



aws INNOVATE

MODERN APPLICATIONS EDITION

20 October, 2022

Building AWS Lambda functions with Java

Jan Tan
Principal Solutions Architect
Amazon Web Services

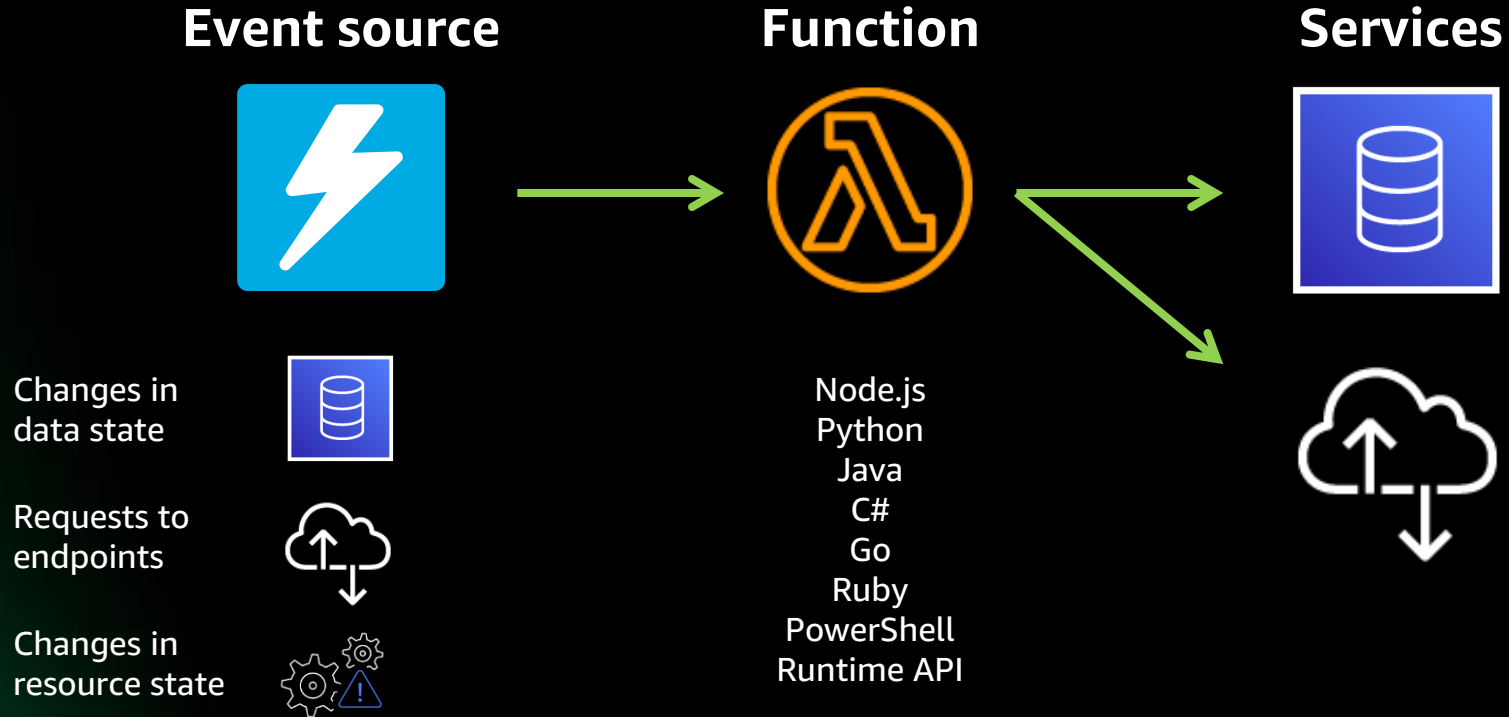
Salvian Reynaldi,
Software Engineer
Traveloka



