Cloud Formation

16 June 2023 22:48

Infrastructure-as-Code (IaC)

1. What is CloudFormation, and Why do we even need this?
Before we look into "what is CloudFormation" first we need to know why do we even need this? Since AWS infrastructure can be created and customized using the AWS dashboard (GUI), CLI, or API. These methods may be able to build an infrastructure quickly as a one off; however, over a long period of time, if we used to create the same infrastructure in a different region to build **Disaster Recovery (DR),** or in a subsidiary AWS account, then those methods would be costly in time and money. Also, in terms of management, modification, and maintenance. So, **CloudFormation helps to create the same environment without any errors.**

*CloudFormation helps to create AWS Infrastructure as Code in **YAML or JSON file format**.*

2. What are the main Core Concepts and Terminologies in CloudFormation?

I) **What is a Template or CloudFormation Templates (CFTs)?**
It's the code that is used to create CloudFormation. AWS CFT describes all AWS resources and their properties in JSON or YAML format. Template extensions can b**e .txt or .json or .yaml.**
You can upload a template using the browser button or directly into the S3 bucket and give the specified URL. Even if you upload it via the browser button it will get stored in the S3 bucket.
When you create a template, **it is not necessary for you to identify AWS resources dependencies.** CloudFormation automatically identifies the resource dependencies and creates them sequentially
Generally, **it's recommended to write a template for each layer of the architecture**; for example, separate templates for networking components, database servers, and web servers.

II) **What is a Stack:**
A stack is created upon a successful execution of a template in CloudFormation. Limited to **2000 stacks per account (as of 2021)**
CloudFormation reports the stack as **CREATE_COMPLETE** when all resources have been created and the appropriate services report creation complete
During the template execution, if CloudFormation is unable to create any resource, the **whole stack creation fails**. When an execution fails it **rolls back all of the execution steps and deletes any resources created during the process**
With the help of ***DeletionPolicy*** you can still maintain part of the AWS resource for future use.
III) What are the reason that Stack Creation get failed?
At the time of creating a stack from a template, AWS CloudFormation only checks for syntax errors in JSON/YAML notation. **It doesn't check whether the IAM user executing the template has sufficient privilege to complete the template execution or not**.
Additionally, **it doesn't check whether any resource creation soft-limits in AWS.**
Insufficient privilege
AWS Soft limits

IV) **How do you get charged for using CloudFormation?**
AWS doesn't charge for using CloudFormation Service. However, it charged for the services that you used in the CloudFormation
Ex: You create a CloudFormation template that is used to create VPC, RDS, and EC2. You only get charged for RDS and EC2. Not for VPC or CloudFormationService

3. What are the main components of the Cloud Formation Template?

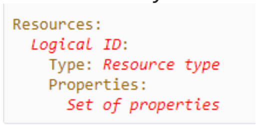
1. **Resources: set of resources (Mandatory)**
2. Parameters: set of parameters
3. AWSTemplateFormatVersion: "version date"
4. Description: String
5. Metadata: template metadata
6. Mappings: set of mappings
7. Conditions: set of conditions
8. Transform: set of transform

9. Outputs: set of outputs

Out of the above 9 sections, the ***Resource*** *section is the only required section to successfully execute an AWS CFT.*

3.1 What are Resources in CFTs?

- The required Resources section declares the AWS resources that you want to include in the stack, such as an Amazon EC2 instance or an Amazon S3 bucket.
- **At least one resource to create and include in stacks**
- You **CAN NOT create Dynamic resources**
- General syntax for creating a resource in CloudFormation



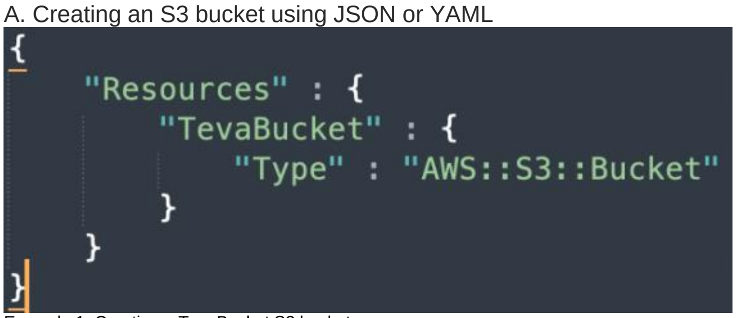
```
Resources:
  Logical ID:
    Type: Resource type
    Properties:
      Set of properties
```

