# AWS Java 1.1



## Extension ID

**com.castsoftware.awsjava**

## What's new?

See AWS Java 1.1 - 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 |
|  | Java AWS Lambda Function |
|  | Java Call to AWS Lambda Function |
|  | Java AWS Simple Queue Service Receiver |

## 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* technol ogy. (Similarly the NodeJS extension is responsible of AWS lambda Functions created for the *nodejs* runtime). To run **fully,** this extension requires a Universal Analysis Unit with the HTML5 language switched on. This allows analyzing the .json or .yml configuration files (which are used by deployment frameworks to build aws applications). The supported deployment frameworks are **Serverless Framework**, **CloudFormation**, and **Serverless Application Model (SAM)**.

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.

Only **Serverless Framework**, **CloudFormation**, and **Serverless Application Model (SAM)** deployment frameworks are supported.

# Lambda functions

## 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
```

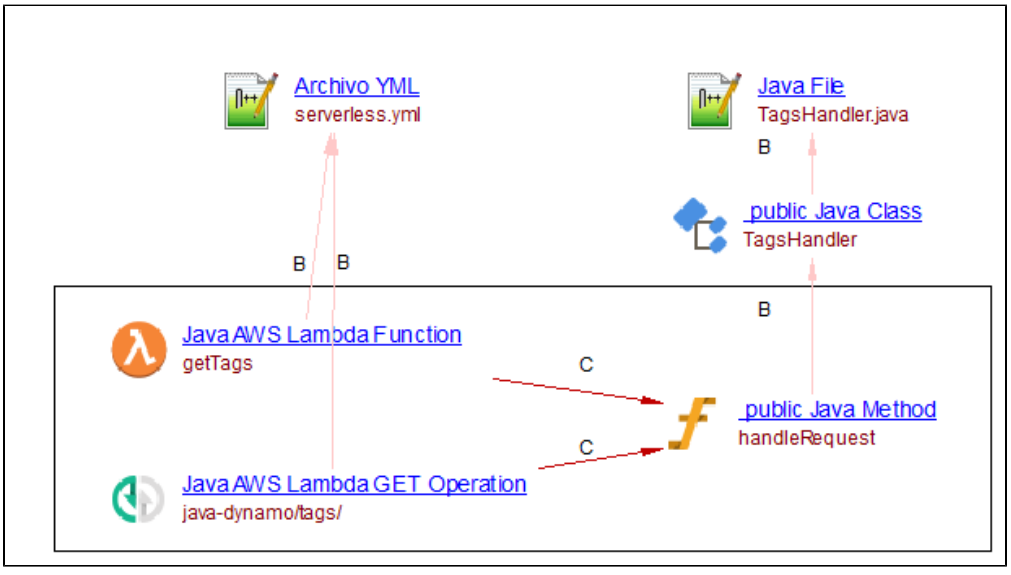
The *events* field in the *serverless* file links a URL to a given Lambda function. This alternative *entry point* for triggering the lambda method is represented by a *Java AWS Lambda GET Operation* object. From extension version 1.1 this operation object is linked directly to the Java handler method (containing 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.***

## CloudFormation templates

The analyzer will create objects in the same line as those above described for the s*erverless* framework. The same restriction applies: only those Lambda Functions using a **Java Runtime** (e.g. *java8*) and their possibly defined Lambda Operations are created by this extension.

In the (simplified) example below, we would have a *Java AWS Lambda Function* and a *Java AWS Lambda GET Operation* created, the latter with the URL name "**/mypath/{}**" (the special notation "**+**" added to the parameter is substituted by an empty bracket).

```
AWSTemplateFormatVersion: "2010-09-09"

Resources:

  MyLambda:
    Properties:
      ...
      Runtime: java8
    Type: AWS::Lambda::Function

  RestApi:
    Properties:
      Body:
        paths:
          "/mypath/{path+}":
            get:
              produces:
              ...

              x-amazon-apigateway-integration:
                ...
                type: aws_proxy
                uri:
                  ...  # reference to MyLambda function
    ...
    Type: 'AWS::ApiGateway::RestApi'
```

The connection between the Lambda object (and eventually its handler Java Method) and the Operation is done via analysis of the "**x-amazon-apigateway-integration**" element. The functions supported for string manipulation are "**Fn::Join**" and "**Fn::GetAtt**".

## Lambda triggers

The lambda can be triggered by several kinds of events. The following event types are supported.

### ApiGateway

The analyzer creates an AWS Lambda operation with a call link to the handler of the function as documented in this section

### SQS

A lambda function can be set to be executed whenever a message is sent to a given SQS queue. For instance, with the following serverless framework file the **mylambda** function will be executed when a message is sent to the **MyQueue** SQS queue.