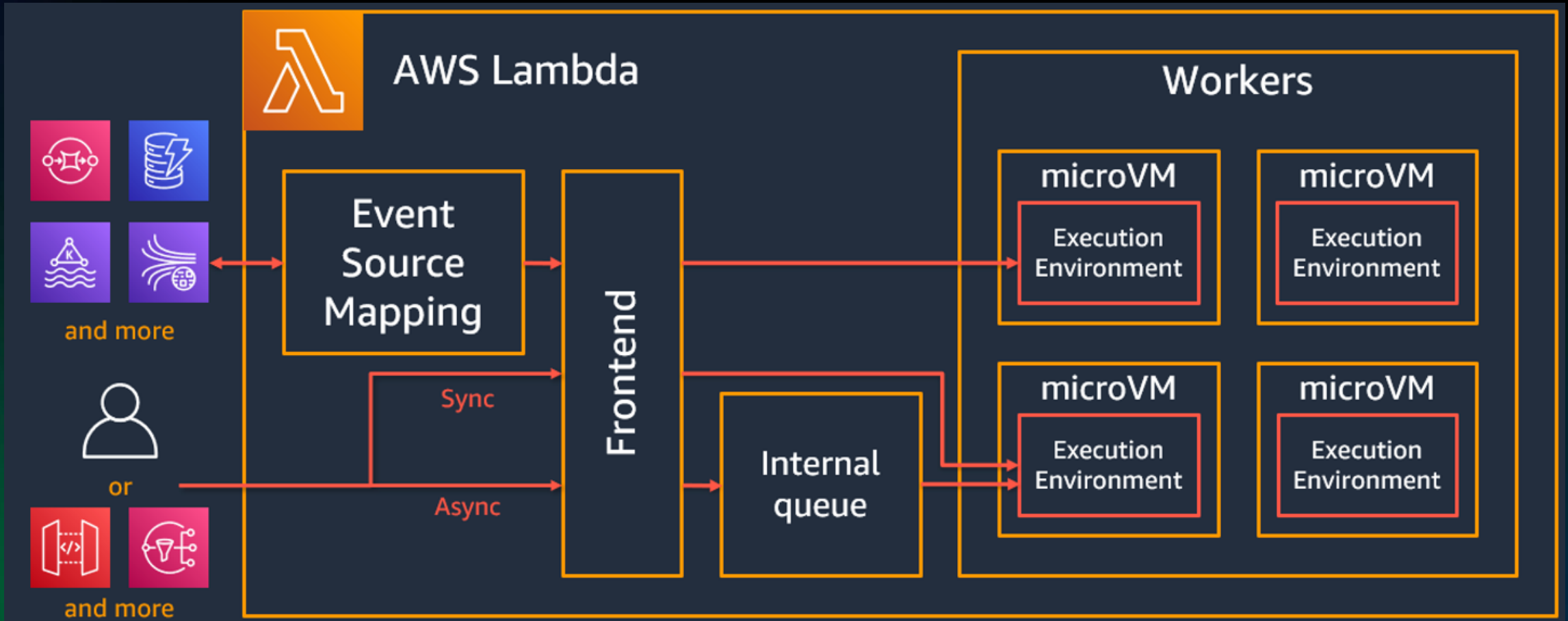
Agenda

- Overview of AWS Lambda
- Writing AWS Lambda functions with Java
- Local Development / Testing with AWS SAM
- Deployment
- Monitoring & Observability
- Best Practices
- Customer: Traveloka
- Summary
- Resources to get started

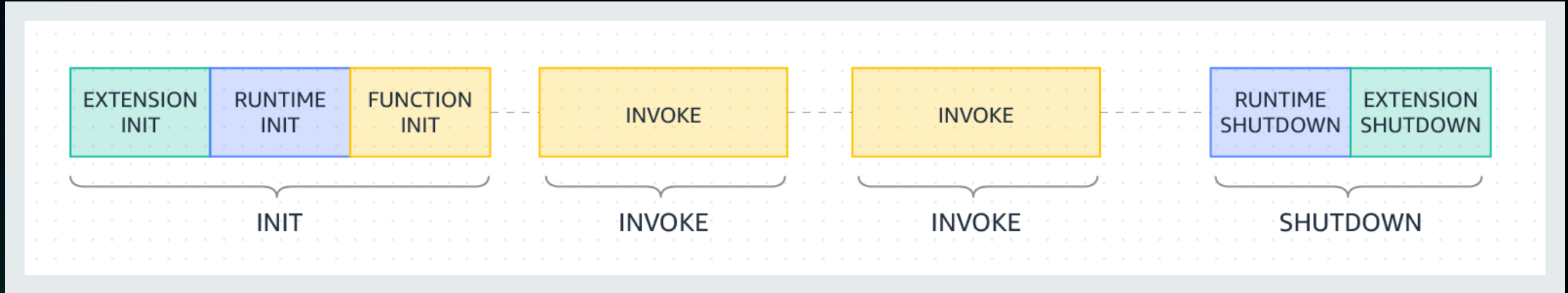
Overview of AWS Lambda



Under the Hood



Runtime Lifecycle



- Depending on the traffic pattern (and the amount of traffic), a Lambda function can be triggered hundreds/thousands of times a day which can translate to new environments being provisioned (triggering initialization code), and existing environments getting reused.
- **Init:** creates or unfreezes an execution environment. For new environments, run the function's initialization code (the code outside the main handler).
- **Invoke:** run the function handler code.
- **Shutdown:** this phase is triggered if the Lambda function does not receive any invocations for a period of time.

Writing AWS Lambda Functions with Java

A Simple Java Example

```
package com.vanilla.example;

public class HelloHandler {
    public HelloResponse hello(HelloRequest request) {
        String message = "hello "+request.getName();

        HelloResponse response = new HelloResponse();
        response.setMessage(message);

        return response;
    }
}
```


Dependencies

`com.amazonaws:aws-lambda-java-core`

Handler interfaces

Context object

`com.amazonaws:aws-lambda-java-events`

Input types for events from AWS services

`com.amazonaws:aws-lambda-java-log4j2`

Example using Handler Interface

```
public class UserTodoFunction implements RequestHandler<UserTodoRequest, List<TodoItem>> {  
    private final DynamoDbClient ddbClient;  
  
    public UserTodoFunction() {  
        ddbClient = DynamoDbClient.create();  
    }  
  
    @Override  
    public List<TodoItem> handleRequest(UserTodoRequest request, Context context) {  
        QueryResponse queryResponse = ddbClient.query(  
            QueryRequest  
                .builder()  
                .keyConditionExpression("user_id = :userId")  
                .expressionAttributeValues(Map.of(":userId", AttributeValue.builder().s(request.getUserId()).build()))  
                .tableName("serverless-todo")  
                .build());  
  
        return queryResponse  
            .items()  
            .stream()  
            .map(  
                row -> new TodoItem(row.get("todo").s(), Long.parseLong(row.get("created_at").n()))  
            ).collect(Collectors.toList());  
    }  
}
```

Input Type

Output Type

Handler: com.vanilla.example.UserTodoFunction

Example Implementation of Lambda Proxy

```
public class UserTodoFunction implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {  
    private final DynamoDbClient ddbClient;  
    private final Gson gson;  
  
    public UserTodoFunction() {  
        ddbClient = DynamoDbClient.create();  
        gson = new Gson();  
    }  
  
    @Override  
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent request, Context context) {  
        String userId = request.getQueryStringParameters().get("userId");  
        QueryResponse queryResponse = ddbClient.query(  
            QueryRequest  
                .builder()  
                .keyConditionExpression("user_id = :userId")  
                .expressionAttributeValues(Map.of(":userId", AttributeValue.builder().s(userId).build()))  
                .tableName("serverless-todo")  
                .build());  
  
        List<TodoItem> todoItems = queryResponse  
            .items()  
            .stream()  
            .map(  
                row -> new TodoItem(row.get("todo").s(), Long.parseLong(row.get("created_at").n()))  
            ).collect(Collectors.toList());  
  
        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent();  
  
        response.setStatusCode(200);  
        response.setBody(gson.toJson(todoItems));  
  
        return response;  
    }  
}
```