General Syntax for Creating Resource

1. *The logical ID must be alphanumeric (A-Za-z0–9) and* ***unique*** *within the template. Use the logical name to reference the resource in other parts of the template.*
2. *The resource type identifies the type of resource that you are declaring. For example, AWS::EC2::Instance declares an EC2 instance*
3. *Resource properties are additional options that you can specify for a resource*

Sample Examples of Templates for Creating various resources:

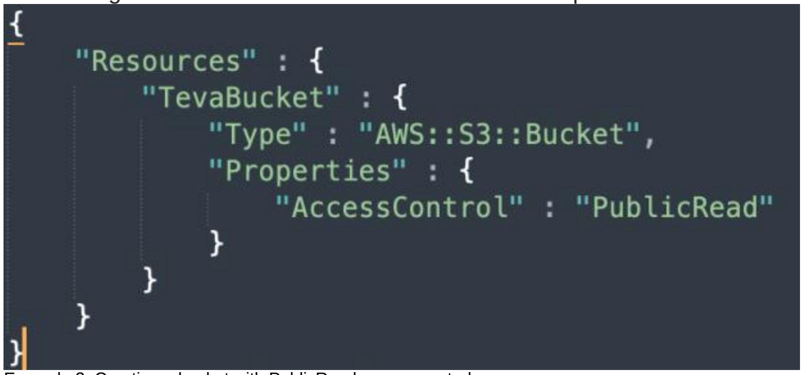
A. Creating an S3 bucket using JSON or YAML



```
{
    "Resources" : {
        "TevaBucket" : {
            "Type" : "AWS::S3::Bucket"
        }
    }
}
```

Example 1: Creating a TevaBucket S3 bucket

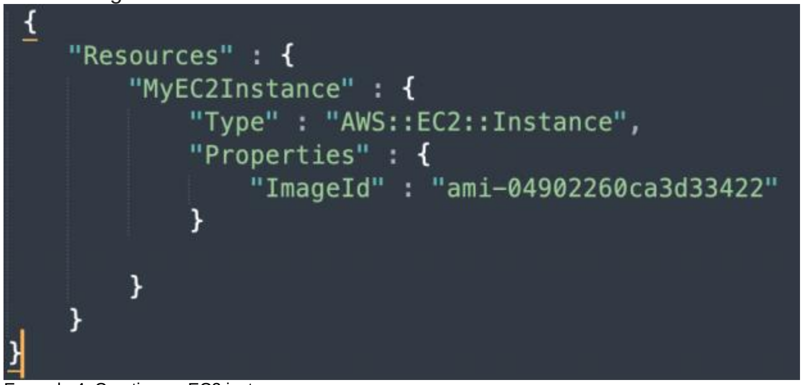
B. Creating an S3 Bucket with Public Read Access Properties



```
{
    "Resources" : {
        "TevaBucket" : {
            "Type" : "AWS::S3::Bucket",
            "Properties" : {
                "AccessControl" : "PublicRead"
            }
        }
    }
}
```

Example 2: Creating a bucket with PublicRead access control

C. Creating an EC2 instance



```
{
    "Resources" : {
        "MyEC2Instance" : {
            "Type" : "AWS::EC2::Instance",
            "Properties" : {
                "ImageId" : "ami-04902260ca3d33422"
            }

        }
    }
}
```

Example 4: Creating an EC2 instance

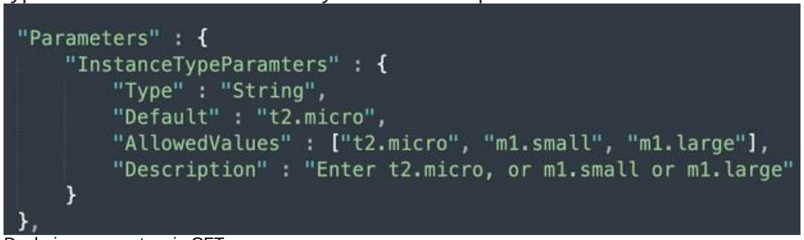
3.2 What are parameters in CFTs?

