Lab3

Internet Applications

Serverless Computing

Prof. Donal O'Mahony

Preamble:

AWS Serverless Computing runs code in containers that are instantiated on-demand. The software support available depends on what code is installed in the container and the execution needs to be stateless as one must assume that a new container could be instantiated for any invocation – even though ones that are already running may endure for as much as a hour of idleness after the first invocation.

When we introduced Lambda in lectures, we used the newest method that AWS have released for developing code i.e. direct editing of a Node.js function in their Cloud9 editor. This may have been a bad choice since it means that there is no framework support (Express) and APIs that may be needed to interact with other services (other than Amazon's) are not available.

Alternative methods of developing Lambda functions involve the use of Frameworks such as 'Claudia', 'Serverless' etc that allow code and supporting frameworks to be bundled and uploaded to AWS so that they are available in the container instance. These frameworks are subject to sudden change and possible obsolescence as the AWS Lambda service evolves – but right now, they are necessary to get any kind of useful service working on Lambda. The objective of this lab is do get something (not very useful!) working on Lambda to allow you to appreciate the possibilities of the service.

Lab Objective

This lab will involve the development of a web API, implemented by a single AWS Lambda function that will maintain a things-to-do-list consisting of a list of simple strings.

Since AWS Lambda is stateless, you will notice that when left idle for a period, the to-do list will disappear and be replaced with an empty (initialized) to-do list – but we will work with this!

Creating the Lambda function

Login to the AWS console; invoke the Lambda Console and "Create Function"..."Author from Scratch". In the "Designer" pane – click on the function to edit it. The simplest function to start off with is the following – which just returns everything passed in to it as a JSON object.

```
exports.handler = async (event) => {


    return("Hello from Lambda");

}
```

The Cloud9 editor lets you setup test events and you can see how the function reacts by pressing "Test".

Using API Gateway to direct requests to your function

Go back to the Aws console home page and select "API Gateway". This will allow you to setup an endpoint on the web that will receive requests from a browser – wrap them up an event and pass that event into your newly created Lambda Function.

Under API gateway click on "Create API"… type:REST select "new API" – call it something like mytasksapi – Create a Resource for the API. For the GET method connect it to your LambdaFunction.

Once you are happy that the function is doing what you want, you can set up a trigger that will cause it to be invoked.

At the Designer View (top of page) click "Add Trigger" – notice the choices you are offered.

We want our function to respond to a REST request that will come in from a web browser, so select API Gateway.

You now need to create a new "REST" api to field the requests that come it and convert them into events to be passed to your Lambda function. Make sure the API has security set to "Open"

Return to the "Designer View". Clicking on the "API Gateway" icon and "Details" should show a URL – something like: https://264ej4vqje.execute-api.us-east-1.amazonaws.com/default/myfirstfunc

Which will invoke the Lambda function you have just written

If your function is written to return JSON.Stringify(event), you will see everything that is passed in to the function.

We have now seen how to execute the function: In the Lambda Cloud9 editor, from the API gateway screen and we can also execute it from the browser. We have also seen how to return values from the code.

Getting Inputs from the Browser into the Lambda Function

```
return("My Lambda function was passed:"+JSON.stringify(event) )
```

You should now be able to field URL requests from a browser of the form:

https://mrud6jbcy3.execute-api.eu-west-1.amazonaws.com/default/mysecondfunc/?task=blahblah

And be able to capture the “task” in your code using:

event.queryStringParameters.task

Exercising the Lambda Function

Now that you can invoke the function, send items in using a query string and send JSON out using the “return” statement – the assignment is as follows:

Implement a very basic to-do list which has a single primitive of the form:

https://mrud6jbcy3.execute-api.eu-west-1.amazonaws.com/default/mysecondfunc/?task=doSomethingElse

This primitive will add a new item to the list and return the full to-do list in a form that is something like:

{ “dosomething”, “doSomethingElse” }

Notice that the Lambda function “remembers” what you sent to it, provided the time-gap is small. This is the so-called “warm start” of a Lambda. The first time you invoke it, a container is spawned – this container stays in place (with its global variables intact) for a time in order to field subsequent requests. After a sustained idle period, it is terminated.