INNOVATE

MODERN APPLICATIONS EDITION

20 October, 2022

Building AWS Lambda functions with Java

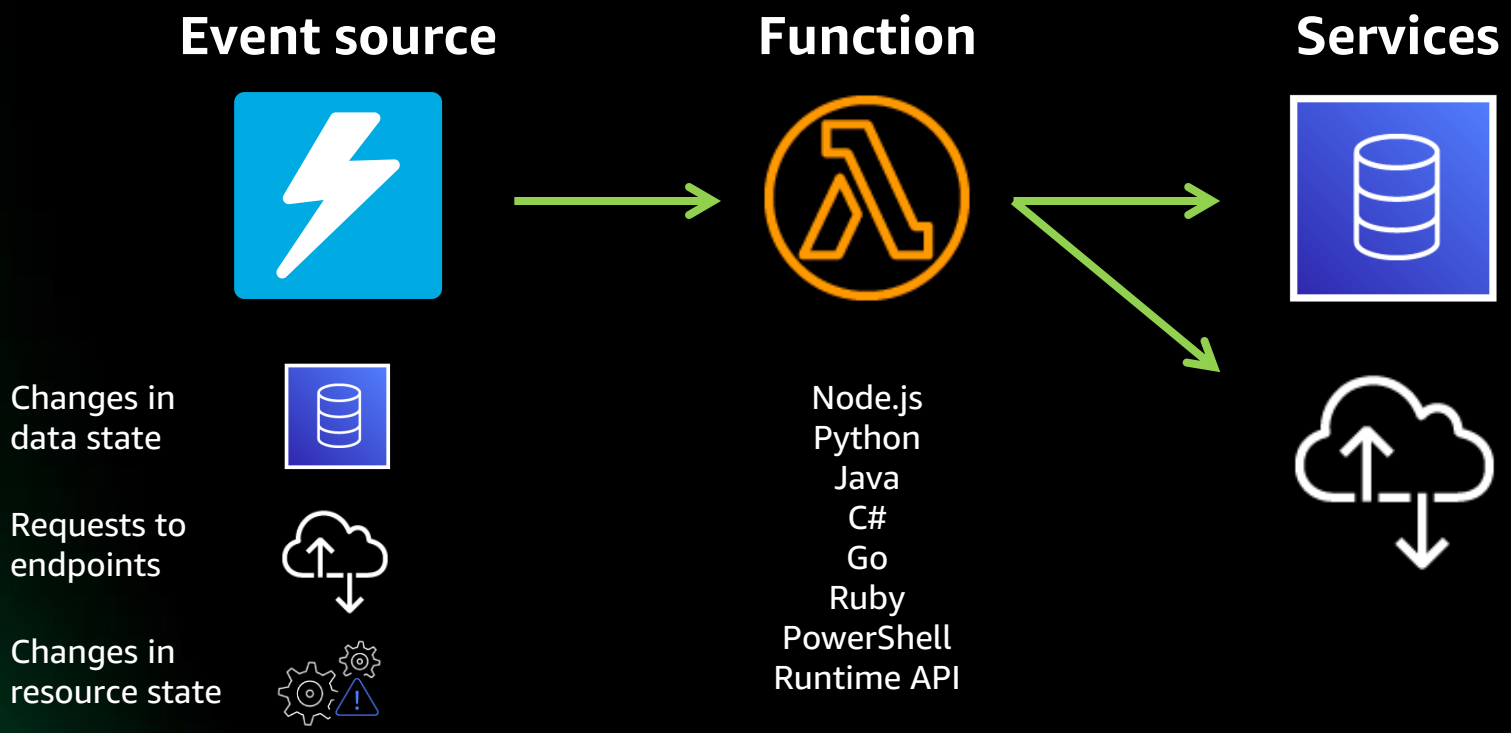
Jan Tan
Principal Solutions Architect
Amazon Web Services

