, providing confidence in its readiness for production use.

Below, Figure 5 shows the list of resources created in AWS:



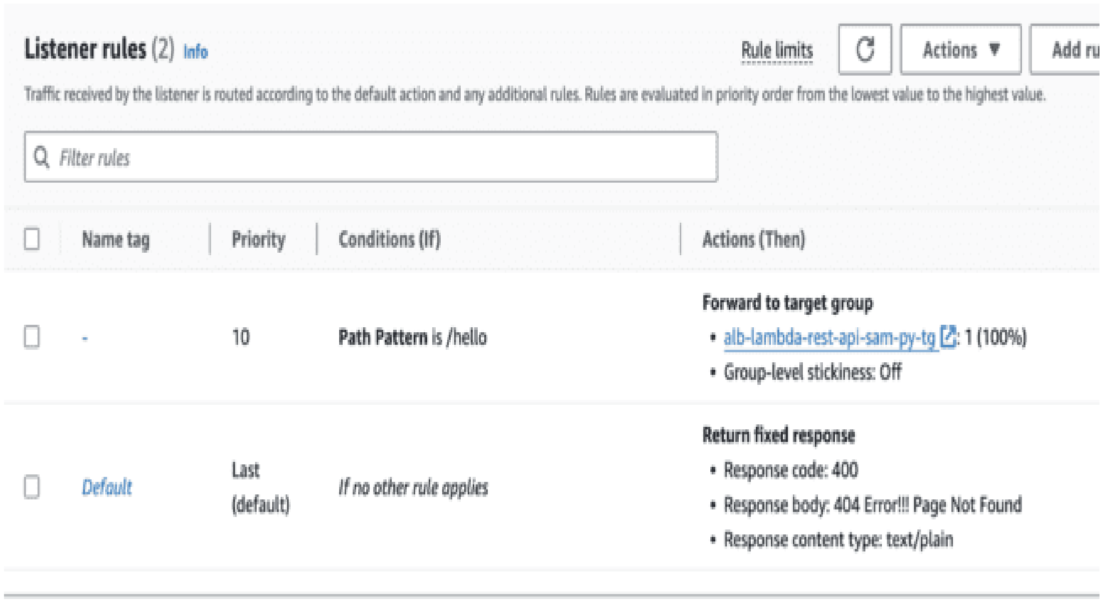
```
CloudFormation stack changeset
---------------------------------------------------------------------------------------------------------------------------
Operation                          LogicalResourceId                  ResourceType                       Replacement
---------------------------------------------------------------------------------------------------------------------------
+ Add                              AlbLambdaRestApiSamPyFunctionHttpListen  AWS::ElasticLoadBalancingV2::ListenerRu  N/A
                                   erRule                             le
+ Add                              AlbLambdaRestApiSamPyFunctionLoadBalanc  AWS::ElasticLoadBalancingV2::LoadBalanc  N/A
                                   er                                 er
+ Add                              AlbLambdaRestApiSamPyFunctionPermission  AWS::Lambda::Permission            N/A
+ Add                              AlbLambdaRestApiSamPyFunctionRole  AWS::IAM::Role                     N/A
+ Add                              AlbLambdaRestApiSamPyFunctionSecurityGr  AWS::EC2::SecurityGroup            N/A
                                   oup
+ Add                              AlbLambdaRestApiSamPyFunctionTargetGrou  AWS::ElasticLoadBalancingV2::TargetGrou  N/A
                                   p                                  p
+ Add                              AlbLambdaRestApiSamPyFunction      AWS::Lambda::Function              N/A
+ Add                              ApplicationInsightsMonitoring      AWS::ApplicationInsights::Application  N/A
+ Add                              ApplicationResourceGroup           AWS::ResourceGroups::Group         N/A
+ Add                              HttpListener                       AWS::ElasticLoadBalancingV2::Listener  N/A
---------------------------------------------------------------------------------------------------------------------------
```

Figure 5. List of AWS resources created as part of *sam deploy* command executed from local

Testing Lambda API Hosted In AWS

Once the application is deployed to AWS, we can test the API behind the Lambda function as any other rest API behind ALB. Let us test 2 basic scenarios for this pattern:

The ALB Listener Rule setup to enable path-based routing looks as shown in Figure 6 below:



Listener rules (2) Info

Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.