Input Type

Output Type

Get parameter from query string

Build response

Local Development and Testing with AWS SAM

What is AWS SAM

- Stands for AWS Serverless Application Model (AWS SAM)
- Components:
 - Template specification
 - CLI
- Extension of AWS CloudFormation
- Enables local debugging and testing

Getting Started with AWS SAM

- Install AWS SAM CLI: <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html>
- Run `sam init` to create the project
- Run `sam build` to build the Lambda functions and generate deployable asset
- Run `sam package` to upload assets into Amazon S3
- Run `sam deploy` to deploy the resources

Example SAM Template

```
Transform: AWS::Serverless-2016-10-31
Resources:
  TodoApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: dev
  ListUserTodoFunctionIamRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole'
      Policies:
        - PolicyName: inline0
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action: 'dynamodb:Query'
                Resource: '*'
  ListUserTodoFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: target/backend-0.0.1-SNAPSHOT.jar
      MemorySize: 512
      Timeout: 60
      Runtime: java11
      Handler: com.todo.UserTodoFunction
      Role: !GetAtt ListUserTodoFunctionIamRole.Arn
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /todos/list
            Method: get
            RestApiId:
              Ref: TodoApi
      Environment:
        Variables:
          ddb_table_name: serverless-todo
```

Macro hosted by AWS CloudFormation to transform SAM template to a compliant CloudFormation template

Serverless specific resources to simplify deployment

Access to the full suite of resources, intrinsic functions, and other template features that are available in AWS CloudFormation

Serverless focused integrations can be defined in a simplified manner

Equivalent AWS CloudFormation Template

```
ListUserTodoFunction:
  Type: AWS::Lambda::Function
  Properties:
    Code: !Ref CodeLocationJar
    Handler: com.todo.UserTodoFunction
    Role: !GetAtt ListUserTodoFunctionIamRole.Arn
    MemorySize: 512
    Runtime: java11
    Timeout: 60
    TracingConfig:
      Mode: Active
    Environment:
      Variables:
        ddb_table_name: serverless-todo

TodoRestApi:
  Type: AWS::ApiGateway::RestApi
  Properties:
    Mode: overwrite
    Parameters:
      endpointConfigurationTypes: REGIONAL

TodosResource:
  Type: AWS::ApiGateway::Resource
  Properties:
    RestApiId: !Ref TodoRestApi
    ParentId: !GetAtt TodoRestApi.RootResourceId
    PathPart: todos

TodosListResource:
  Type: AWS::ApiGateway::Resource
  Properties:
    RestApiId: !Ref TodoRestApi
    ParentId: !Ref TodosResource
    PathPart: list

TodosListGetMethod:
  Type: AWS::ApiGateway::Method
  Properties:
    RestApiId: !Ref TodoRestApi
    ResourceId: !Ref TodosListResource
    HttpMethod: GET
    Integration:
      Type: AWS_PROXY
      IntegrationHttpMethod: GET
      Uri: !Sub
        - arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${lambdaArn}/invocations
        - lambdaArn: !GetAtt ListUserTodoFunction.Arn

TodoRestApiDeployment:
  Type: AWS::ApiGateway::Deployment
  Properties:
    RestApiId: !Ref TodoRestApi
    StageName: dev
```

```
Transform: AWS::Serverless-2016-10-31
Resources:
  TodoApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: dev
  ListUserTodoFunctionIamRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole'
      Policies:
        - PolicyName: inline0
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action: 'dynamodb:Query'
                Resource: '*'
  ListUserTodoFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: target/backend-0.0.1-SNAPSHOT.jar
      MemorySize: 512
      Timeout: 60
      Runtime: java11
      Handler: com.todo.UserTodoFunction
      Role: !GetAtt ListUserTodoFunctionIamRole.Arn
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /todos/list
            Method: get
            RestApiId:
              Ref: TodoApi
      Environment:
        Variables:
          ddb_table_name: serverless-todo
```


Local Development and Testing (AWS Lambda Function)

Generate Amazon API Gateway event JSON using the following command:

```
sam local generate-event apigateway aws-proxy > local-event.json
```

AWS SAM Local supports a variety of different event types, run the following to see the full list: `sam local generate-event -h`

Run the following command to invoke your AWS Lambda function locally:

```
sam local invoke <FUNCTION_LOGICAL_ID> -e <EVENT_JSON_FILE>  
-t <TEMPLATE_FILE>
```

Example of Local AWS Lambda Invocation

```
+ > sam local invoke ListUserTodoFunction -e local-event.json -t sam-template.yaml
Invoking com.todo.UserTodoFunction (java11)
Decompressing /Users/jantan/Work/demo2/target/backend-0.0.1-SNAPSHOT.jar
Skip pulling image and use local one: public.ecr.aws/sam/emulation-java11:rapid-1.40.1-x86_64.

Mounting /private/var/folders/vx/rt2v24x16nq3ql7b6nlhystw0000gr/T/tmpjt0mgarl as /var/task:ro,delegated inside runtime container
START RequestId: def1e309-5242-4bff-a458-2d1e0fbc86f9 Version: $LATEST
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{"statusCode":200,"body":[{"todo":{"Buy milk"},"createdAt":1646211521260}, {"todo":{"Go to gym"},"createdAt":1646620598539}]}END RequestId: def1e309-5242-4bff-a458-2d1e0fbc86f9
REPORT RequestId: def1e309-5242-4bff-a458-2d1e0fbc86f9 Init Duration: 0.45 ms Duration: 5577.16 ms Billed Duration: 5578 ms Memory Size: 512 MB Max Memory Used: 512 MB
```

