

Module 17

AWS Cloud Formation

Overview

So far you have been creating AWS resources (Lambda functions, S3 buckets, queues, Dynamo tables, etc.) manually from the AWS Management Console - You signed in to the AWS console → went to some service page → and created some service/resource with a few button clicks. Not only you created the resources manually but you also manually did all other needed configurations like configuring a Lambda to be triggered by some S3 bucket PUT event, etc.

This method of creating resources is OK when experimenting or doing proof-of-concepts to demonstrate the feasibility of some idea. In a production setting and for complex cloud applications with dozens of resources you can probably guess that this approach suffers from many problems. We list a few below:

- Assume you want to create multiple environments (alpha, beta, and production). You plan to do all your unit and integration testing in the alpha and beta environments. And once satisfied with the testing results, you promote your code and install it in the production environment (the environment that has the resources that are used by customers worldwide). This is what you typically do in a Continuous Integration (CI) style of development where you have frequent check-ins and deployments.

If we decide to manually create resources, then we would need to repeat the creation steps 3 times (one for each of the alpha, beta, and production environments). And if we introduce a new resource (e.g., we added a queue or a Lambda function to our application), we would have to do the same manual operations 3 times per resource.

- If we want to make our application available in many AWS regions, we would need to create all the resources once for every region.
- Creating resources using manual steps is susceptible to errors. As the number of resources increase and their configuration become more complex, it becomes more likely to make minor mistakes when repeating the steps multiple times.
- Manual steps cannot be automated and integrated in a Continuous Integration style of development work. We ideally would like to automate the creation (and deletion) of resources and make them part of the development → build → testing → deployment pipelines.
- Manual steps cannot be saved in a revision control software. Ideally we would like to turn the resources creation steps into some script that can be created, updated, and tracked with version control just like any source code file.

- Rolling back the creation of resources in case of an error becomes difficult. Imagine a cloud application with 12 resources. You successfully manually created 8 of them and encountered an error while creating the 9th resource. You now need to delete the 8 resources you created in the correct reverse order to bring your system back to how it was – a tedious and tricky process. We would like to have some intelligence similar to database transactions: when you bundle many database operations in a transaction, if some operation in the middle fails, the transaction gives you the ability to rollback all activities and bring the database back to the state it was originally in.

CloudFormation is an AWS service built for the purpose of solving the above problems. It basically allows you to turn what you have been manually doing into an automated script.

CloudFormation and Stacks

Using a declarative language, **CloudFormation** allows you to encode the creation and configuration of a collection of resources in a text file of type json or yaml. Such file is known as **CloudFormation template file**. And because you encoded all that information in a text file, that file can be checked-in into the source code versioning repository you are using, and tracked and updated just like any other source code file.

In CloudFormation, a collection of resources that can be managed as a single unit is known as a **stack**. You create, delete, or update a collection of resources by creating, deleting, or updating a stack. All the resources in a stack are defined by the stack's CloudFormation template file. For example, a stack might include an S3 bucket, a Lambda function, the configuration of Lambda with an S3 trigger, a queue, and a role with some policies attached to it. If you no longer want the resources you simply delete the stack, and all its resources will be deleted.

Before we discuss the basics of CloudFormation template files, let's look at a sample template file to get an idea how it looks like.

- Go to this [file](#).
- Notice that the template has sections like “Description”, “Resources”, “Outputs”, etc. A template file can have more types of sections that you will look at shortly. Of all the sections, only the “Resources” one is required. All other sections are optional.
- Read the “Description” section which usually tells you what the template does.

Unlike XML, JSON doesn't have the concept of comments (like `<!-- -->` in XML). So if you want to include a comment in JSON, you just have to create a key for it (e.g., a “Description” key). Although the “Description” section is optional, it is always a good practice to start a

CloudFormation template file with a “Description” section to describe what the template does.

- Go to [this page](#) and look at the anatomy of a CloudFormation template file. Read what each section means. Two sections of particular importance are:

“Resources”: The only required section. This is the most important section as it is the place where you declare the resources you want to create.

“Parameters”: Optional but very useful. It is the place where you can define things and use them in the “Resources” section. This is useful for readability or if you have the same thing that you need to use multiple times. You define it once and you refer to it as many times as you need to.

How to Create CloudFormation Templates

1. Manually edit a template file. You can start with an existing sample template file and edit it, or start a new template file and copy from existing templates any portions you need. Here are some existing [templates](#). For example, assume you need the declaration text to create an EC2 instance. You can go to an existing template like [this one](#) and search for “AWS::EC2::Instance”. You can then copy that portion to your template file and edit it to your liking.

```
"EC2Instance" : {  
  "Type" : "AWS::EC2::Instance",  
  "Properties" : {  
    "InstanceType" : { "Ref" : "InstanceType" },  
    "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],  
    "KeyName" : { "Ref" : "KeyName" },  
    "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },  
                                     { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] }  
  }  
},
```

2. Use the CloudFormation template designer. This is easier because the designer does most of the tedious work, and you fill in/change properties as needed. It is not a complete WYSIWYG type of deal because you still need to make manual edits. However, it gives you a visual of your resources and the basic CloudFormation code that you need to expand on. We will use this methodology.

