AWS Java 1.0

Extension ID

com.castsoftware.awsjava

What's new?

See AWS Java 1.0 - Release Notes for more information.

Objects

Icon	Description
	Java AWS Lambda GET Operation
	Java AWS Lambda POST Operation
	Java AWS Lambda PUT Operation
	Java AWS Lambda DELETE Operation
	Java AWS Lambda ANY Operation
λ	Java AWS Lambda Function
λ	Java Call to AWS Lambda Function

In what situation should you install this extension?

The **AWSJava** extension is responsible of creating objects describing Amazon Web Service (AWS) lambda functions but only in the context of *java* technology. (Similarly the NodeJS extension is responsible of AWS lambda Functions created for the *nodejs* runtime). To run **fully,** this extension needs to add a Universal Analysis Unit and to switch on the HTML5 language. This allows analyzing serverless configuration (YAML) files used to define the lambda functions. This is particularly necessary for the creation of operation-type objects like *Java AWS Lambda GET Operation* and the *Java AWS Lambda Function* object*.* The complementary analysis responsible of creating *Java Call to AWS Lambda Function* objects is based on the analysis of java files and it will be launched without any further requirement upon installation of the *com.castsoftware.awsjava* plugin.

The resolution of handler methods in present version of the analyzer relies on the interpretation of *serverless* configuration files. If these files are missing (or a different framework is used) no server-side objects are expected to be created.

Server side (*serverless* framework)

Below a typical serverless configuration file *serverless.yml* is presented (any name is supported, with both 'yml' and 'yaml' extensions, but the latter extension might require to be added explicitly in the Analysis Unit configuration in CAST-MS).

```
# serverless.yml, adapted from https://github.com/zanon-io/aws-serverless-demo

service: my-serverless-demo
provider:
  name: aws
  runtime: nodejs4.3
  region: us-east-1
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - 'sdb:Select'
      Resource: "arn:aws:sdb:${self:provider.region}:*:domain/Weather"
functions:
  weather:
    handler: handler.currentTemperature
    events:
      - http: GET weather/temperature
      - http: POST weather/temperature
    memorySize: 128
    timeout: 10
  meanweather:
    handler: handler.meanTemperature
    events:
      - http:
          path: weather/temperature/mean
          method: get
    memorySize: 128
    timeout: 10
    runtime: java8
```

The *com.castsoftware.awsjava* will analyze the *.yml* file and it will create a *JAVA AWS Lambda Function* object **only** for the *meanweather* lambda function, because of the runtime java8.

In the example above the lambda function *weather* belongs to the nodejs4.3 runtime because of the global runtime set in the "provider" element and not being specified otherwise as in *meanweather*. Thus the former is not handled by the *com.castsoftware.awsjava* analyzer (it will be handled by the NodeJS extension).

The links between the *Java AWS Lambda Function* object and the actual java method (if present and resolved) will be done at application-level and indicated with a line in the log similar to:

```
Running plugin com.castsoftware.awsjava...
Start application level analysis (AWSJava)
Created link from lambda function meanweather to Java handler method com.castsoftware.api.common.lambda.
MeanWeather.handleRequest
...
Done running plugin com.castsoftware.awsjava.
```

As an example, we can analyze one of the samples provided by Amazon lambda-java8-dynamodb.

```
service: java-dynamo    # simplified version of the "serverless.yml" file

provider:
  name: aws
  runtime: java8
  ...
...

functions:
  getTags:
    handler: com.serverless.TagsHandler
    events:
      - http:
          path: java-dynamo/tags/{brand_id}/{language}
          method: get
```

The *events* field in the *serverless* file links a URL to a given Lambda function. This connection is represented by a *Java AWS Lambda GET Operation* object.

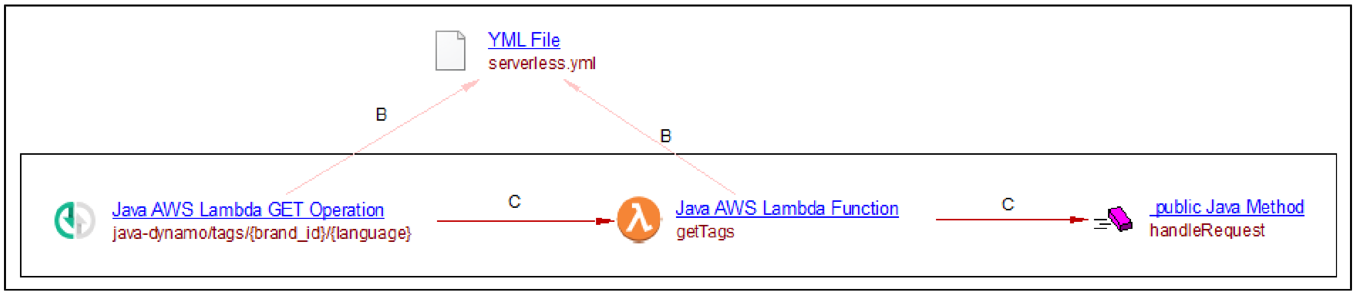
The call-back code to be executed after invocation of the Lambda function:

```
import com.amazonaws.services.lambda.runtime.RequestHandler;
import ...

public class TagsHandler implements RequestHandler<Map<String, Object>, ApiGatewayResponse> {
    ...

    @Override
    public ApiGatewayResponse handleRequest(Map<String, Object> input, Context context) {
                   // the content of the AWS Lambda Function
        }
    ...
}
```

The results of the analysis as shown in Enlighten:



Similar results are expected for the predefined interface *com.amazonaws.services.lambda.runtime.**RequestStreamHandler.***

Client-side

@LambdaFunction annotation

The following two annotations (equally named) are supported (one for Android and the other one for Java applications):

- com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction
- com.amazonaws.services.lambda.invoke.LambdaFunction

The example below reproduced from the official documentation https://docs.aws.amazon.com/lambda/latest/dg/with-android-example.html, illustrates how one can define (but not implemented) the lambda function by annotating the *AndroidBackendLambdaFunction* method.

```
// https://docs.aws.amazon.com/lambda/latest/dg/with-android-example.html
import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;
public interface MyInterface {

    /**
     * Invoke the Lambda function "AndroidBackendLambdaFunction".
     * The function name is the method name.
     */
    @LambdaFunction
     ResponseClass AndroidBackendLambdaFunction(RequestClass request);

}
```

The *@LambdaFunction* annotation maps the specific client method to the same-name Lambda function. The expected behavior of the JEE analyzer is to resolve the actual Lambda Function invocation via method calls (elsewhere) to the annotated interface methods. The *com.castsoftware.awsjava* extension on the other hand will create a *Java Call to AWS Lambda Function* object and the respective incoming *callLink* from the annotated interface method *Androi dBackendLambdaFunction.*

Linking client/server

The extension *com.castsoftware.wbslinker* is responsible of matching *Java Call to AWS Lambda Function* objects to Lambda Function objects such as *Jav a AWS Lambda Function.*

Limitations

- A single inheritance depth level is only supported for classes implementing predefined interfaces such as *RequestHandler* andRequestStreamHan dler when searching for handler methods. However overriding/overloading of the handler methods is not fully supported.
- No **custom lambda function resolver** is supported, i.e. that passed to *lambdaFunctionNameResolver* (conntected to the building of *LambdaInvo kerFactory*).