Command to invoke
function locally with the
given event payload

Function response payload

Local Development and Testing (Local API Gateway)

```
❯ sam local start-api --template-file sam-template.yaml
Mounting ListUserTodoFunction at http://127.0.0.1:3000/todos/list [GET]
You can now browse to the above endpoints to invoke your functions. You do not need to restart/reload SAM CLI while working on your functions, changes will be reflected instantly/automatically. You only need to restart SAM CLI if you update your AWS SAM template
2022-03-10 09:32:46 * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
Invoking com.todo.UserTodoFunction (java11)
Decompressing /Users/jantan/Work/demo2/target/backend-0.0.1-SNAPSHOT.jar
Image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-java11
Building image..... Requires Docker to be installed locally
Skip pulling image and use local one: public.ecr.aws/sam/emulation-java11:rapid-1.40.1-x86_64.

Mounting /private/var/folders/vx/rt2v24x16nq3ql7b6nlhystw0000gr/T/tmpuhkbc3x3 as /var/task:ro,delegated inside runtime container
START RequestId: a9bb48b3-df77-4a4e-8baa-cd726fd6b512 Version: $LATEST
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
END RequestId: a9bb48b3-df77-4a4e-8baa-cd726fd6b512
REPORT RequestId: a9bb48b3-df77-4a4e-8baa-cd726fd6b512 Init Duration: 0.25 ms Duration: 7892.33 ms Billed Duration: 7893 ms Memory Size: 512 MB Max Memory Used: 512 MB
No Content-Type given. Defaulting to 'application/json'.
2022-03-10 09:33:45 127.0.0.1 - - [10/Mar/2022 09:33:45] "GET /todos/list?userId=530a8a2b-02a0-4091-a294-7dbd98928135 HTTP/1.1" 200 -
```

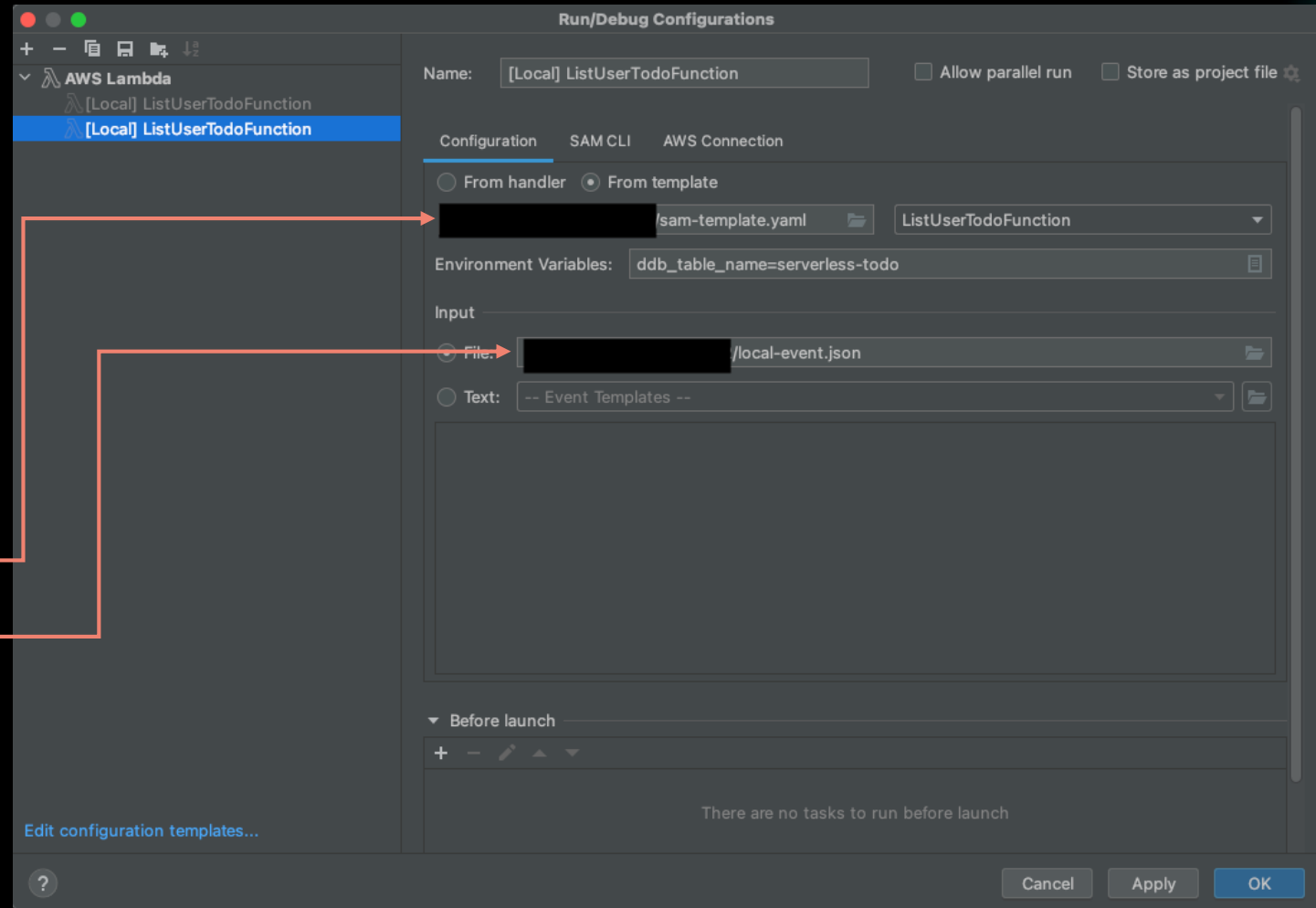
To test the API and the associated Lambda function defined in the example template file, run the following command:

```
sam local start-api --template-file <TEMPLATE_FILE_PATH>
```

The API is accessible via the browser/curl using the URL (using default port): `http://localhost:3000`

Debug using IDE

- Install AWS Toolkit plugin
- Double check that SAM is configured properly in the AWS Toolkit plugin
- Right click on function handler Java code and initiate the Debug process
- Provide the following:
 - Template location
 - Event payload location