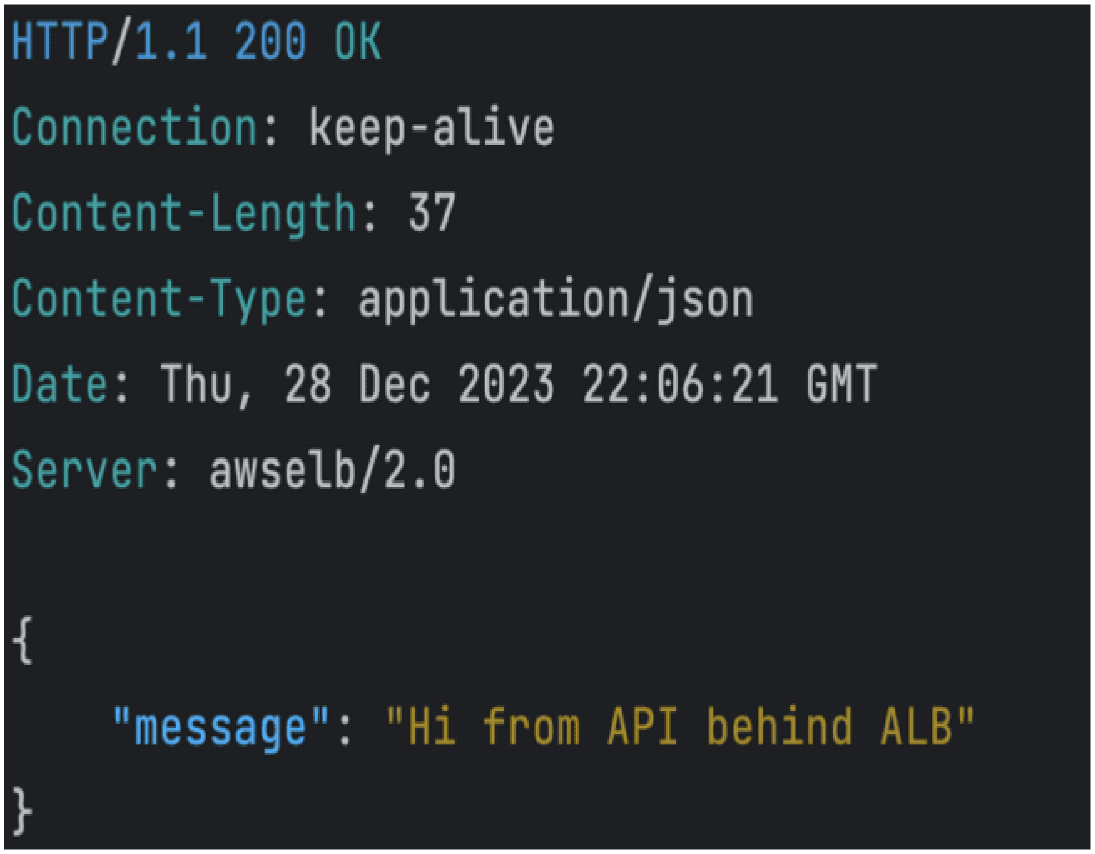
☐	Name tag	Priority	Conditions (If)	Actions (Then)
☐	-	10	**Path Pattern** is /hello	**Forward to target group** • alb-lambda-rest-api-sam-py-tg: 1 (100%) • Group-level stickiness: Off
☐	*Default*	Last (default)	*If no other rule applies*	**Return fixed response** • Response code: 400 • Response body: 404 Error!!! Page Not Found • Response content type: text/plain

