The programming model explains how to write your Java code in detail, for example the input/output types AWS Lambda supports. For more information about the programming model, see Programming Model for Authoring Lambda Functions in Java (p. 30). For now, note the following about this code:

- When you package and upload this code to create your Lambda function, you specify the `example.Hello::myHandler` method reference as the handler.
- The handler in this example uses the `int` type for input and the `String` type for output.

  AWS Lambda supports input/output of JSON-serializable types and InputStream/OutputStream types. When you invoke this function you will pass a sample int (for example, 123).
- You can use the Lambda console to manually invoke this Lambda function. The console always uses the `RequestResponse` invocation type (synchronous) and therefore you will see the response in the console.
- The handler includes the optional `Context` parameter. In the code we use the `LambdaLogger` provided by the `Context` object to write log entries to CloudWatch logs. For information about using the `Context` object, see The Context Object (Java) (p. 40).

First, you need to package this code and any dependencies into a deployment package. Then, you can use the Getting Started exercise to upload the package to create your Lambda function and test using the console. For more information creating a deployment package, see Creating a Deployment Package (Java) (p. 89).

Programming Model for Authoring Lambda Functions in Python

The following sections explain how common programming patterns and core concepts apply when authoring Lambda function code in Python.

Topics

- Lambda Function Handler (Python) (p. 50)
- The Context Object (Python) (p. 51)
- Logging (Python) (p. 54)
- Function Errors (Python) (p. 56)

Lambda Function Handler (Python)

At the time you create a Lambda function, you specify a *handler*, which is a function in your code, that AWS Lambda can invoke when the service executes your code. Use the following general syntax structure when creating a handler function in Python.

```
def handler_name(event, context):
    ...
    return some_value
```

In the syntax, note the following:

- `event` – AWS Lambda uses this parameter to pass in event data to the handler. This parameter is usually of the Python `dict` type. It can also be `list`, `str`, `int`, `float`, or `NoneType` type.
- `context` – AWS Lambda uses this parameter to provide runtime information to your handler. This parameter is of the `LambdaContext` type.
- Optionally, the handler can return a value. What happens to the returned value depends on the invocation type you use when invoking the Lambda function:
  - If you use the `RequestResponse` invocation type (synchronous execution), AWS Lambda returns the result of the Python function call to the client invoking the Lambda function (in the HTTP

response to the invocation request, serialized into JSON). For example, AWS Lambda console uses the `RequestResponse` invocation type, so when you invoke the function using the console, the console will display the returned value.

If the handler returns `NONE`, AWS Lambda returns null.

- If you use the `Event` invocation type (asynchronous execution), the value is discarded.

For example, consider the following Python example code.

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
                                    event['last_name'])
    return {
        'message' : message
    }
```

This example has one function called `my_handler`. The function returns a message containing data from the event it received as input.

To upload and test this code as a Lambda function

1. Save this file (for example, as `hello_python.py`).
2. Package the file and any dependencies into a .zip file. When creating the zip, include only the code and its dependencies, not the containing folder.

   For instructions, see Creating a Deployment Package (Python) (p. 96).
3. Upload the .zip file using either the console or AWS CLI to create a Lambda function. You specify the function name in the Python code to be used as the handler when you create a Lambda function. For instructions to create a Lambda function using the console, see Create a Simple Lambda Function (p. 9). In this example, the handler is `hello_python.my_handler` (*`file-name`*.*`function-name`*). Note that the Getting Started (p. 3) uses a blueprint that provides sample code for a Lambda function. In this case you already have a deployment package. Therefore, in the configure function step you choose to upload a zip.

   The following `create-function` AWS CLI command creates a Lambda function. Among other parameters, it specifies the `--handler` parameter to specify the handler name. Note that the `--runtime` parameter specifies `python3.6`. You can also use `python2.7`. For a complete description of the `create-function` command and its parameters, see CreateFunction (p. 429)

   ```
   aws lambda create-function \
   --region region \
   --function-name HelloPython \
   --zip-file fileb://deployment-package.zip \
   --role arn:aws:iam::account-id:role/lambda_basic_execution  \
   --handler hello_python.my_handler \
   --runtime python3.6 \
   --timeout 15 \
   --memory-size 512
   ```

The Context Object (Python)

Topics