Example Debugging Session w/ IntelliJ

```
@Override
public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent request, Context context) {
    String userId = request.getQueryStringParameters().get("userId");
    QueryResponse queryResponse = ddbClient.query(
        QueryRequest
            .builder()
            .keyConditionExpression("user_id = :userId")
            .expressionAttributeValues(Map.of( k1: ":userId", AttributeValue.builder().s(userId).build()))
            .tableName(ddbTableName)
            .build());

    List<TodoItem> todoItems = queryResponse
        .items()
        .stream()
        .map(
```

Debug: [Local] UserTodoFunction x [Local] UserTodoFunction x

Debugger Console

Frames

- main@1 in g...ain: RUNNING
- handleRequest:32, UserTodoFunction (com.todo)
- handleRequest:18, UserTodoFunction (com.todo)
- handleRequest:199, EventHandlerLoader\$PojoHan
- call:899, EventHandlerLoader\$2 (lambdainternal)
- startRuntime:262, AWSLambda (lambdainternal)
- startRuntime:199, AWSLambda (lambdainternal)
- main:193, AWSLambda (lambdainternal)

Variables

- this = (UserTodoFunction@2819)
- request = (APIGatewayProxyRequestEvent@2820) "{resource: /(proxy+),path: /path/to/resource,httpMethod: POST,headers: {Accept=text/ht
- version = null
- resource = "/(proxy+)"
- path = "/path/to/resource"
- httpMethod = "POST"
- headers = (LinkedHashMap@2830)
- multiValueHeaders = (LinkedHashMap@2831)
- queryStringParameters = (LinkedHashMap@2832)
- multiValueQueryStringParameters = (LinkedHashMap@2833)
- pathParameters = (LinkedHashMap@2834)
- stageVariables = (LinkedHashMap@2835)
- requestContext = (APIGatewayProxyRequestEvent\$ProxyRequestContext@2836) "{accountId: 123456789012,resourceId: 123456,stage: p
- body = "eyJ0ZXN0IjoIYm9keSJ9"
- isBase64Encoded = (Boolean@2838)
- context = (LambdaContext@2821)
- ddbClient = (DefaultDynamoDbClient@2822)
- ddbTableName = "serverless-todo"

Deployment

Packaging

Supports either Zip or Jar format

- Gradle – using Zip build type

- Maven – using Maven Shade plugin

Supports deployment as a container image

- Optional AWS base image for Lambda

- Run and test image locally

- Bring your own base image

Example Dockerfile

```
FROM public.ecr.aws/lambda/java:11

COPY target/classes ${LAMBDA_TASK_ROOT}
COPY target/dependency/* ${LAMBDA_TASK_ROOT}/lib/

CMD ["com.todo.UserTodoFunction"]
```


Example Local Image Testing

Run locally using:

```
docker run -p 9000:8080 -e AWS_REGION=<REGION> -e  
AWS_ACCESS_KEY_ID=<ACCESS_KEY> AWS_SECRET_ACCESS_KEY=<SECRET_KEY> -e  
ddb_table_name=serverless-todo user-todo-function
```

Invoke locally using:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations"  
-d '@local-event.json'
```

Other Deployment Options

- Directly using the AWS Lambda console
- AWS CLI
- AWS CodeDeploy
- Infrastructure as Code
 - AWS Cloud Development Kit (AWS CDK)
 - AWS CloudFormation
 - Terraform
 - Pulumi

Monitoring & Observability

Key concepts

Metrics – out of the box metrics such as number of invocations, duration, error count and success rate, throttles, etc.

Logs – automatically sent to a central location (Amazon CloudWatch) by default.

Tracing – following a single request end-to-end throughout a system composed of multiple microservices / AWS services.

Using Amazon CloudWatch Logs Insights

- Search and aggregate data across thousands of individual log files
- Search and analyze log data using Amazon CloudWatch Logs Insights query syntax
- Functions that use structured logging can take advantage of the query syntax to create targeted queries

The screenshot shows the Amazon CloudWatch Logs Insights console. At the top, there's a dropdown for "Select log group(s)" and a time range selector set to "5m". Below this, a query is entered in the text area:

```
1 fields @message
2 filter @message like /INFO/
3 parse @message."UploadedBytes=" as bytes
4 stats min(bytes), avg(bytes), max(bytes) by bin (1m)
```

 Buttons for "Run query", "Save", "Actions", and "History" are visible. Below the query area, there's a "Visualizations" tab. It shows "Showing 38 of 3,654 records matched" and "14,620 records (1.7 MB) scanned in 3.4s @ 4,302 records/s (501.3 kB/s)". A table displays the results with columns: #, bin (1m), min(bytes), avg(bytes), and max(bytes). The table contains 5 rows of data.

#	bin (1m)	min(bytes)	avg(bytes)	max(bytes)
1	2020-10-21T17:34:00.000-04:00	115302	3637261.4444	14899397
2	2020-10-21T17:33:00.000-04:00	74862	5143198.5413	14995087
3	2020-10-21T17:32:00.000-04:00	178115	4974870.1545	14920368
4	2020-10-21T17:31:00.000-04:00	25370	4659734.9381	14838911
5	2020-10-21T17:30:00.000-04:00	14032	4852778.6491	14993216

The screenshot shows the Amazon CloudWatch Logs Insights console with a sidebar on the left and a "Discovered fields" panel on the right. The query in the text area is:

```
1 fields @message
2 filter @message like /INFO/
3 filter uploadedBytes > 1000000
4 filter uploadTimeMS > 1000
5 filter invocation != 1
```

 The "Discovered fields" panel lists various fields and their percentages: @ingestionTime (100%), @logStream (100%), @message (100%), @timestamp (100%), @requestId (99%), @type (74%), uploadTime (25%), @billedDuration (24%), @duration (24%), @maxMemoryUsed (24%), @memorySize (24%), invocation (24%), uploadedBytes (24%), uploadTimeMS (24%), and @initDuration (-). The main panel shows "Showing 1000 of 5,299 records matched" and "33,008 records (4.1 MB) scanned in 3.4s @ 9,722 records/s (1.2 MB/s)". A table displays the results with columns: #, @message, and the log data.

#	@message
1	2020-10-22T12:56:45.853Z 609f62f7-cff0-4664-99fd-ef009b42c18a INFO { uploadedBytes: 6267459, invocation: 375, uploadTimeMS: 2082 }
2	2020-10-22T12:56:44.515Z 5294a304-e9d9-4d8b-9371-08a7011f07e4 INFO { uploadedBytes: 4084813, invocation: 372, uploadTimeMS: 2867 }
3	2020-10-22T12:56:43.627Z f5a4d2ed-8495-4bca-8107-ee4f427523e INFO { uploadedBytes: 1850753, invocation: 370, uploadTimeMS: 1229 }
4	2020-10-22T12:56:42.744Z 49064079-475a-4534-a7c9-4ac1f7975e00 INFO { uploadedBytes: 2951740, invocation: 368, uploadTimeMS: 1063 }
5	2020-10-22T12:56:42.069Z e5fb0924-e142-440b-ad33-f5e590a6f66c INFO { uploadedBytes: 2197056, invocation: 367, uploadTimeMS: 2404 }
6	2020-10-22T12:56:41.968Z 5e72a917-97bb-473c-b230-c56c3fa00090 INFO { uploadedBytes: 1081188, invocation: 366, uploadTimeMS: 1226 }

Tracing Request using AWS X-Ray

