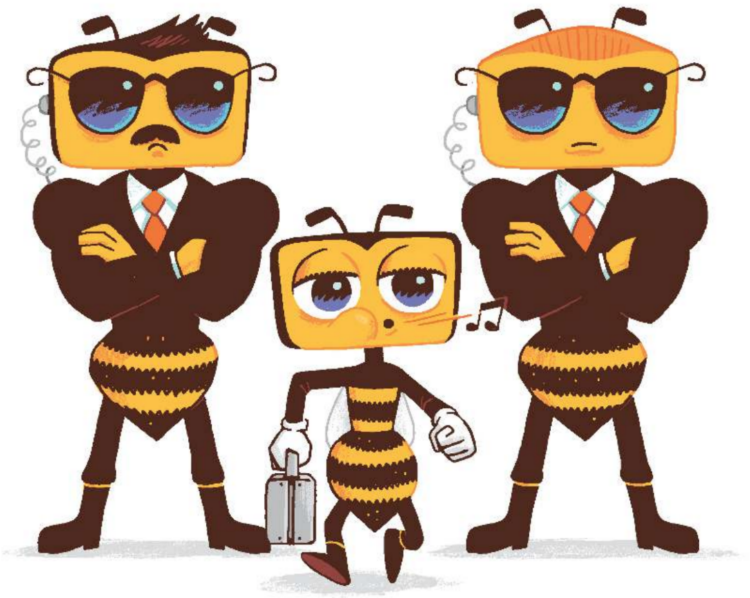
Safe Deployments

In this lesson, you will learn about function configurations for AWS Lambda functions.

WE'LL COVER THE FOLLOWING

- Function configurations

This chapter explains two core features of AWS Lambda: versions and aliases. You will also learn how to use aliases to protect against unexpected deployment problems.



Although AWS Lambda technically works like a virtualised container management system, the life-cycle for Lambda functions is quite different from that of the usual container-based applications. This might be counter-intuitive to people who are used to managed container services. Luckily, AWS SAM hides most of the underlying complexity from developers. In this chapter, you will peek under the hood just enough so you can avoid subtle mistakes. Knowing how Lambda routes requests and provisions runtime environments is also critical for getting the most out of cloud functions.

In the previous chapter, you deployed a new version of the example stack to add logging. New code was available and it responded to user requests almost immediately after deployment. This can give the wrong impression that Lambda works similarly to application hosting services such as App Engine or Heroku. With application hosts, a new deployment typically uploads code to running containers or sets up new instances with the updated code and then destroys the old containers. However, a Lambda deployment does not create or destroy any containers. It only creates a new *function configuration*.

Function configurations

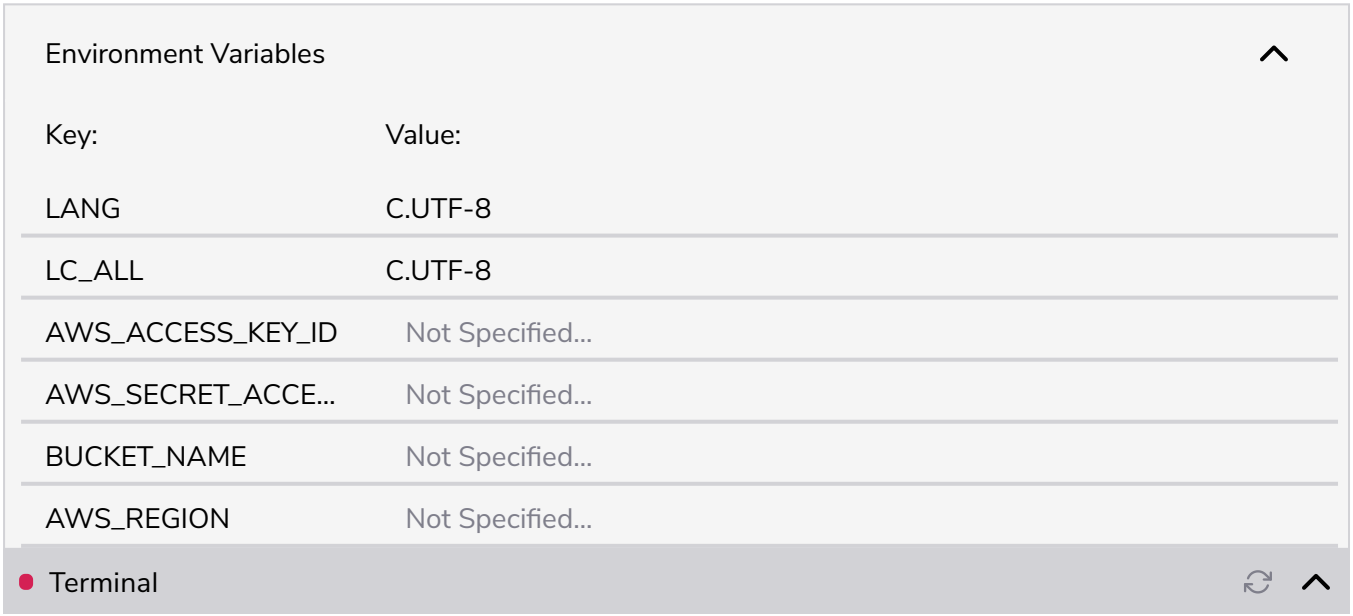
The core principle of serverless work is that the platform is responsible for receiving events from clients, not your code. The network socket (*server*) belongs to the hosting provider, not to your function. This allows the platform to decide how many instances to run and when. Lambda will not start any containers for your function until it receives events (unless you explicitly provisioned concurrency). Once related events arrive, the platform will start and maintain just enough instances to handle the workload. After a period of inactivity, if no events ask for a particular function, Lambda will remove the inactive containers. Although as a user you do not have any control over this process, it's relatively easy to prove it: just measure request latency to spot cold starts.