- Parameters are a way to **provide inputs** to your AWS CloudFormation template
- Values to pass to your template at runtime

- Use the optional Parameters section to customize templates
- Parameters enable you to input custom values to your template each time **you create or update a stack.**
- You can have a maximum of **200** parameters in an AWS CloudFormation template
- If you want to **REUSE** the template think of Parameters

3.2.1 How to Define parameter in CFTs

The following example declares a parameter named InstanceTypeParameter. This parameter lets you specify the Amazon EC2 instance type for the stack to use when you create or update the stack.

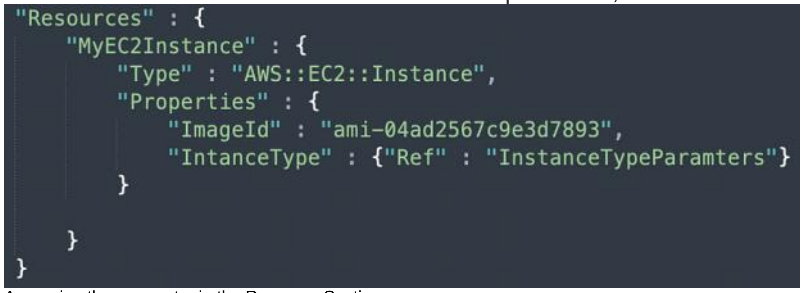


```
"Parameters" : {
    "InstanceTypeParamters" : {
        "Type" : "String",
        "Default" : "t2.micro",
        "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],
        "Description" : "Enter t2.micro, or m1.small or m1.large"
    }
},
```

Declaring parameters in CFTs

3.2.2 How to use the parameters in the template?

You use the **Ref** intrinsic function to reference a parameter, and AWS CloudFormation uses the parameter's value to provision the stack.

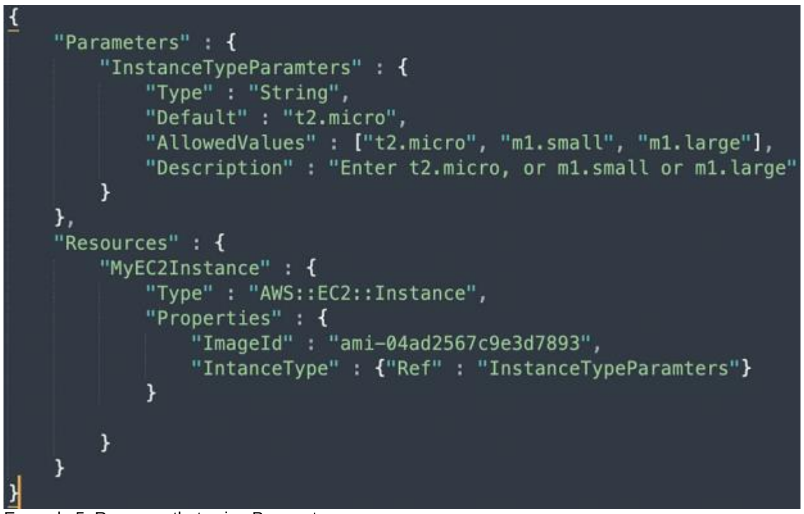


```
"Resources" : {
    "MyEC2Instance" : {
        "Type" : "AWS::EC2::Instance",
        "Properties" : {
            "ImageId" : "ami-04ad2567c9e3d7893",
            "IntanceType" : {"Ref" : "InstanceTypeParamters"}
        }

    }
}
```

Accessing the parameter in the Resource Section

You can reference parameters from the Resources and Outputs sections of the same template.
Complete code segment

