Lambda Layers

Here, you'll get familiar with Lambda layers and use them in your application!

WE'LL COVER THE FOLLOWING

`image-magick-lambda-layer`

The `image-magick-lambda-layer` SAR component makes ImageMagick utilities available to Lambda functions in the form of a layer.

A *Lambda layer* is a file package that can be deployed to AWS and then attached to many functions. Layers are useful for sharing large packages across functions and for speeding up deployments. For example, ImageMagick tools consume about 60 MB of file storage. Instead of including them with every single Lambda function that needs access to ImageMagick, you could pack and deploy those tools once in a layer. Individual function packages can then contain only the source code unique to each use case and can request access to image conversion tools during deployment.

From the perspective of a Lambda function, a layer is effectively a shared read-only file system. Files from a layer appear in the `/opt` directory of the Lambda virtual machine, and you can access them as if they were included in the function itself. For example, once you link the ImageMagick layer, the `mogrify` utility will be in `/opt/bin/mogrify`.

A single function can only attach up to five layers. Although layers deploy separately from functions, the attached layer size counts towards the total Lambda size. The total unpacked size of the function, including all linked layers, has to be less than 250 MB. Layers do, however, significantly reduce overall consumed deployment capacity for an AWS account. For more

information on layers and general Lambda size restrictions, check out the *AWS Lambda Limits* page in the AWS documentation.

Linking functions with layers

Layers, similar to functions, get a numerically incremented version every time they are published. Unlike Lambda functions, there are no textual aliases for numerical layer versions, so it's not possible to mark a current or latest version easily or to use labels to differentiate between production and testing layer versions. This is a serious limitation, and it's logical to expect AWS to provide a better solution in the future, but at the time this was written, a function had to specify the exact version of a layer it wanted to use. That's why you'll see lots of references to `LayerVersion` resources.

To attach a version of a layer to a function, use the `Layers` property of the `AWS::Serverless::Function` resource. The property needs to be formatted as a list of ARN identifiers of layer versions, even if you want to attach just a single layer. To get the ARN of the layer version created by the ImageMagick nested application, use the `ImageMagick.Outputs.LayerVersion` reference. Here is this concept deconstructed for easier understanding:

1. The first part, `ImageMagick`, is the name of the resource in the template you used to import the SAR component. It is just a local name used in your template, and does not need to match the remote application name.
2. The middle part, `Outputs`, is how you can tell CloudFormation to look into the outputs of an embedded template. You will always use this to read outputs from nested applications.
3. The final part, `LayerVersion`, is how the nested template called the output containing the layer version. This is specific to a particular application, and you will likely use a different name when importing a different component.

Check out the source code for the embedded template on GitHub, and you'll see a section declaring that output.

SAR application outputs

> Remember that you used template outputs in your application to make it easier for client code to learn about important resources such as the web page URL. Nested applications are just CloudFormation templates, and they can provide outputs for that same purpose.

Let's attach the layer to your `ConvertFileFunction` resource and also pass the new parameter for thumbnail width that you added earlier in this chapter. Because you'll now actually convert files, it is also good to give this function a bit more memory. This will speed up execution significantly. You can modify the function template to look similar to the following listing (the important additions are lines 14, 18, 22 and 23).

```yaml
ConvertFileFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: image-conversion/
    Handler: index.handler
    Runtime: nodejs12.x
    Events:
      FileUploaded:
        Type: S3
        Properties:
          Bucket: !Ref UploadS3Bucket
          Events: s3:ObjectCreated:*
    Timeout: 600
    MemorySize: 1024
    Environment:
      Variables:
        OUTPUT_BUCKET: !Ref ThumbnailsS3Bucket
        THUMB_WIDTH: !Ref ThumbnailWidth
    Policies:
      - S3FullAccessPolicy:
          BucketName: !Ref ThumbnailsS3Bucket
    Layers:
      - !GetAtt ImageMagick.Outputs.LayerVersion
```

Line 89 to Line 111 of code/ch10/template.yaml

In the next lesson, you'll look at how to invoke system utilities. See you there!