Function as a Service

Dr Peadar Grant

February 1, 2024

1 Serverless compute

Serverless functions are the basic PaaS compute component:

- Sometimes referred to as Function-as-a-Service
- basic FaaS offering in AWS is called Lambda.

1.1 Lambda components

Function is an AWS resource you can *invoke* to run your own code on AWS infrastructure.

- Lambda runs your custom code on AWS
- No need for you to setup EC2 instances and VPC.

Executable consists of *code* within a lambda-provided *runtime*:

Code: provided by you, (source code, bytecode or compiled code).

Runtime: provided by AWS. (must match code)

Event invokes the lambda function. Might be:

- explicit `invoke` at the CLI or console, or
- triggered from other AWS services (S3, SNS, API Gateway, SQS polling).

2 Code

2.1 Runtime

Runtime refers to the support AWS Lambda offers for different languages. Supports offered for different languages varies:

- **Python and JavaScript are best supported:**
  - You can write / edit functions in the AWS Console.
  - Most examples online involve these languages.
- A number of other languages are fully supported:
  - must be authored / edited locally and uploaded in a *deployment package*.
  - These include: Java, .net (incl. C# and PowerShell).
- Other languages or legacy code can be run on Lambda by creating a *custom runtime*. (Involved process!)

2.2 Deployment package creation

Code is bundled into a ZIP file, named a *deployment package*. The precise layout will depend on the chosen runtime.

- You don't need to worry about this when creating code in the AWS Console.
- Interpreted languages (like Python, JS) will need the source files.
- Bytecode-compiled languages (like Java) will consist of the bytecode-compiled class files.

The relevant files can be ZIPped using:

- On Windows `Compress-Archive` cmdlet in PowerShell.
- On Mac / Linux `zip` command in Bash/zsh.

3 Execution role

Execution roles are assumed by a lambda function, and grant the function permissions to use other AWS resources in your account.

Each execution role has a name (e.g. `helloworld-ex`) from which we can derive its ARN:

```
$ExecutionRoleName="helloworld-ex"
$ExecutionRoleArn="arn:aws:iam::123456789012:role/helloworld-ex"
Write-Host $ExecutionRoleArn
```

3.1 Trust policy

The trust policy specifies what AWS component(s) can assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

You can use this trust policy as-is for all your lambda code.

3.2 Attached policies

Execution roles can then have policies attached using `iam`. Rather than write these from scratch, we will attached AWS Managed Policies.

The simplest policy (`AWSLambdaBasicExecutionRole`) allows a lambda function to write to CloudWa
Logs:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Additional policies can be attached (or created and attached) if the lambda function needs access to other AWS services.

4 Hello World example

Our simple `Hello World` will just output a message to CloudWatch logs. It will ignore the input.

The program is a single function, named `hello_handler` in a file named `hello_handler.py`. It will run as the execution role `hello-ex`.

4.1 Code

We have a python file `hello_handler.py` with the following simple program:

```
def hello_handler(event, context):

    greeting = "Hello %s %s" % ( event['firstname'], event['surname'])

    # "print" statement (redirected to log)
    print(greeting)

    # return value
    return {
        "message": greeting
        }
```

4.2 Deployment package creation

To package the code into a ZIP we can:

```
Compress-Archive -Path hello_handler.py -DestinationPath hello_code.zip

# issues on Mac/Linux in PowerShell due to file permissions, use instead (in Bash):
zip hello_code.zip hello_handler.py
```

4.3 Execution role creation

Our simple trust policy allows Lambda to assume the role (and will work for most functions), defined in `trust_policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Then we can create the execution role itself:

```
# create the role
aws iam create-role `
--role-name hello-ex `
--assume-role-policy-document file://trust_policy.json

# attach AWSLambdaBasicExecutionRole (for basic I/O needed by Lambda function)
aws iam attach-role-policy `
--role-name helloworld-ex `
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

4.4 Function creation

The handler parameter specifies the entry point that handles the event. Its format varies dependin
on the runtime (language) used. For python it normally is:

```
[file (without extension)].[function name]
```

```
aws lambda create-function `
--function-name hello `
--zip-file fileb://hello_code.zip `
--handler hello_handler.hello_handler `
--runtime python3.8 `
--role arn:aws:iam::123456789012:role/hello-ex
```

4.5 Invoking the function

We invoke (or run) the function:

```
# invoke the function
aws lambda invoke --function-name hello out.txt

# read output produced
Get-Content hello_out.txt
```

4.6 Updating code

If we want to update the function's code, we can modify the source files and then:

```
# new ZIP file:
Compress-Archive -Force -Path hello_handler.py -DestinationPath hello_code.zip

# update the code on Lambda
aws lambda update-function-code --function-name hello --zip-file fileb://hello_code.zip
```

4.7 Deletion

```
aws lambda delete-function --function-name hello
```

5 Input handling

Input is passed in/out of lambda functions using JSON-formatted text. Python has a built-in dictionary time which the incoming JSON is transparently converted to. Imagine we modified our hello function to be:

We expect a firstname and surname input. These are provided as JSON, so here we make a file for PowerShell to send the lambda function:

Then invoke, this time with the payload given:

```
aws lambda invoke --function-name hello `
--payload fileb://payload.json hello_out.txt
```