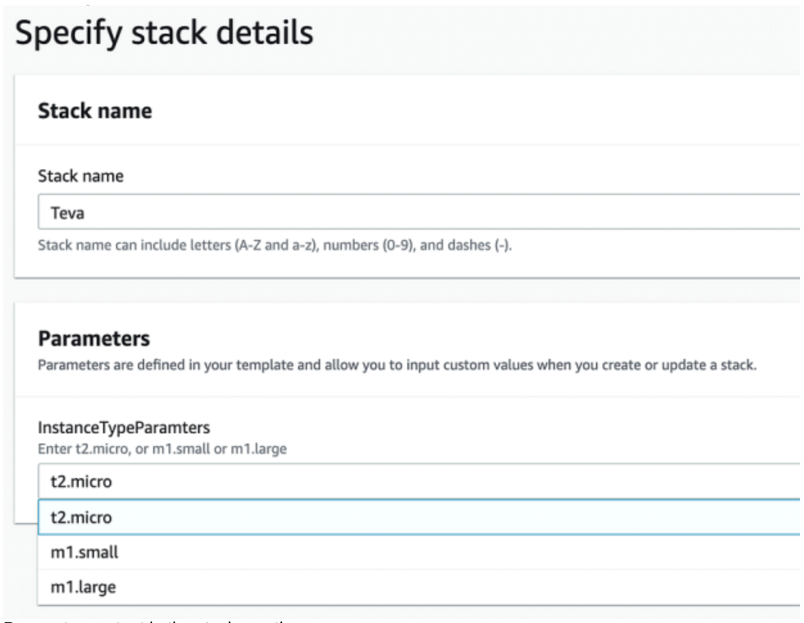


```
{
    "Parameters" : {
        "InstanceTypeParamters" : {
            "Type" : "String",
            "Default" : "t2.micro",
            "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],
            "Description" : "Enter t2.micro, or m1.small or m1.large"
        }
    },
    "Resources" : {
        "MyEC2Instance" : {
            "Type" : "AWS::EC2::Instance",
            "Properties" : {
                "ImageId" : "ami-04ad2567c9e3d7893",
                "IntanceType" : {"Ref" : "InstanceTypeParamters"}
            }

        }
    }
}
```

Example 5: Resource that using Parameters

When you create a stack using the above template in the middle of the stack creation user needs to provide the input parameter as the following:



Specify stack details

Stack name

Stack name: Teva

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

InstanceTypeParamters

Enter t2.micro, or m1.small or m1.large

t2.micro

t2.micro

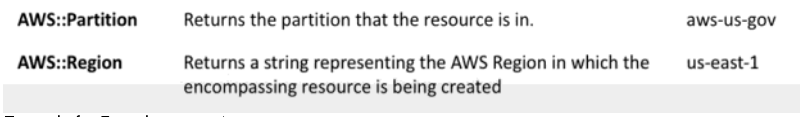
m1.small

m1.large

Parameters output in the stack creation

3.2.3. What are Pseudo parameters?

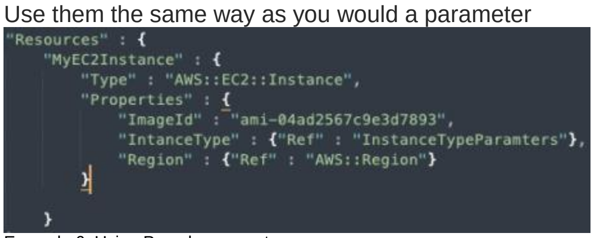
Pseudo parameters are parameters that are predefined by AWS CloudFormation. You don't declare them in your template.



