CloudFormation Fundamentals: Terminology

Terminology: Template

A template is a JSON or YAML formatted text file

Input into CloudFormation

Describes end state of Infrastructure (Is a blueprint for building your AWS resources)

CloudFormation will orchestrate the provisioning of the Infrastructure based on the template

Templates are the blueprint, which is provided to CloudFormation to describe our AWS Infrastructure.

Terminology: Stack

When CloudFormation executes a template, it creates a stack (containing those resources described in the template)

To update the resources within a template, you need to update the stack with the new template

A set of related resources as a single unit is called a stack.

In our first lab our template simply contained a single EC2 instance

A template is provided to CloudFormation as we look to create a new set of resources. CloudFormation will pass the template and create a set of resources described in the template. The resources created become a stack in CloudFormation. In other words, the stack is a provisioned instance of the template.

Terminology: Change set

Before updating a stack, you can generate a change set.

A change set allows you to see how the changes will impact your running/existing resources.

This can be very important for live systems as certain changes to existing resources may delete or recreate the resource, rather than simply update the existing resource. It all depends on the nature of the change.

For example, renaming an RDS Instance will:

1) Create a new one
2) Delete the old one

Potentially causing data loss!

Change sets provide visibility to the actions that will be taken

Let's recap: Our template inputs into CloudFormation which creates our stack. In the case of updating the template, the new template is provided to CloudFormation. CloudFormation will provide a **Change set**, which is a summary of the proposed changes. In this example, we are simply going to create an SNS topic within our template. If accepted, this will be executed on the stack and hence create an SNS topic in the stack.

In summary:

A Template is a description of the desired end state of infrastructure

This is provided to CloudFormation (Infrastructure as Code aka the Orchestrator) which will orchestrate the provisioning and configuring of the resources, such as EC2, S3, Lambda functions, CloudFront, or any other service available on AWS, and this is called a stack. A stack is a set of resources created by a template. Prior to performing an update on the stack, CloudFormation can create a Change set to validate the changes that will occur. A Change set is a summary of proposed changes.

CloudFormation Fundamentals: Anatomy of a Template

Let's analyze the following JSON code below in order to discover the sections we can use in a CloudFormation template.

```
{
        "AWSTemplateFormatVersion" : "version date",
        "Description" : "JSON string",
        "Metadata" : { template metadata },
        "Parameters" : { set of parameters },
        "Mappings" : { set of mappings },
        "Conditions" : { set of conditions },
        "Transform" : { set of transforms },
        "Resources" : { set of resources },
        "Outputs" : { set of outputs }
}
```

1) In the above JSON code, the first section is the AWSTemplateFormatVersion, which simply defines a CloudFormation template format version. This is NOT the same as the API or the WSDL version. At the moment there is only one valid value for the version and that is 2010-09-09. This section is **optional** in CloudFormation templates.

2) The next section is the Description, which is a text string that describes this template. This section is also **optional** in CloudFormation templates.
3) The Metadata is a section for objects that provide additional information about the template. This is **optional** as well.
4) Parameters is a section which allows you to pass values into your template at runtime. This is also **optional**.
5) Mappings is a set of keys and values which can operate as a lookup table. Mappings is also **optional.**
6) Conditions is a section where we can place conditions on whether certain resources are created. It is **not required**.
7) The Transform section is where you can specify transforms that CloudFormation will use to process your template. For example, including external snippets into a template or specifying a version of the Serverless Application Model. It is **not required.**
8) Resources is our main section and the only **required** section among the sections mentioned. This defines the stack resources and their properties.
9) Outputs is the section that allows you to return values from your stack.

Conditions and Transforms are advanced topics.

Here is the sample Template in YAML format (as opposed to JSON):

AWSTemplateFormatVersion: “version data”

Description: String

Metadata:

    … template metadata …

Parameters:

    … set of parameters …

Mappings:

    … set of mappings …

Conditions:

    … set of conditions …

Transform:

    … set of transforms …

Resources:

… set of resources …

Outputs:

… set of outputs …

Terminology: Templates.Resources

Let's take a look at how the Resources section is structured in JSON Format. Afterall, it is the only required section

```
{
    "Resources" : {
        "Logical ID" : {
            "Type" : "Resource type",
            "Properties" : {
                … set of properties ..
            }
        }
    }
}
```

1) The "Resources" object contains a list of resources and their properties.
2) The "Logical ID" aka logical identifier for the resource can be used to reference it's resource in other parts of the template. This is **NOT** a physical ID, which is assigned to the resource on provision. Use the physical ID to reference this resource **outside** the template. Use the Logical ID to reference the resource **within** the template.
3) The first field within this object is of "Type". "Type" identifies the type of the resource that we are declaring.

