AWS Lambda Learning

Reference: http://docs.aws.amazon.com/lambda/latest/dg/lambda-app.html

1. About AWS Lambda

AWS Lambda is a compute service that lets us run code without provisioning or managing servers. AWS Lambda executes our code only when needed and scales automatically, from a few requests per day to thousands per second. We pay only for the compute time we consume - there is no charge when our code is not running. With AWS Lambda, we can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs our code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All we need to do is supply our code in one of the languages that AWS Lambda supports (currently Node.js, Java, C# and Python).

2. Lambda Functions

After we package up our custom code, including any dependencies, and upload it to AWS Lambda, we have created a Lambda function.

- Authoring Code for Our Lambda Function

Language	Tools and Options for Authoring Code	More Info
Node.js	◆ AWS Lambda console ◆ Visual Studio, with IDE plug-in (see AWS Lambda Support in Visual Studio) ◆ Our own authoring environment	We can use the console if the languages we choose do not require compilation, the code is saved in a single file, and it does not depend on any libraries.
Java	◆ Eclipse, with AWS Toolkit for Eclipse (see Using AWS Lambda with the AWS Toolkit for Eclipse) ◆ Our own authoring environment	The AWS Toolkit also creates the deployment package, which is explained in Deploying Code and Creating a Lambda Function.
C#	◆ Visual Studio, with IDE plug-in (see AWS Lambda Support in Visual Studio) ◆ .NET Core (see .NET Core installation guide) ◆ Our own authoring environment	The AWS Toolkit also creates the deployment package, which is explained in Deploying Code and Creating a Lambda Function.
Python	◆ AWS Lambda console ◆ Our own authoring environment	We can use the console if the languages we choose do not require compilation, the code is saved in a single file, and it does not depend on any libraries.

3. Programming Model - Python

- Lambda Function Handler

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

def handler_name(event, context):

...
return some_value

- *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 our handler. This parameter is of the LambdaContext type.*
- *some_value – the handler can return a value. What happens to the returned value depends on the invocation type we use when invoking the Lambda function.*

- The Context Object

While a Lambda function is executing, it can interact with the AWS Lambda service to get useful runtime information such as:

- How much time is remaining before AWS Lambda terminates our Lambda function (timeout is one of the Lambda function configuration properties).
- The CloudWatch log group and log stream associated with the Lambda function that is executing.
- The AWS request ID returned to the client that invoked the Lambda function. We can use the request ID for any follow up inquiry with AWS support.
- If the Lambda function is invoked through AWS Mobile SDK, we can learn more about the mobile application calling the Lambda function.

- Logging

Our Lambda function can contain logging statements. AWS Lambda writes these logs to CloudWatch. If we use the Lambda console to invoke our Lambda function, the console displays the same logs.

The following Python statements generate log entries:

- *print* statements.
- Logger functions in the *logging* module (for example, *logging.Logger.info* and *logging.Logger.error*).

- Function Errors

If our Lambda function raises an exception, AWS Lambda recognizes the failure and serializes the exception information into JSON and returns it.

4. Creating a Deployment Package - Python

To create a Lambda function we first create a Lambda function deployment package, a .zip or .jar file consisting of our code and any dependencies. When creating the zip, include only the code and its dependencies, not the containing folder.

- **Simple scenario** – If our custom code requires only the AWS SDK library, then we can use the inline editor in the AWS Lambda console. Using the console, we can edit and upload our code to AWS Lambda. The console will zip up our code with the relevant configuration information into a deployment package that the Lambda service can run. The Lambda service has preinstalled the AWS SDK for Python.

- Advanced scenario – If we are writing code that uses other resources, such as a graphics library for image processing, or we want to use the AWS CLI instead of the console, we need to first create the Lambda function deployment package, and then use the console or the CLI to upload the package. After we create a deployment package, we may either upload it directly or upload the .zip file first to an Amazon S3 bucket in the same AWS region where we want to create the Lambda function, and then specify the bucket name and object key name when we create the Lambda function using the console or the AWS CLI.