AWS::Partition	Returns the partition that the resource is in.	aws-us-gov
AWS::Region	Returns a string representing the AWS Region in which the encompassing resource is being created	us-east-1

Example for Pseudo parameters

Use them the same way as you would a parameter



```
"Resources" : {
    "MyEC2Instance" : {
        "Type" : "AWS::EC2::Instance",
        "Properties" : {
            "ImageId" : "ami-04ad2567c9e3d7893",
            "IntanceType" : {"Ref" : "InstanceTypeParamters"},
            "Region" : {"Ref" : "AWS::Region"}
        }

    }
```

Example 6: Using Pseudo parameter

3.3 What is AWSTemplateFormatVersion in CFTs?

- The AWSTemplateFormatVersion section (**optional**) identifies the capabilities of the template.
- The latest currently supported version is "2010–09–09". It needs to be defined as string in " ".
- If you don't specify the version the default value gets assigned.
- The template format version is **NOT the same as** the API or WSDL version

3.4 What is Description in CFTs?

- The Description section (optional) enables you to include comments about your template.
- This helps other developers to understand the purpose of this template
- The value for the description declaration must be a literal string that is between **0 and 1024 bytes in length.**

3.5 What is Metadata in CFTs?

- You can use the optional Metadata section to include arbitrary JSON or YAML objects that provide details about the template
- It supports metadata keys, which enable us to retrieve the defined configuration or settings in the **Resources** section.
- Some AWS CloudFormation features retrieve settings or configuration information that you define in the Metadata section.
- You define this information in the following AWS CloudFormation-specific metadata keys:

a. AWS::CloudFormation::Init

Defines configuration tasks for the **cfn-init helper script**

This is a more powerful and feature-rich way to provide configuration and perform **instance bootstrapping from within CloudFormation**.

b. AWS::CloudFormation::Interface

Defines the grouping and ordering of input parameters when they are displayed in the AWS CloudFormation console.

c. AWS::CloudFormation::Designer

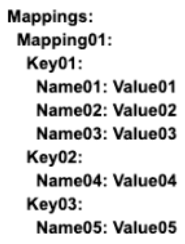
Describes how your resources are laid out in AWS CloudFormation Designer (Designer)

3.6 What is Mappings in CFTs?

- The optional Mappings section matches a key to a corresponding set of named values.
- For example, if you want to set values based on a region, you can create a mapping that uses the region name as a key and

contains the values you want to specify for each specific region

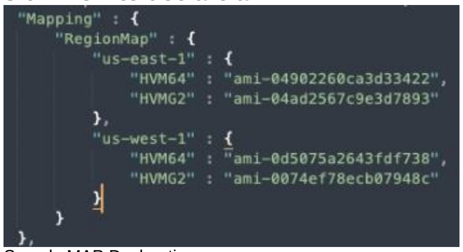
- You use the **Fn::FindInMap** intrinsic function to retrieve values in a map.
- **!FindInMap [MapName, TopLevelKey, SecondLevelKey]**



```
Mappings:
 Mapping01:
  Key01:
   Name01: Value01
   Name02: Value02
   Name03: Value03
  Key02:
   Name04: Value04
  Key03:
   Name05: Value05
```

Basic template of Mapping in YAML format

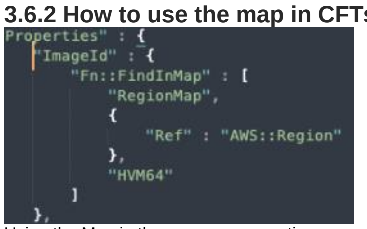
3.6.1 How to declare a MAP?



```
"Mapping" : {
    "RegionMap" : {
        "us-east-1" : {
            "HVM64" : "ami-04902260ca3d33422",
            "HVMG2" : "ami-04ad2567c9e3d7893"
        },
        "us-west-1" : {
            "HVM64" : "ami-0d5075a2643fdf738",
            "HVMG2" : "ami-0074ef78ecb07948c"
        }
    }
},
```