Figure 6. ALB Listener rule with Path Pattern Condition

Simplifying the process, we will utilize xh utility[5], a lightweight and elegantly formatted Command-Line Interface (CLI). The call template for the application, as illustrated in Figure 2, takes on the following straightforward structure:

xh http://{ALB_ID}.{REGION}.elb.amazonaws.com/hello Client-Correlation-Id:bb245

Successful, response on invoke would return a response as below, Figure 7:

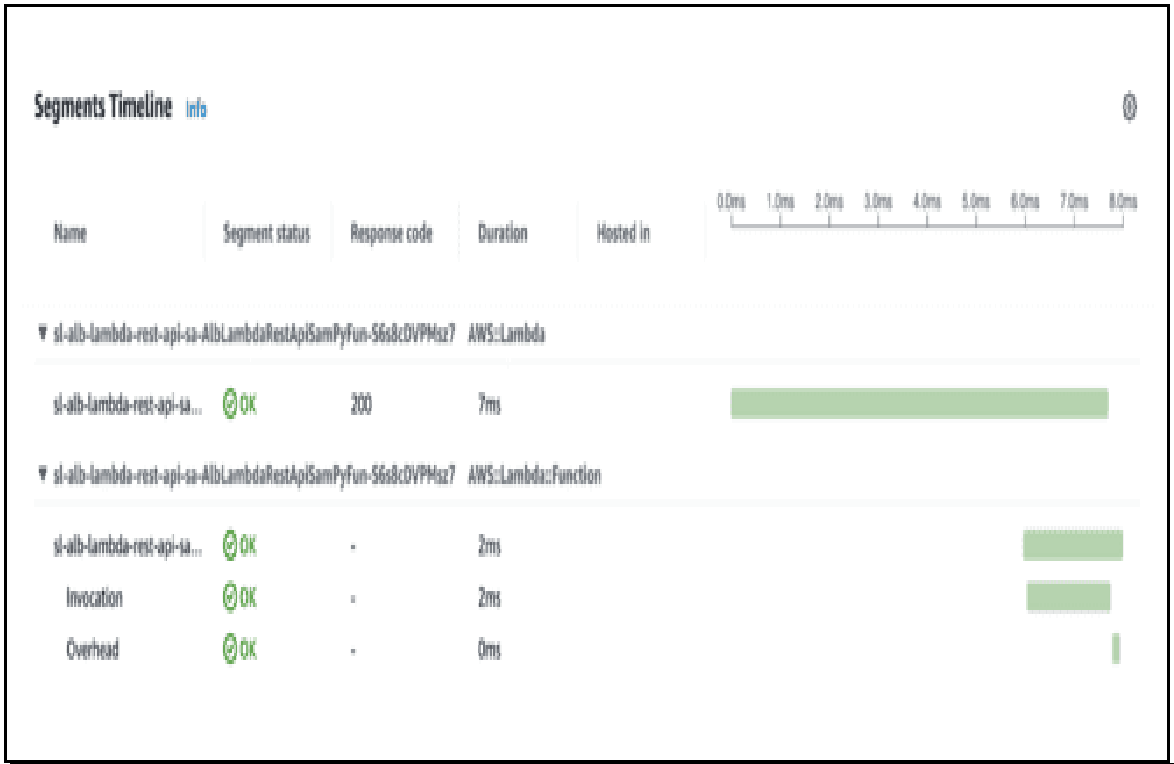

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 37
Content-Type: application/json
Date: Thu, 28 Dec 2023 22:06:21 GMT
Server: awselb/2.0