Salvian Reynaldi,
Software Engineer
Traveloka

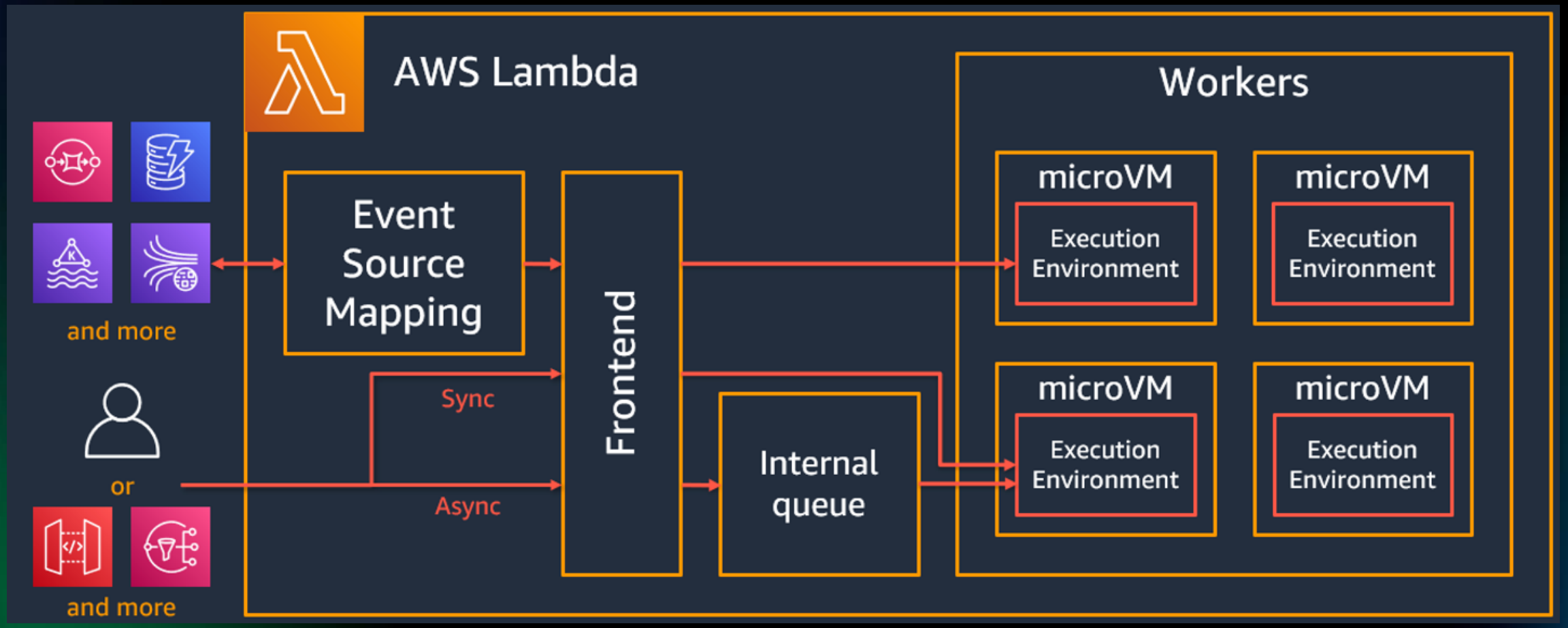
Agenda

- 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

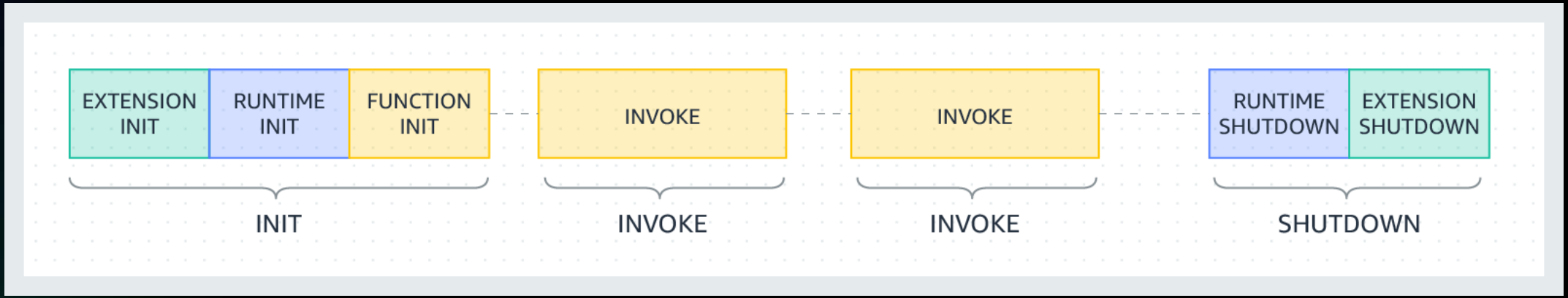
Overview of AWS Lambda



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

```java
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