```
Resources:
  TodoApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: dev
      TracingEnabled: true
  ListUserTodoFunctionIamRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
        - arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess
      Policies:
        - PolicyName: inline0
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: dynamodb:Query
                Resource: '*'
  ListUserTodoFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ListUserTodoFunction
      MemorySize: 512
      Timeout: 60
      Runtime: java11
      Handler: com.todo.UserTodoFunction
      Role:
        Fn::GetAtt:
          - ListUserTodoFunctionIamRole
        - Arn
      Tracing: Active
      Events:
        ApiEvent:
          Type: Api
```

Additional permission

No node selected

Select a node to see its details

[View logs](#) [View traces](#) [Analyze traces](#) [View dashboard](#)

Trace Summary

Method	Response Code	Duration	Age
GET	200	8.6s	a minute (2022-03-11 12:26:11)

Segments Timeline

	Segment status	Response code	Duration	
▼ SamTodoJava/dev AWS::ApiGateway::Stage				
SamTodoJava/dev	OK	200	8.55s	GE...
Lambda	OK	200	8.55s	Inv...
▼ SamTodoJava-ListUserTodoFunction-wgdkvWEyPRBL AWS::Lambda				
SamTodoJava-ListUserTo...	OK	200	8.51s	
▼ SamTodoJava-ListUserTodoFunction-wgdkvWEyPRBL AWS::Lambda::Function				
SamTodoJava-ListUserTo...	OK	-	5.60s	
Initialization	OK	-	2.70s	
Invocation	OK	-	5.59s	
DynamoDB	OK	200	4.32s	Query: s...
Overhead	OK	-	0ms	
▼ DynamoDB AWS::DynamoDB::Table				
DynamoDB	OK	200	4.32s	Query: s...

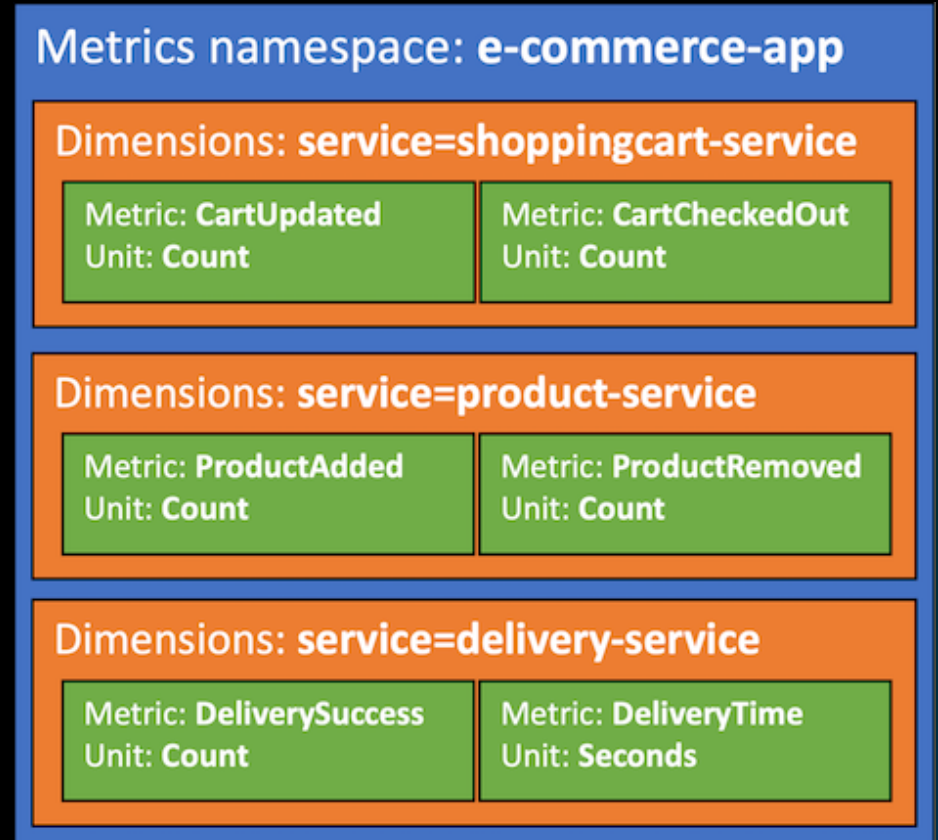


AWS Lambda Power tools for Java

- A suite of utilities for AWS Lambda Functions that makes the following easier:
 - Tracing with AWS X-Ray
 - Structured Logging
 - Creating custom metrics asynchronously (*using CloudWatch Embedded Metric Format*)
- Installation (via either Maven or Gradle) using the following dependencies:
 - powertools-tracing
 - powertools-logging
 - powertools-metrics
- See <https://awslabs.github.io/aws-lambda-powertools-java/> for more information

Async Custom Metrics using AWS Lambda Power tools for Java

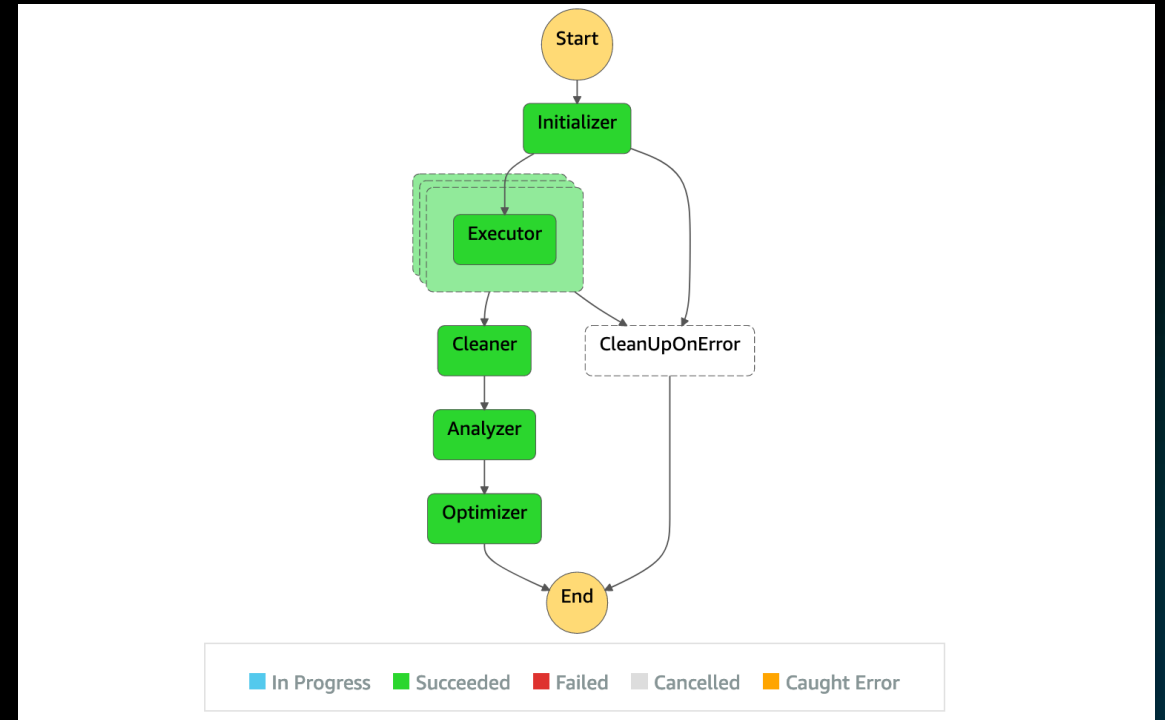
- Easily generate custom metrics from AWS Lambda functions without requiring custom batching code, making blocking network requests or relying on third-party software.
- Using the Embedded Metric Format, you will be able to visualize and alarm on custom metrics, but also retain the original, detailed and high-cardinality context which is queryable using Amazon CloudWatch Logs Insights.



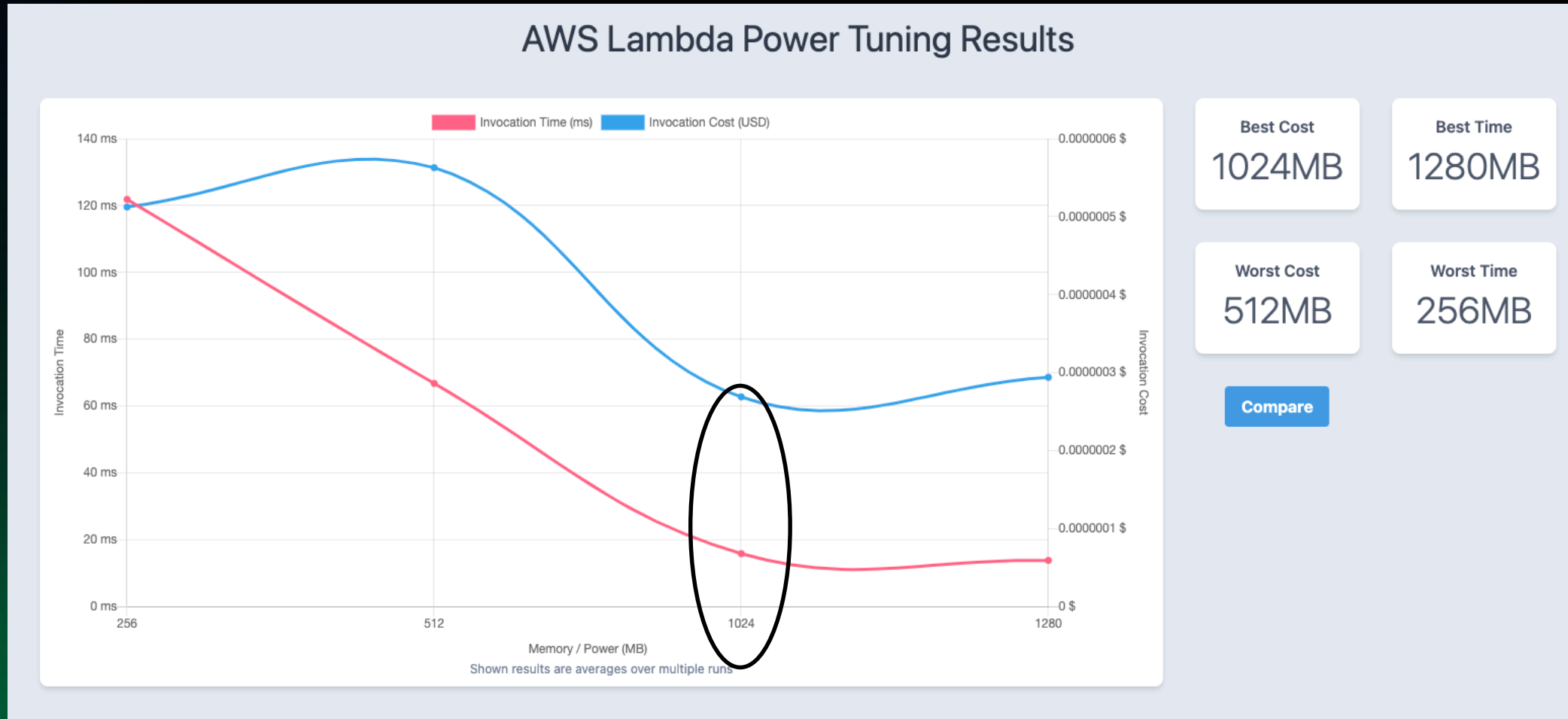
Best Practices

Balancing Act (Right Sizing)

- Finding the right balance between cost and execution time
- Increasing AWS Lambda function memory configuration also scales the allocated CPU
- [AWS Lambda Power Tuning](#) can help optimize your AWS Lambda functions for cost and/or performance in a data-driven way



Example Output from AWS Lambda Power Tuning



Improving Cold Start

Objects that are expected to be reused across multiple invocations should move its respective initialization code to either the static initializer or constructor.

When using the AWS SDK, pre-configure specific items during client initialization. For example: region, the credentials provider, and endpoint.