{
    "message": "Hi from API behind ALB"
}
```

Figure 7. A successful response from Lambda API

Capturing both CloudWatch logs[3] and X-Ray[4] traces, the API seamlessly integrates with the AWS Lambda Powertools for Python, as depicted in the visual representation provided in Figure 8. This integration showcases the synergy between powerful monitoring tools, offering developers a comprehensive view of the performance and behavior of the serverless application.

International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 5, Issue No. 2, pp. 1 - 15, 2024


Segments Timeline Info

Name	Segment status	Response code	Duration	Hosted in
▾ sl-alb-lambda-rest-api-sa-AlbLambdaRestApiSamPyFun-S6s8cDVPMsz7	AWS::Lambda			
sl-alb-lambda-rest-api-sa...	✓ OK	200	7ms	
▾ sl-alb-lambda-rest-api-sa-AlbLambdaRestApiSamPyFun-S6s8cDVPMsz7	AWS::Lambda::Function			
sl-alb-lambda-rest-api-sa...	✓ OK	-	2ms	
Invocation	✓ OK	-	2ms	
Overhead	✓ OK	-	0ms	

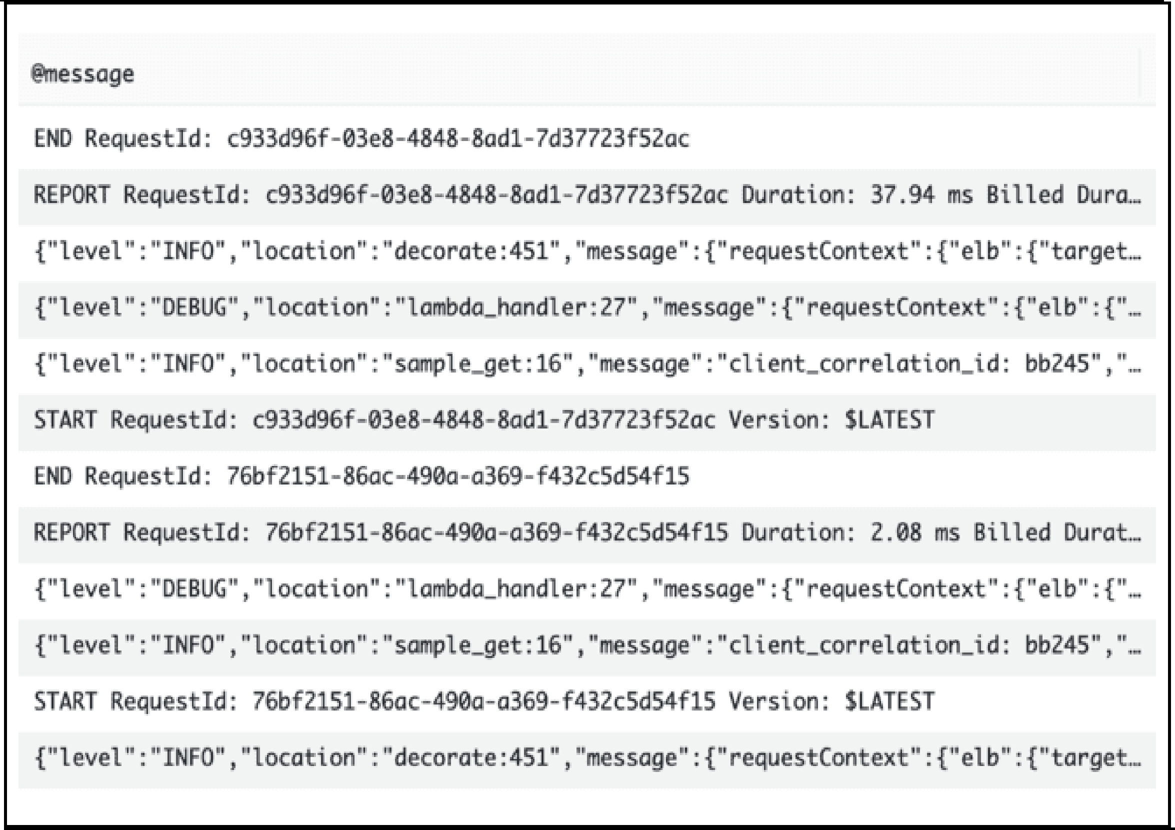
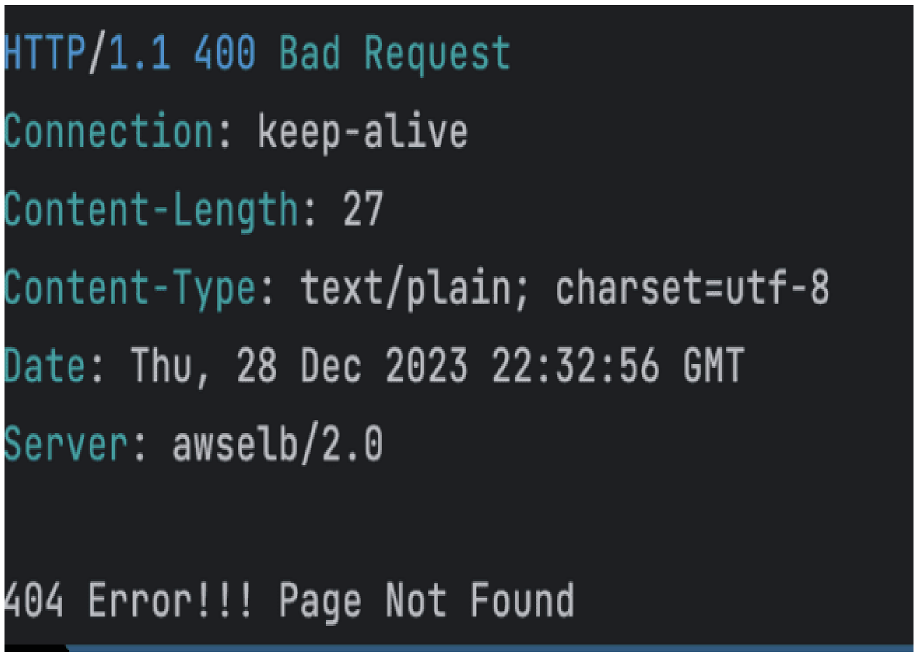