```java
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

```java
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

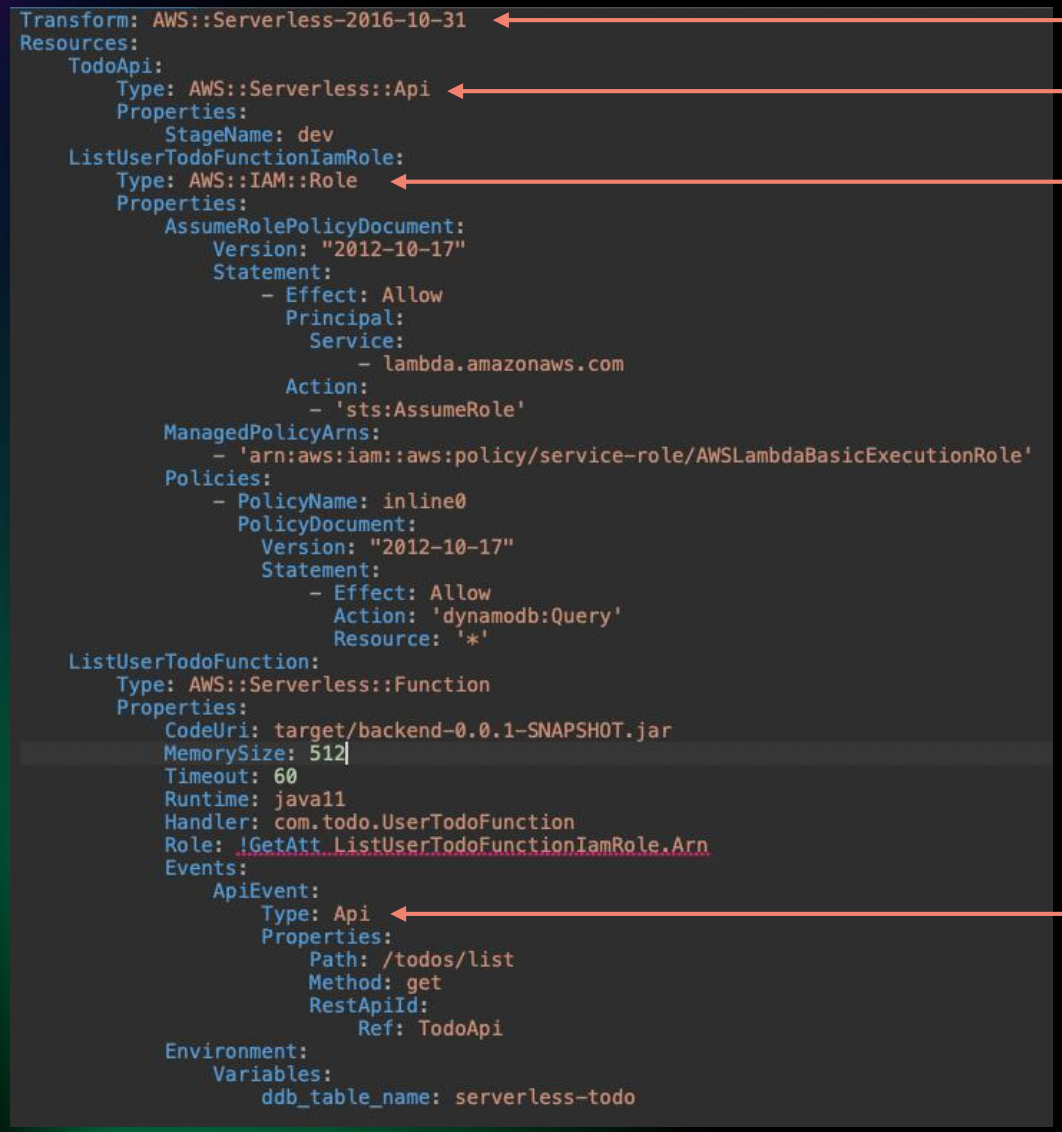
What is AWS SAM

- 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



```yaml
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

```yaml
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
```

```yaml
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 up
date 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.....
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 -
```

Requires Docker to be installed locally

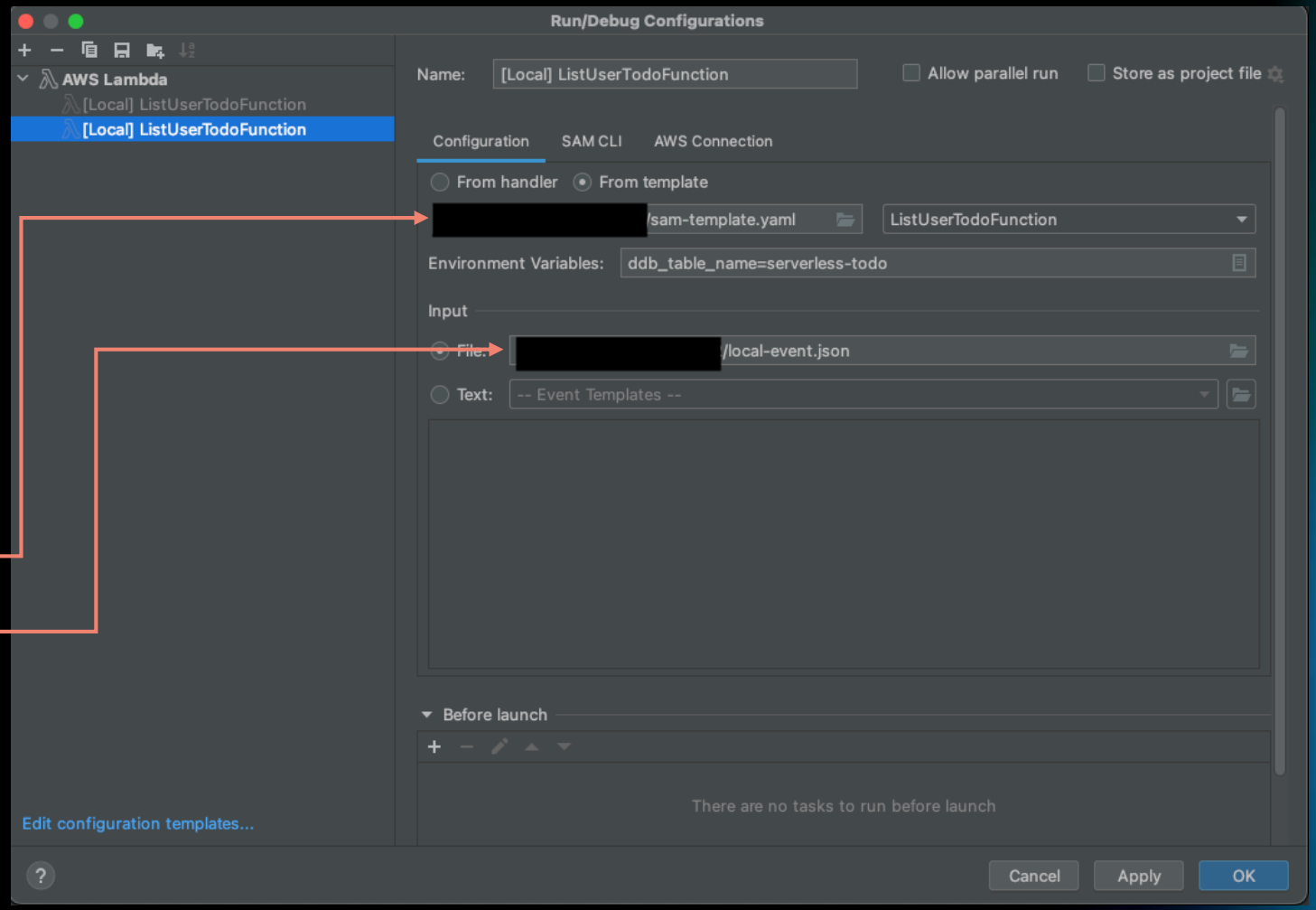
To test the API and the associated Lambda functioned defined in the example template file, run the following command:

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

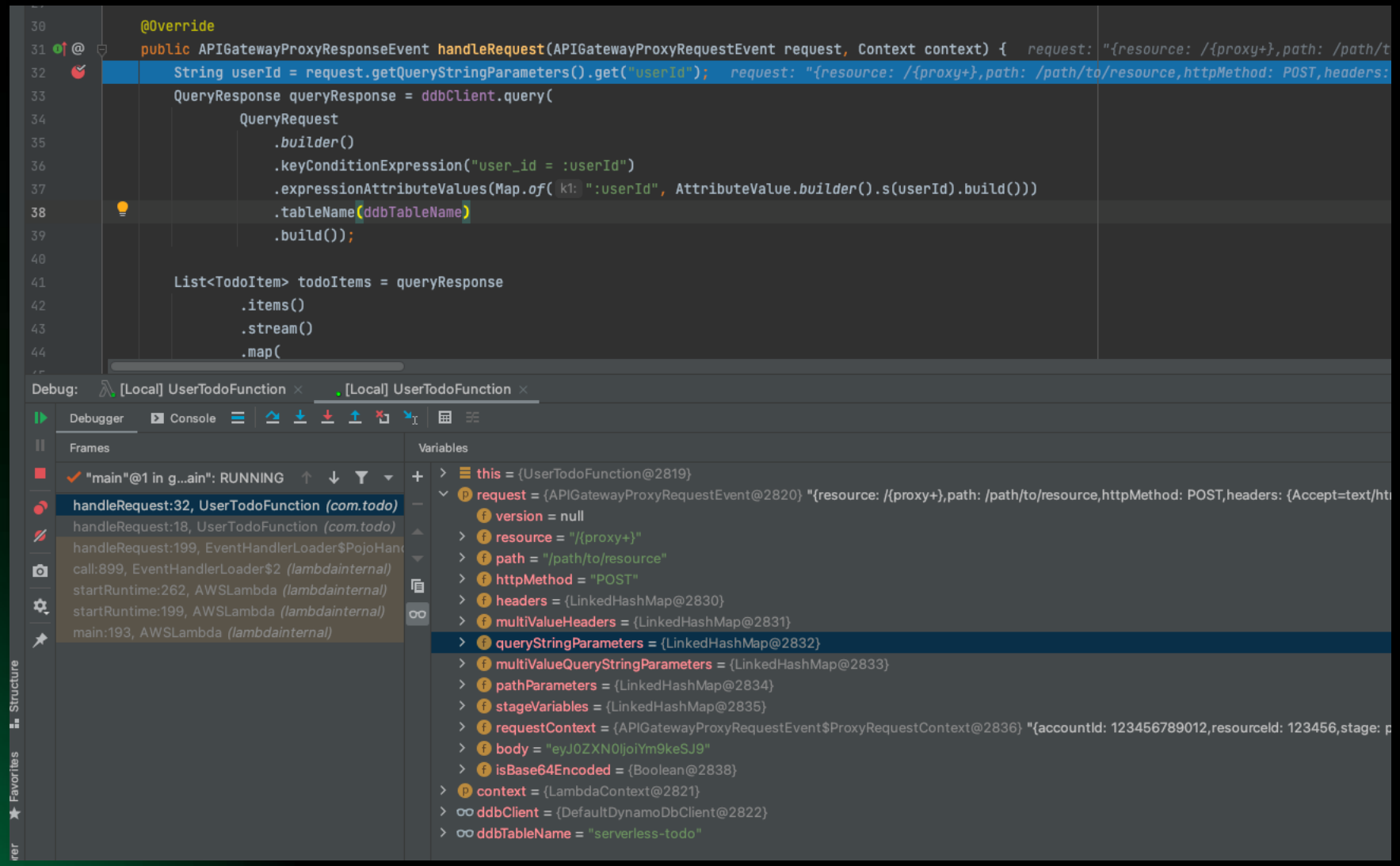
The API is accessible via the browser/curl using the URL (using default port): `http://localhost:3000`