```
public class UserTodoFunction implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    private final DynamoDbClient ddbClient;
    private final Gson gson;
    private final String ddbTableName;

    public UserTodoFunction() throws URISyntaxException {
        ddbClient = DynamoDbClient.builder()
            .httpClient(URLConnectionHttpClient.create())
            .endpointOverride(new URI( str: "https://dynamodb.ap-southeast-1.amazonaws.com"))
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .region(Region.AP_SOUTHEAST_1)
            .build();
        gson = new Gson();
        ddbTableName = System.getenv( name: "ddb_table_name");
    }

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent request, Context context) {
        String userId = request.getQueryStringParameters().get("userId");
        QueryResponse queryResponse = ddbClient.query(
            QueryRequest
                .builder()
                .keyConditionExpression("user_id = :userId")
                .expressionAttributeValues(Map.of( k1: ":userId", AttributeValue.builder().s(userId).build()))
                .tableName(ddbTableName)
                .build());

        List<TodoItem> todoItems = queryResponse
            .items()
            .stream()
            .map(
                row -> new TodoItem(row.get("todo").s(), Long.parseLong(row.get("created_at").n()))
            ).collect(Collectors.toList());

        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent();

        response.setStatusCode(200);
        response.setBody(gson.toJson(todoItems));

        return response;
    }
}
```

Improving Cold Start Using Tiered Compilation

- From version 8, the JDK introduced 2 types of JIT compiler:
 - C1 – client side, enable short feedback loops
 - C2 – server side, achieve higher performance after profiling
- Use language-specific environment variables to set Tiered Compilation level to 1. Example:
 - `JAVA_TOOL_OPTIONS=-XX:+TieredCompilation
-XX:TieredStopAtLevel=1`

Level 4 - C2
Level 3 - C1 w/ full profiling
Level 2 - C1 w/ basic profiling
Level 1 - C1 w/o profiling
Level 0 - Interpreter

More Optimizations

- Avoid using reflections as much as possible
- Use lightweight dependencies
- Use compile-time dependency injection frameworks
- Right sizing your functions
- Explore microframeworks like Micronaut or Quarkus
- Explore using Native Image technology like GraalVM
- Use provisioned concurrency to warm-up ahead of time
- Remove unnecessary jar dependencies

Customer: Traveloka

Summary

- AWS provides tooling to enable local Lambda function development, testing, and debugging.
- There's a variety of packaging and deployment options that are available and they can be integrated with your existing CI/CD pipeline to automate the end to end process.
- Tracing can be used to help pinpoint problem areas (e.g. higher latencies). Especially critical for deep/complicated multi-service requests.
- Modern microframeworks and technologies like Micronaut, Quarkus, and GraalVM makes working with Java in the serverless world easier and faster.

Example codes / Demos

- Serverless using GraalVM: <https://github.com/aws-samples/serverless-graalvm-demo>
- Serverless using frameworks like Micronaut, Quarkus, and Spring Boot: <https://github.com/aws-samples/serverless-java-frameworks-samples>
- Java on AWS Lambda workshop: <https://catalog.workshops.aws/java-on-aws-lambda/en-US>

Visit the Modern Applications resource hub

Dive deeper with these resources to help you develop an effective plan for your modernization journey.

- Build modern applications on AWS
- Business value of cloud modernization
- An introduction to event-driven architectures
- Accelerate full-stack web and mobile app development
- Determining the total cost of ownership: Comparing serverless and server-based technologies
- Building event-driven architectures with AWS
- Continuous learning, continuous modernization



<https://tinyurl.com/modern-apps-aws>

Visit resource hub

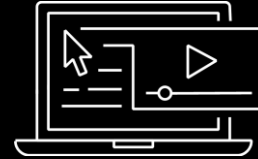


AWS Training and Certification

Get started with Free Digital Training for you and your team today



Achieve key milestones and plan your next steps with the AWS Modern Application skills training



Access 500+ free digital courses with [AWS Skill Builder](#)



Earn an industry-recognized credential:
[AWS Certified Developer – Associate](#)
[AWS Certified DevOps – Professional](#)



Create a self-paced learning roadmap
[AWS ramp-up guide - Developer](#)
[AWS ramp-up guide - DevOps](#)

Thank you for attending AWS Innovate Modern Applications Edition

We hope you found it interesting! A kind reminder to **complete the survey**.
Let us know what you thought of today's event and how we can improve the event
experience for you in the future.



aws-apj-marketing@amazon.com



twitter.com/AWSCloud



facebook.com/AmazonWebServices



youtube.com/user/AmazonWebServices



slideshare.net/AmazonWebServices



twitch.tv/aws

Thank you!