```
@message
END RequestId: c933d96f-03e8-4848-8ad1-7d37723f52ac
REPORT RequestId: c933d96f-03e8-4848-8ad1-7d37723f52ac Duration: 37.94 ms Billed Dura…
{"level":"INFO","location":"decorate:451","message":{"requestContext":{"elb":{"target…
{"level":"DEBUG","location":"lambda_handler:27","message":{"requestContext":{"elb":{"…
{"level":"INFO","location":"sample_get:16","message":"client_correlation_id: bb245","…
START RequestId: c933d96f-03e8-4848-8ad1-7d37723f52ac Version: $LATEST
END RequestId: 76bf2151-86ac-490a-a369-f432c5d54f15
REPORT RequestId: 76bf2151-86ac-490a-a369-f432c5d54f15 Duration: 2.08 ms Billed Durat…
{"level":"DEBUG","location":"lambda_handler:27","message":{"requestContext":{"elb":{"…
{"level":"INFO","location":"sample_get:16","message":"client_correlation_id: bb245","…
START RequestId: 76bf2151-86ac-490a-a369-f432c5d54f15 Version: $LATEST
{"level":"INFO","location":"decorate:451","message":{"requestContext":{"elb":{"target…
```

Figure 8. AWS Cloudwatch Log Insights and X-Ray Traces

I. Now, let us evaluate if the path-based rule behavior where we expect to receive 404 returned from ALB when a non-matching path pattern is received. The invocation pattern would look like

xh http://{ALB_ID}.{REGION}.elb.amazonaws.com/other Client-Correlation-Id:test404

As expected, we received a response with 404, as shown in Figure 8 below:



```
HTTP/1.1 400 Bad Request
Connection: keep-alive
Content-Length: 27
Content-Type: text/plain; charset=utf-8
Date: Thu, 28 Dec 2023 22:32:56 GMT
Server: awselb/2.0