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



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

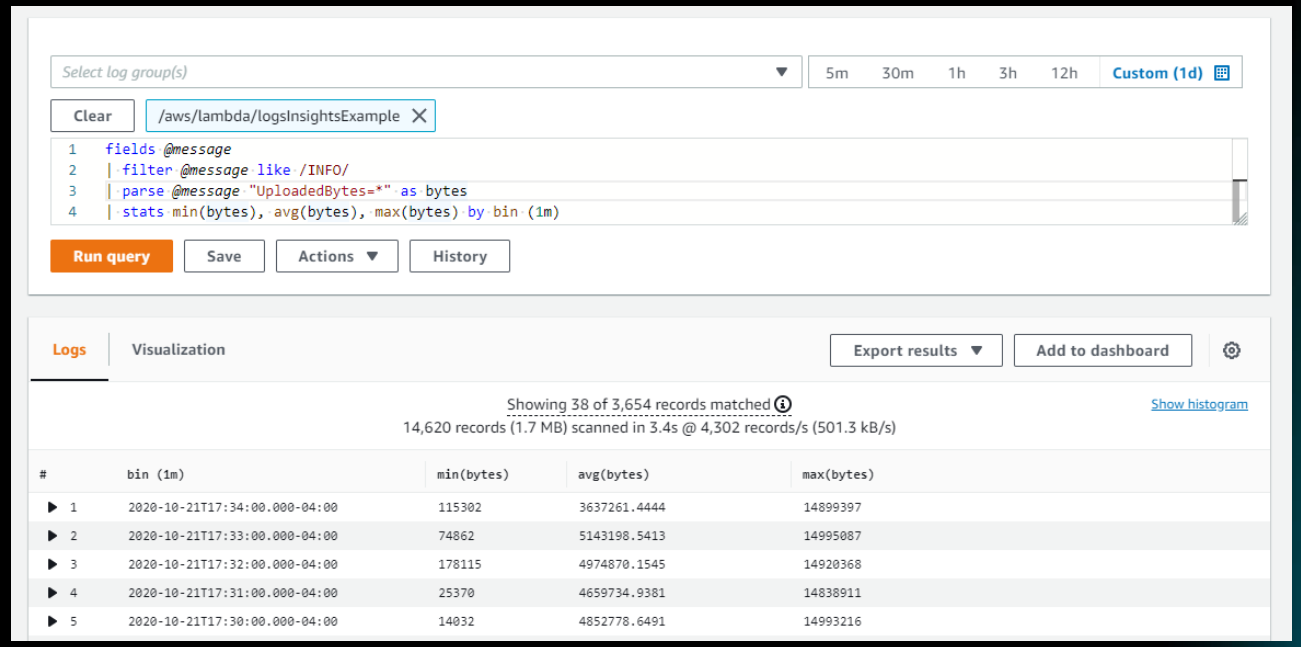
Metrics – out of the box metrics such as number of invocations, duration, error count and success rate, throttles, etc.

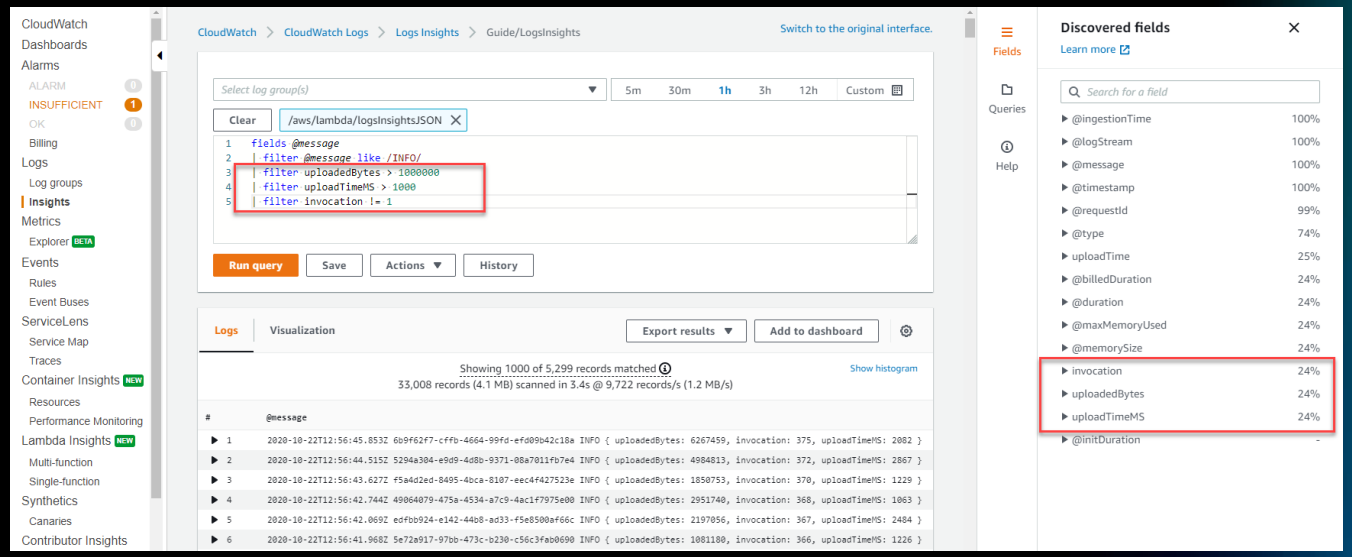
Logs – automatically sent to a central location (Amazon CloudWatch) by default.