That process is common for all serverless platform providers. With AWS Lambda in particular, events do not target a particular function; they target a particular version of a function, or, more precisely, events are connected to a version of the function configuration. In Lambda terminology, the *function configuration* describes all the properties of a runtime environment necessary to spin up a new container. That includes the following:

- The runtime type and version (so far you have used Node 12, but this could be a version of Python, Java, Ruby, and so on).
- How much memory and time the function can use.
- The URL of the function code package (SAM will set this up on S3 for you during deployment).
- The IAM role specifying what this function can access in AWS and who can call this function.
- Error recovery, logging, and environment parameters of the function.

You can see the configuration for any Lambda function by using the AWS CLI tools. First, find the actual name of the function by listing the stack resources (replace `<STACK_NAME>` with your stack name):

```
aws cloudformation list-stack-resources --stack-name <STACK_NAME>
```



In the results, find the physical ID associated with a resource of type `AWS::Lambda::Function`. For example, in the following output, the name of the function is next to `PhysicalResourceId` :

```
$ aws cloudformation list-stack-resources --stack-name sam-test-1
{
  "StackResourceSummaries": [
    {
      "LogicalResourceId": "HelloWorldFunction",
      "PhysicalResourceId": "sam-test-1-HelloWorldFunction-QL5VEY42DZ2C",
      "ResourceType": "AWS::Lambda::Function",
      "LastUpdatedTimestamp": "2020-01-16T08:58:53.841Z",
      "ResourceStatus": "UPDATE_COMPLETE"
    },
```

Now that you know the function name, it's easy to look up the details of the configuration. Use the following command line and replace the placeholder `<NAME>` with the physical resource ID from the previous example:

```
aws lambda get-function-configuration --function-name <NAME>
```

Here's an example of the output for the author's stack:

```
$ aws lambda get-function-configuration --function-name sam-test-1-HelloWorldFunction-1CWVM0T
{
    "FunctionName": "sam-test-1-HelloWorldFunction-1CWVM0TOM9PEV",
    "FunctionArn": "arn:aws:lambda:us-east-1:536317743507:function:sam-test-1-HelloWorldFunct
    "Runtime": "nodejs12.x",
    "Role": "arn:aws:iam::536317743507:role/sam-test-1-HelloWorldFunctionRole-111BMRQLOR8U6",
    "Handler": "app.lambdaHandler",
    "CodeSize": 127418,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2020-01-27T08:30:23.430+0000",
    "CodeSha256": "Mmtt62vbphdBhB5QkIDDENE2Nnhz7N8N5p5hNiW2iTc=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "6c72268b-abe9-4d09-ac0e-7879bda70fe7",
    "State": "Active",
    "LastUpdateStatus": "Successful"
}
```

Notice the `Version` property in the output. The value `$LATEST` is a special key, assigned by Lambda to the default configuration version. When you run `sam deploy`, Lambda only needs to update this configuration; it does not really need to start up or stop any containers.

As an experiment, try extracting the version automatically using the `--query` and `--output` parameters as explained in the section *Inspecting a stack from the command line* in Chapter 3. For extra points, try combining all the commands into a single line that only requires the stack name as an argument.

In the next lesson, you'll have a look at versions and aliases for the Lambda functions.