404 Error!!! Page Not Found
```

Figure 9. A 404 HTTP status response from ALB

Rationale

API Gateway [6] offers comprehensive features for building and managing REST APIs, including robust authentication and authorization mechanisms. However, the cost associated with API Gateway can become an important factor for scenarios with exceptionally high throughput. API Gateway pricing is structured to account for various features. Moreover, in high-traffic situations, the cumulative charges can be significant.

In contrast, the Application Load Balancer (ALB) excels in efficiently handling large volumes of traffic and is generally more cost-effective for scenarios where sheer throughput is a priority. ALBs provide advanced load-balancing capabilities, making them a compelling choice for applications that require rapid and reliable traffic distribution.

Another notable advantage of employing ALB path-based routing is its applicability in breaking monolithic APIs into finer, granular ones. This approach can independently host each endpoint as an individual Lambda function. By configuring path-based routing or other conditions on the ALB listener, it becomes possible to direct traffic to the respective Lambda function that is handling a specific API endpoint.

This strategy aligns with the microservices architectural paradigm and offers a pathway for independent scaling of each Lambda function. This is particularly beneficial when specific API endpoints experience varying traffic levels or have distinct resource requirements. By decoupling individual endpoints, organizations can optimize resource allocation and ensure efficient scaling based on the unique demands of each component.

While AWS API Gateway provides a comprehensive set of API management capabilities, path-based dynamic routing[7] is a viable solution under specific circumstances. This approach addresses constraints related to AWS footprint that organizations might want to reduce. It offers the flexibility to architect finely granular APIs, providing organizations with a pragmatic means to enhance scalability and resource utilization in their serverless applications.

CONCLUSION

In conclusion, our exploration of building a serverless REST API with AWS SAM, ALB, and comprehensive testing methodologies underscores the power and flexibility of serverless computing in the AWS ecosystem. By leveraging AWS SAM, developers can streamline the development and deployment of serverless applications, benefiting from its intuitive template structure and seamless integration with AWS services.

The architectural pattern introduced, involving ALB path-based routing with Lambda behind, presents a forward-thinking approach to serverless API design. The emphasis on blocking unwanted requests at the ALB level enhances security and contributes to efficient resource utilization, aligning with best practices for serverless application development.

The local testing phase, facilitated by SAM CLI and emulation, empowers developers to validate their applications before deployment confidently. From dependency resolution to Lambda-compatible package creation, SAM CLI automates crucial steps, ensuring a smooth transition from local development to the AWS cloud.

The deployment process in the AWS testing phase is demystified with the `sam deploy --guided` command, offering a guided and configurable deployment experience. Thorough testing, including ALB path-based routing, allows developers to ensure the robustness and reliability of their serverless applications.

As we conclude this journey, it is evident that mastering serverless development in AWS involves a combination of innovative architectural patterns, streamlined development processes, and rigorous testing methodologies. By embracing the tools and best practices outlined in this paper, developers can confidently navigate the complexities of serverless computing, delivering scalable and resilient serverless applications in the AWS cloud.

REFERENCES

[1] AWS (n.d.). *Using the AWS SAM CLI*. AWS Serverless Application Model Developer Guide. https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/using-sam-cli.html

[2] AWS (2019, March 27). *New – Advanced Request Routing for AWS Application Load Balancers*. AWS News Blog. https://aws.amazon.com/blogs/aws/new-advanced-request-routing-for-aws-application-load-balancers/

[3] AWS (n.d.). *What are Amazon CloudWatch Logs?* Amazon CloudWatch Logs. https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html

[4] AWS (n.d.). *What is AWS X-Ray?* AWS X-Ray. https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html

[5] ducaale / xh (n.d.). *Xh*. GitHub. https://github.com/ducaale/xh

[6] AWS (n.d.). *What is Amazon API Gateway?* Amazon API Gateway Developer Guide. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[7] AWS (n.d.). *Working with routes for HTTP APIs*. Amazon API Gateway Developer Guide. https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-develop-routes.html

[8] AWS (n.d.). *AWS SAM*. Amazon ECR Public Gallery. https://gallery.ecr.aws/sam?operatingSystems=Linux

[9] Balasubrahmanya Balakrishna, Comparative Analysis of Amazon Api Gateway and Application Load Balancer for Serverless Architectures, Journal of Software Engineering (JSE), 2(1), 2024, pp. 1–6