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

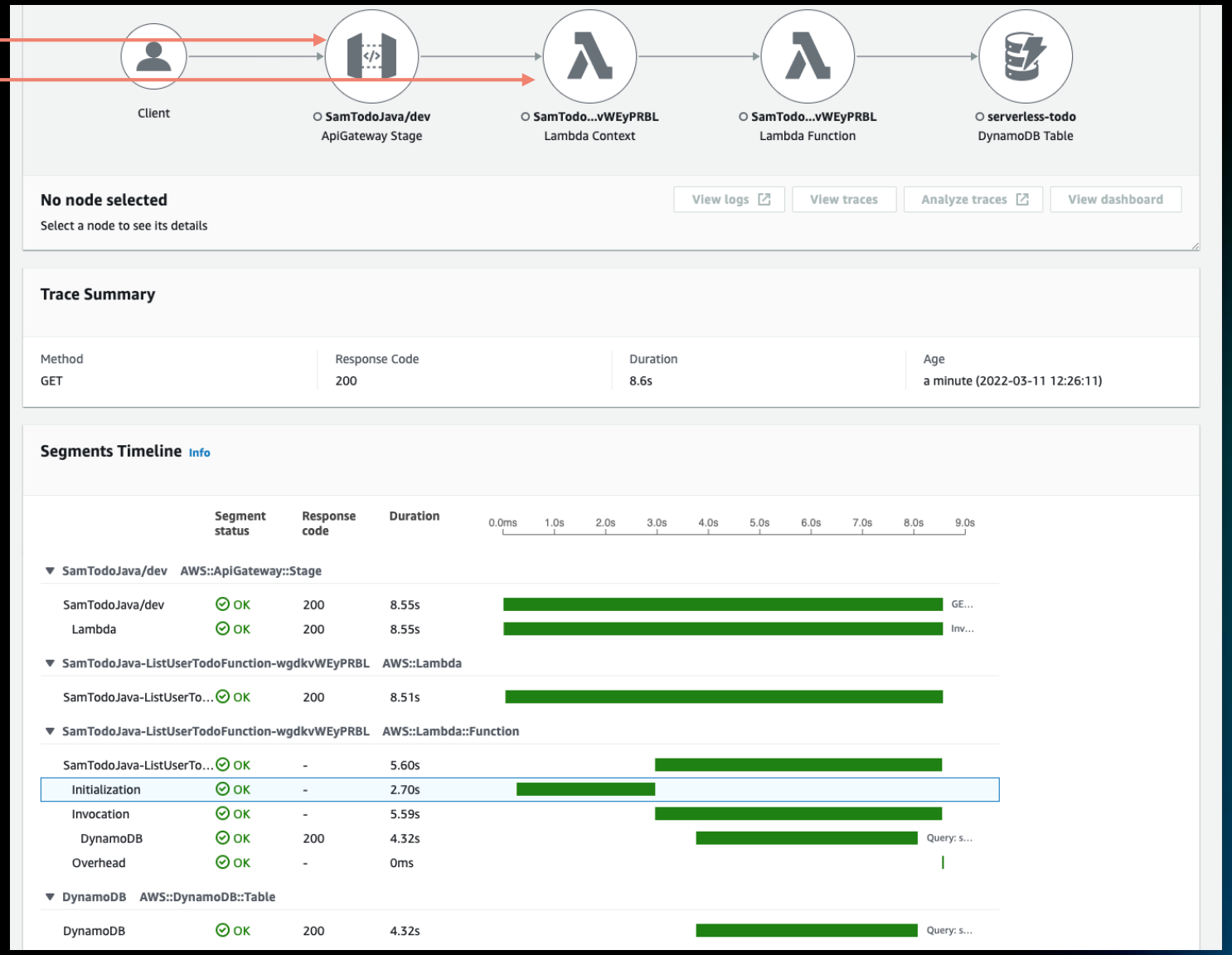




Tracing Request using AWS X-Ray

```yaml
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



AWS Lambda Power tools for Java

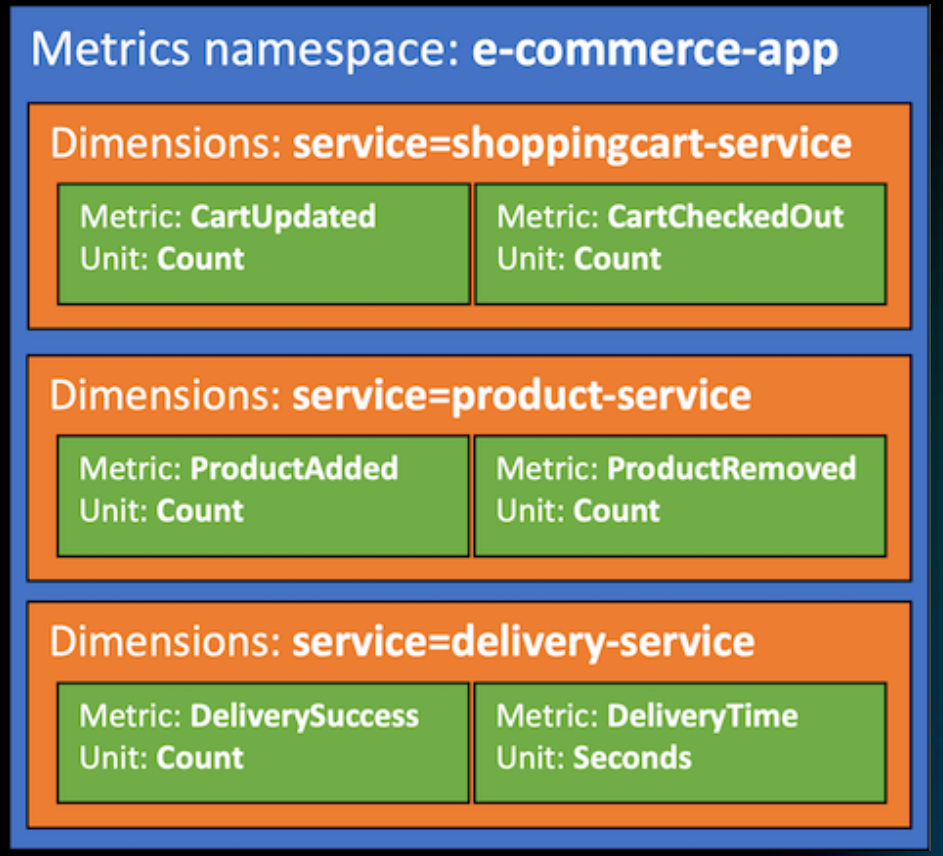
- A suite of utilities for AWS Lambda Functions that makes the following easier:
  - Tracing with AWS X-Ray
  - Structured Logging
  - Creating custom metrics asynchronously (*using CloudWatch Embedded Metric Format*)

- 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

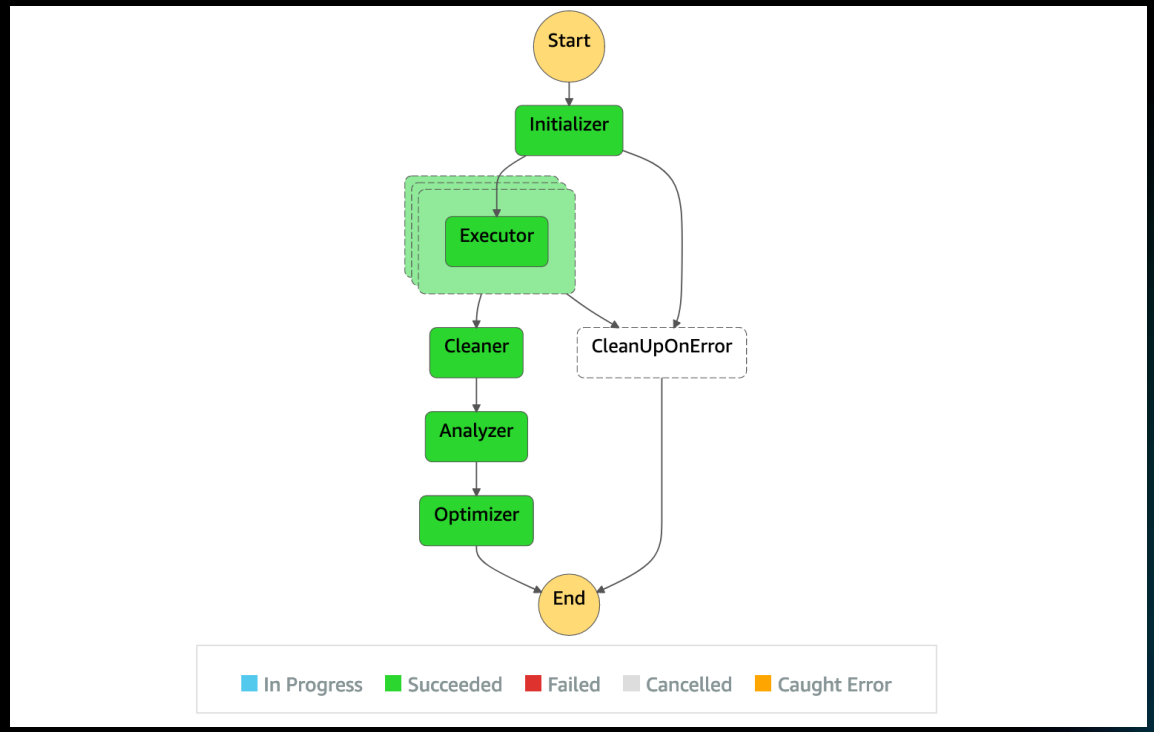