Steps

Preliminary Steps

1. Create an S3 bucket (you will use this bucket to store the code of a Lambda function and the CloudFormation stack you will prepare).

- a. For bucket name use something that resembles this (using your name):

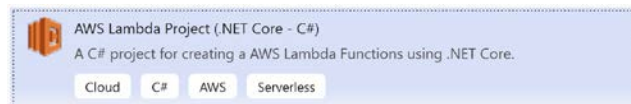
alfredn-code-and-cloudformation-bucket1

- b. For region use: US East (N. Virginia).

2. The CloudFormation stack we will create will have a Lambda function. We need the packaged code of this function (a zip file). Let's first create the function:

- a. In Visual Studio, create a solution and then create a Lambda project of type AWS Lambda Project (.NET Core – C#).

Give the solution and project the names **ResizeImageSolution** and **ResizeImageFunction**, and choose the Simple S3 Function blueprint.



- b. Modify **Function.cs** to print "Hello from my Lambda" from method **FunctionHandler**. You can delete the line that calls the **GetObjectMetadataAsync** method.

```
public async Task<string> FunctionHandler(S3Event evnt, ILambdaContext context)
{
    var s3Event = evnt.Records?[0].S3;
    if(s3Event == null)
    {
        return null;
    }

    try
    {
        Console.WriteLine("Hello from my Lambda");

        var response = await this.S3Client.GetObjectMetadataAsync(s3Event.Bucket
        return response.Headers.ContentType;
    }
}
```

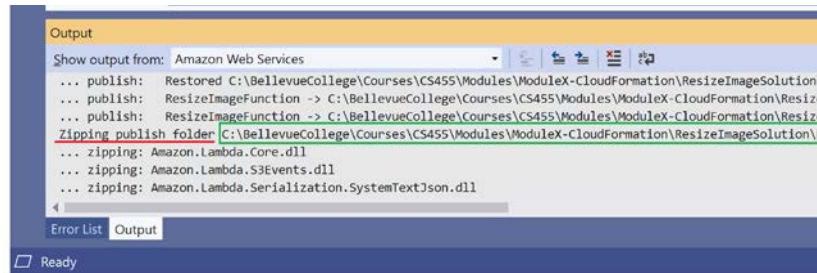
- c. Build the project. Then right-click on project **ResizeImageFunction** and choose menu **Publish to AWS Lambda....**

Give the function the name **resize-image-function**.

For Role name, choose **LabRole**.

Click the **Upload** button.

- d. Switch to the Output tab in Visual Studio and find the line that starts with **Zippping publish folder**.



- e. Find that zip file (ResizeImageFunction.zip) on your computer (highlighted in green above).
 - f. Upload this zip file to the bucket you created in step 1 above (your equivalent of alfredn-code-and-cloudformation-bucket1).
3. Login to the AWS console and go to the Lambda service.

Select function **resize-image-function** and click menu **Action → Delete**.

What do you have so far?

- You have the code of a Lambda function packaged as zip file ResizeImageFunction.zip.
- You placed that zip file in an S3 bucket that you own (your equivalent of alfredn-code-and-cloudformation-bucket1).

What you don't have?

- You don't have this Lambda function created in AWS (remember that you just deleted it).

Creating a CloudFormation template file using the CloudFormation designer

In the following steps you will create the CloudFormation template file that you will use to create the cloud resources that you need. We assume that we have a cloud application that needs the following cloud resources:

- a) A Lambda function.
- b) An S3 bucket that the Lambda uses as an event trigger.

The steps below walk you through completing (a). And you will do step (c) on your own as an exercise.

1. Login to the AWS console and go to the **CloudFormation** service.
2. Click the **Create stack** button.
3. Choose the **Create template in Designer** option. Then click the **Create template in designer** button.

This opens the template designer.

- The pane on the left has AWS resources (under **Resource types**)
- The pane on the right is the designer area.
- You drag resources from the left pane to the designer area.
- As you drag resources to the designer area, the lower pane is updated with the respective CloudFormation declaration code that reflects what you have in the design area. You get basic CloudFormation declaration text.

Let's accomplish (a): create a Lambda function, and configure the Lambda function with the LabRole.