```
functions:
  mylambda:
    handler: com.amazonaws.example.serverless.Handler
    events:
      - sqs:
          arn:
            Fn::GetAtt:
              - MyQueue
              - Arn


resources:
  Resources:
    MyQueue:
      Type: "AWS::SQS::Queue"
      Properties:
        QueueName: "MyQueue"
```

The analyzer creates a Java AWS Simple Queue Service Receiver object linked to the handler of the lambda:

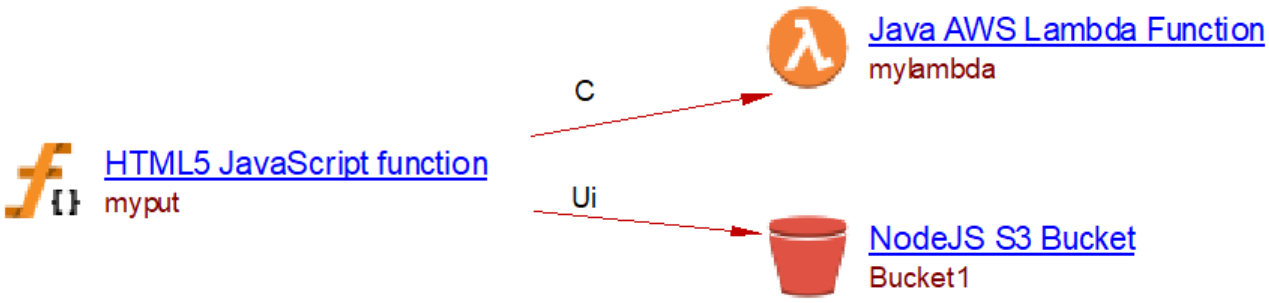

The com.castsoftware.wbslinker would then link any SQS Publisher object (created by any extension) with the same queue name to that AWS Simple Queue Service Receiver object.

## S3

A lambda function can be set to be executed whenever a given event on a given S3 bucket happens. For instance, with the following serverless framework file the **mylambda** function will be executed when an object is created in the **Bucket1** S3 bucket.

```
functions:
  mylambda:
    handler: com.foo.fooo.Handler
    events:
      - s3:
          bucket: Bucket1  #bucket name
          event: s3:ObjectCreated:*
```

The analyzer will create a call link to the lambda from all callables linked to **Bucket1** through a useInsert link.




**The support for S3 (i.e. creation of the S3 bucket objects and links to that bucket) is not carried out by this extension. Note that, S3 is currently supported only for nodejs.**

If the bucket was created by a version of com.castsoftware.nodejs older or equal to 2.5.3-funcrel, the linking between the bucket caller and the lambda will not be created.

The following table tells which link type will match which event type.

| event_type | matching link types |
|---|---|
| No event_type | all |
| ObjectCreated | useInsert |
| ObjectRemoved | useDelete |
| ObjectRestore | None |
| ReducedRedundancyLostObject | None |
| Replication | None |

In AWS, the event_type can be more specific by adding information after the semicolon. However, the analyzer does not consider this information. For instance, it will make no distinction between ObjectCreated:* and ObjectCreated:Put or ObjectCreated:Post event types.

## @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 *AndroidBackendLambdaFunction.*

# Linking

The extension *com.castsoftware.wbslinker* is responsible for matching *Java Call to AWS Lambda Function* objects to Lambda Function objects such as *Java AWS Lambda Function* during application-level analysis.

# Limitations

- A single inheritance depth level is only supported for classes implementing predefined interfaces such as *RequestHandler* andRequestStreamHandler 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* (connected to the building of *LambdaInvokerFactory*).