- Easily generate custom metrics from AWS Lambda functions without requiring custom batching code, making blocking network requests or relying on third-party software.
- Using the Embedded Metric Format, you will be able to visualize and alarm on custom metrics, but also retain the original, detailed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



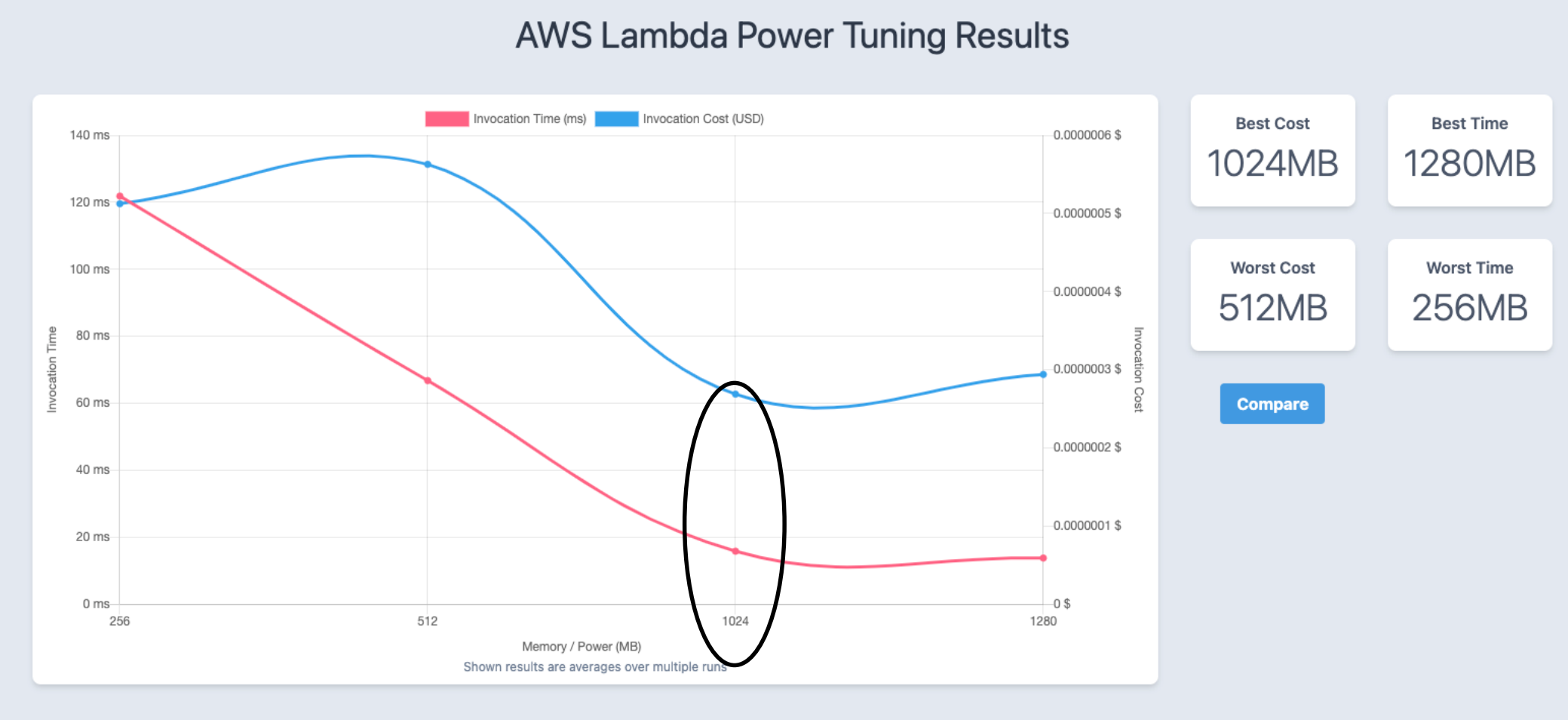
Best Practices

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

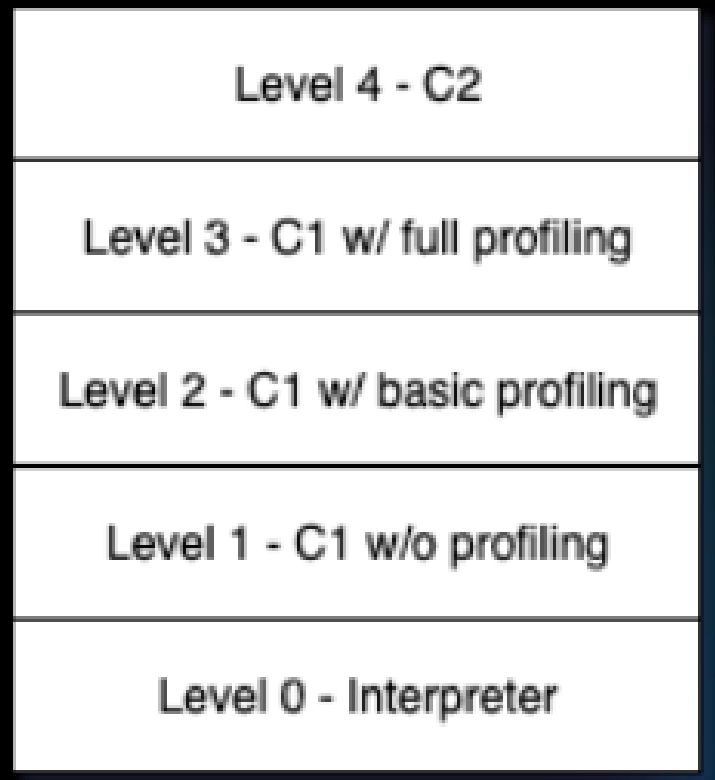
Objects that are expected to be reused across multiple invocations should move its respective initialization code to either the static initializer or constructor.

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`



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

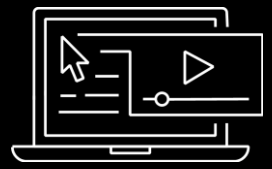
https://tinyurl.com/modern-apps-aws

Visit resource hub

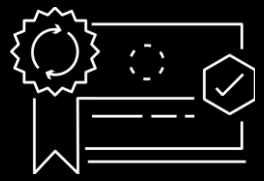
AWS Training and Certification

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

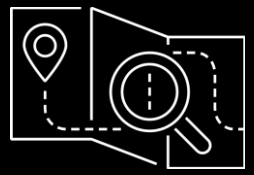
Achieve key milestones and plan your next steps with the AWS Modern Application skills training



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

aws

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!