4. Drag a **Function** (under Lambda) from the pane on the left to the designer area.
5. Change its declaration code to look like this:
 - a. Note that **LambdaFunction1** is a logical ID of this resource. It is a name for the Lambda function resource in the script (that is, this is the name you are giving to the resources in the script). It is not a name that will end up in AWS under Lambda. This name is specified by field `FunctionName`.
 - b. The **metadata** section might be different on your system. Keep the one you have.
 - c. Change the value of **S3Bucket** to the name of your bucket.
 - d. Change the value of `S3Key` to `ResizeImageFunction.zip`

```
"Code": {  
    "S3Bucket": "alfredn-code-and-cloudformation-bucket1",  
    "S3Key": "ResizeImageFunction.zip"  
}
```

- e. Change the Role part to look something like this:

Give it some name other than `HelloWorldLambdaRole`.

Replace `Arn` with the LabRole ARN (you can get that from the AWS Console – IAM -> Roles -> LabRole)

```

"Role": {
  "Fn::GetAtt": [
    "HelloWorldLambdaRole",
    "Arn"
  ]
}

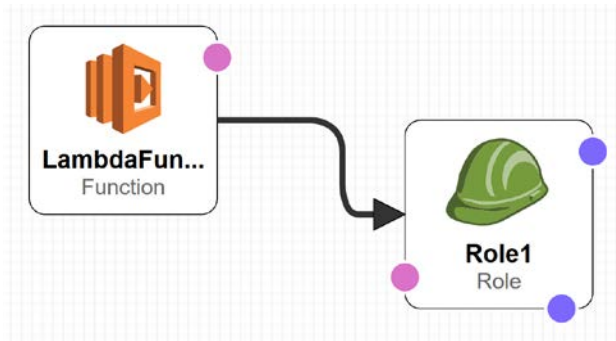
```

```

"LambdaFunction1": {
  "Type": "AWS::Lambda::Function",
  "Properties": {
    "FunctionName": "Resize-Image-Function",
    "Description": "Resizes images dropped in an S3 bucket",
    "Handler": "ResizeImageFunction::ResizeImageFunction.Function::FunctionHandler",
    "Runtime": "dotnetcore3.1",
    "Code": {
      "S3Bucket": "alfredn-code-and-cloudformation-bucket1",
      "S3Key": "ResizeImageFunction.zip"
    },
    "Role": {
      "Fn::GetAtt": [
        "LabRole",
        "arn_goes_here"
      ]
    }
  },
  "Metadata": {
    "AWS::CloudFormation::Designer": {
      "id": "cf1a5a5f-85e0-4b2b-90ef-65abd552d0e9"
    }
  }
}

```

6. Because you specified that the Lambda uses LabRole (see part highlighted in blue above), you must tell the CloudFormation engine that LabRole must be created before the Lambda. You do this by specifying that one resource depends on another. To do this, in the designer put the cursor on the pink dot of LambdaFunction1 and drag a line to LabRole (you may need to try several times to work). When done your designer should look like this (LabRole instead of Role1)



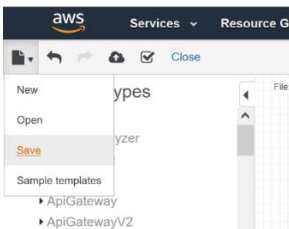
The arrow means that one resource depends on another (in this case Lambda depends on the role). And you should see the below getting added to LambdaFunction1:

```
"DependsOn": [
  "LabRole"
]
```

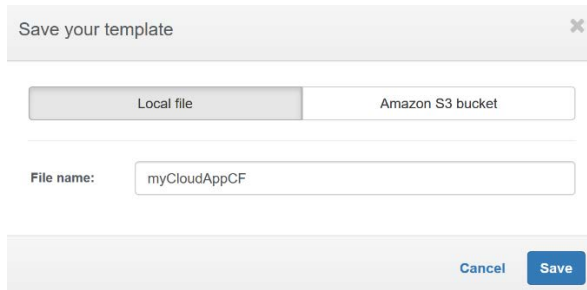
7. In the **Template** tab in the lower pane, observe what you have so far (the important things are under the “Resources” section).



8. Save your work:



9. Choose **Local file**, give your template a name (e.g., myCloudAppCF), and click the **Save** button. (If you want to later edit the template, the designer allows you to open it).