4) In the next field we declare the “Properties” that are appropriate for this type of resource. Each type of resource will contain a list of their own properties. Some are required and some are optional.

Now let’s take a look at the Resource section in YAML format.

Resources:

    Logical ID:

        Type: Resource type

        Properties:

            Set of properties

It is more concise but very similar to the JSON format

Just as before:

The Resources object contains the resources and their Properties.

The Logical ID can be used to reference it’s resource in other parts of the template.

The first field is the Type which identifies the type of the resource that we are declaring.

The next field is the Properties, which declares the appropriate properties for this type of resource. Each type of resource will contain a list of their own properties. Some are required and some are optional.

Let's create a simple template similar to the one we used in the first lab tutorial where we created a single EC2 instance.

```
{
        "Resources" : {
            "MyEC2Instance" : {
                "Type" : "AWS::EC2::Instance",
                "Properties" : {
                    "ImageId" : "ami-43874721",
                    "InstanceType" : "t2.micro",
                }
            }
        }
}
```

1) In the template above, the top level **object** is "Resources"
2) The "Resources" object contains a list of resources and their properties. Inside this object, we have a resource logically named "MyEC2Instance". So "MyEC2Instance" is the Logical ID.
3) The first field within this object is of "Type". We are creating an EC2 instance, hence "AWS::EC2::Instance"
4) In the next field we declare the properties appropriate for this type of resource, which is an EC2 instance. In this case, an EC2 instance can be defined by the "ImageID" which is essentially the AMI ID as well as the "InstanceType", which in this case is a t2.micro

All the template will do is provision a t2.micro EC2 instance that has that given AMI ID.

Let's see what the template looks like in YAML format.

```
Resources:
    MyEC2Instance:
        Type: “AWS::EC2::Instance”
        Properties:
            ImageID: “ami-43874721”
            InstanceType: t2.micro
```

The above YAML template accomplishes the same thing as the JSON template.

The top-level section is Resources. Within the Resources we declare our resource which is logically named MyEC2Instance. Then we have the Type which we have defined as an EC2 instance, hence “AWS::EC2::Instance” Then we’ve got the Properties for the EC2 instance, which includes the ImageID and the InstanceType.

The YAML format is a lot more concise and readable. It avoids a lot of noisey curly brackets that the JSON format has.

Now let’s look at the structure of a template that creates a public readable S3 bucket.

```
Resources:
    MyS3Bucket:
        Type: AWS::S3::Bucket
        Properties:
            BucketName: HelloWorldWebsite777
            AccessControl: PublicRead
            WebsiteConfiguration:
                IndexDocument: index.html
```

In this case, the Logical ID/name is MyS3Bucket. The Type is defined as an AWS::S3::Bucket

The properties are the BucketName which is set to HelloWorldWebsite777

AccessControl which is set to PublicRead

WebsiteConfiguration which is has an IndexDocument of index.html

And that is how we create an S3 bucket with a template.

Now let's look at the structure of a template that creates a security group.

```
Resources:
  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        -IpProtocol: tcp
         FromPort: '22'
         ToPort: '22'
         CidrIp: 0.0.0.0/0
```

The above template declares a security group which opens up TC port 22 for all IPs.

The logical ID is named MySecurityGroup

The Type is AWS::EC2::SecurityGroup

In the Properties section we define the GroupDescription, which is simply to Enable SSH access to port 22.

We describe the Ingress rules, which opens up port 22 to all IPs

We have looks at the high level structure of CloudFormation templates and we have explored the main section, which is the Resources section. We have seen how we can define an EC2 instance, an S3 bucket, and a Security Group.

In the next lesson we are going to start adding on more features to our simple CloudFormation template example where we created an EC2 instance. We will build on that.