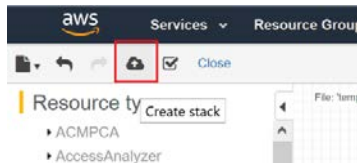
Save your template

Local file Amazon S3 bucket

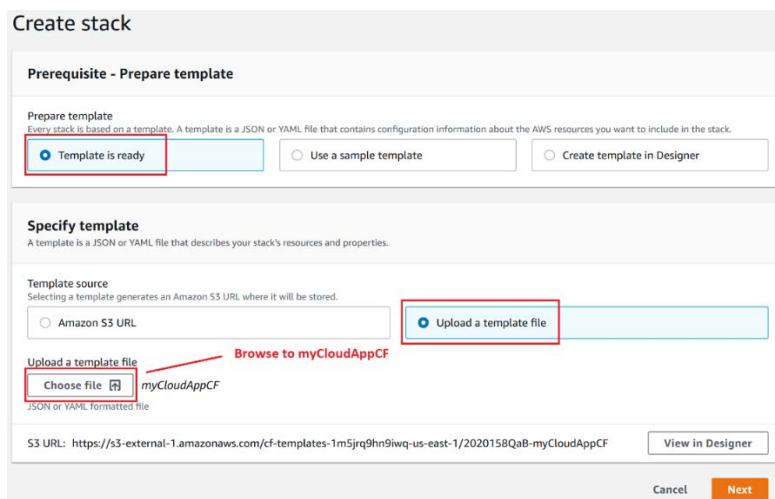
File name: myCloudAppCF

Cancel Save

10. Click the Create Stack button.



11. Complete the Create stack page as shown below. Then click **Next**.



Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Template is ready ☐ Use a sample template ☐ Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL ☒ Upload a template file

Upload a template file
Choose file Browse to myCloudAppCF

S3 URL: https://s3-external-1.amazonaws.com/cf-templates-1m5jr9hn9iwq-us-east-1/2020158QaB-myCloudAppCF View in Designer

Cancel Next

12. Enter a stack name: MyCloudAppStack-1

13. Click the **Next** button.

14. Don't make any changes in the **Configure stack options** page. Click **Next**.

15. In the **Review MyCloudAppStack-1** page, check the "I acknowledge..." checkbox at the bottom of the page.

16. Click the **Create stack** button.

The creation will be in progress as shown below. You can click the Refresh button to see if it is done. If you get errors inspect what the problem is, open myCloudAppCF in the designer again, and fix the problem. Repeat the steps.

If you don't get any errors, it should say CREATE_COMPLETE in green.

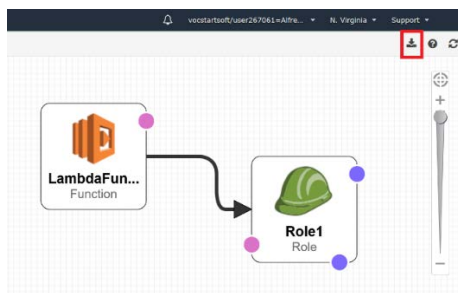
Events (8)				
<input type="text" value="Search events"/>				
Timestamp	Logical ID	Status	Status reason	
2020-06-07 14:47:53 UTC-0700	MyCloudAppStack-1	CREATE_COMPLETE	-	
2020-06-07 14:47:51 UTC-0700	LambdaFunction1	CREATE_COMPLETE	-	
2020-06-07 14:47:50 UTC-0700	LambdaFunction1	CREATE_IN_PROGRESS	Resource creation Initiated	
2020-06-07 14:47:49 UTC-0700	LambdaFunction1	CREATE_IN_PROGRESS	-	
2020-06-07 14:47:47 UTC-0700	Role1	CREATE_COMPLETE	-	
2020-06-07 14:47:34 UTC-0700	Role1	CREATE_IN_PROGRESS	Resource creation Initiated	
2020-06-07 14:47:34 UTC-0700	Role1	CREATE_IN_PROGRESS	-	
2020-06-07 14:47:30 UTC-0700	MyCloudAppStack-1	CREATE_IN_PROGRESS	User Initiated	

17. Go to the **Lambda** service. Verify that you have a Lambda function named **Resize-Image-Function**.

18. In the above example we still had to do one manual step to create the stack (we had to do it from inside the AWS Console). But this can be done from the CLI (using command `aws cloudformation create-stack`). See [here](#).

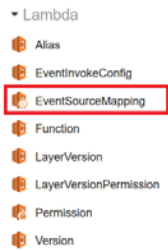
The command can then be bundled as part of some build-test-deploy automation script.

19. The designer also has a bonus feature that allows you to export the designer pane to an image. This can be useful to put in a design document or a PowerPoint presentation since the drawing should display all your cloud resources and in a way explains your cloud application.



Exercise to do on your own

1. Go back to the Template designer and open file `myCloudAppCF` in the CloudFormation designer.
2. Add 1 new resource: an S3 bucket.
3. Configure the S3 bucket as an event source for the Lambda function (use `EventSourceMapping` under Lambda). The `EventSourceMapping` documentation is shown [here](#). See HINTS below



HINTS:

- The EventSourceMapping you will add is making the S3 bucket as a trigger to the Lambda. Therefore, EventSourceMapping **DependsOn** the S3 bucket and the Lambda.
4. When done and the stack creates successfully, go the AWS console and verify that you have things the way you designed them. For example, go to Lambda and verify that the Resize-Image-Function Lambda does have an S3 trigger.

What to Submit:

Nothing to submit for this module