Sample MAP Declaration

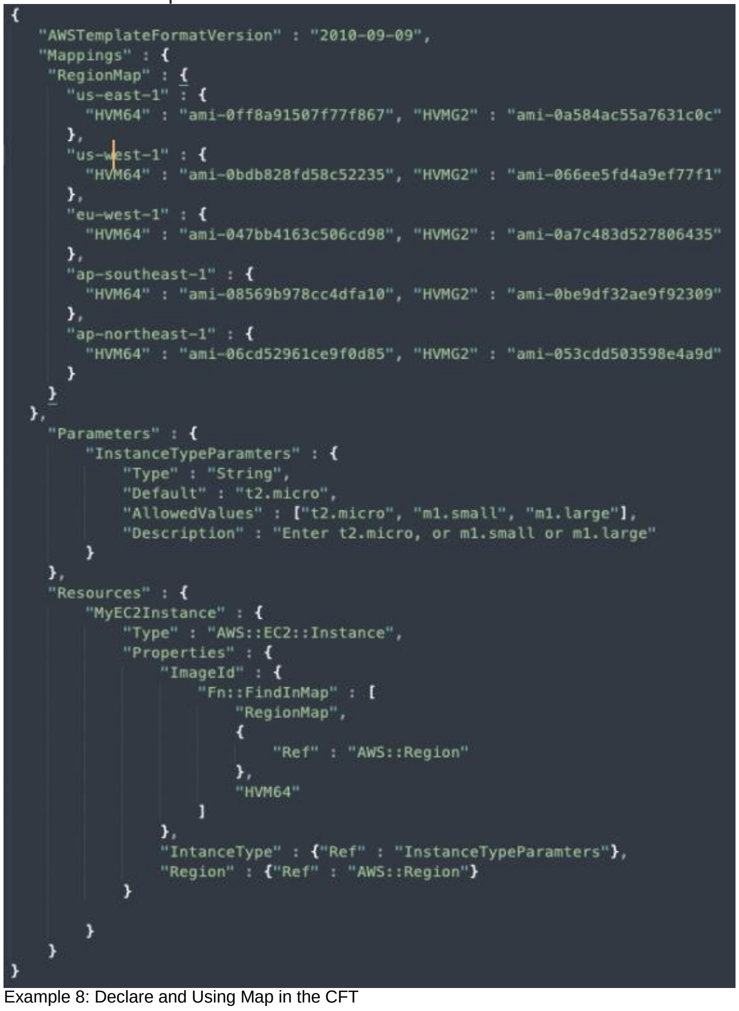
3.6.2 How to use the map in CFTs?



```
Properties" : {
  "ImageId" : {
      "Fn::FindInMap" : [
          "RegionMap",
          {
              "Ref" : "AWS::Region"
          },
          "HVM64"
      ]
  },
```

Using the Map in the resource properties

Here is the complete code:



```
{
   "AWSTemplateFormatVersion" : "2010-09-09",
   "Mappings" : {
    "RegionMap" : {
      "us-east-1" : {
        "HVM64" : "ami-0ff8a91507f77f867", "HVMG2" : "ami-0a584ac55a7631c0c"
      },
      "us-west-1" : {
        "HVM64" : "ami-0bdb828fd58c52235", "HVMG2" : "ami-066ee5fd4a9ef77f1"
      },
      "eu-west-1" : {
        "HVM64" : "ami-047bb4163c506cd98", "HVMG2" : "ami-0a7c483d527806435"
      },
      "ap-southeast-1" : {
        "HVM64" : "ami-08569b978cc4dfa10", "HVMG2" : "ami-0be9df32ae9f92309"
      },
      "ap-northeast-1" : {
        "HVM64" : "ami-06cd52961ce9f0d85", "HVMG2" : "ami-053cdd503598e4a9d"
      }
    }
  },
    "Parameters" : {
        "InstanceTypeParamters" : {
            "Type" : "String",
            "Default" : "t2.micro",
            "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],
            "Description" : "Enter t2.micro, or m1.small or m1.large"
        }
    },
    "Resources" : {
        "MyEC2Instance" : {
            "Type" : "AWS::EC2::Instance",
            "Properties" : {
                "ImageId" : {
                    "Fn::FindInMap" : [
                        "RegionMap",
                        {
                            "Ref" : "AWS::Region"
                        },
                        "HVM64"
                    ]
                },
                "IntanceType" : {"Ref" : "InstanceTypeParamters"},
                "Region" : {"Ref" : "AWS::Region"}
            }

        }
    }
}
```

Example 8: Declare and Using Map in the CFT

3.7 What are Conditions in CFTs?

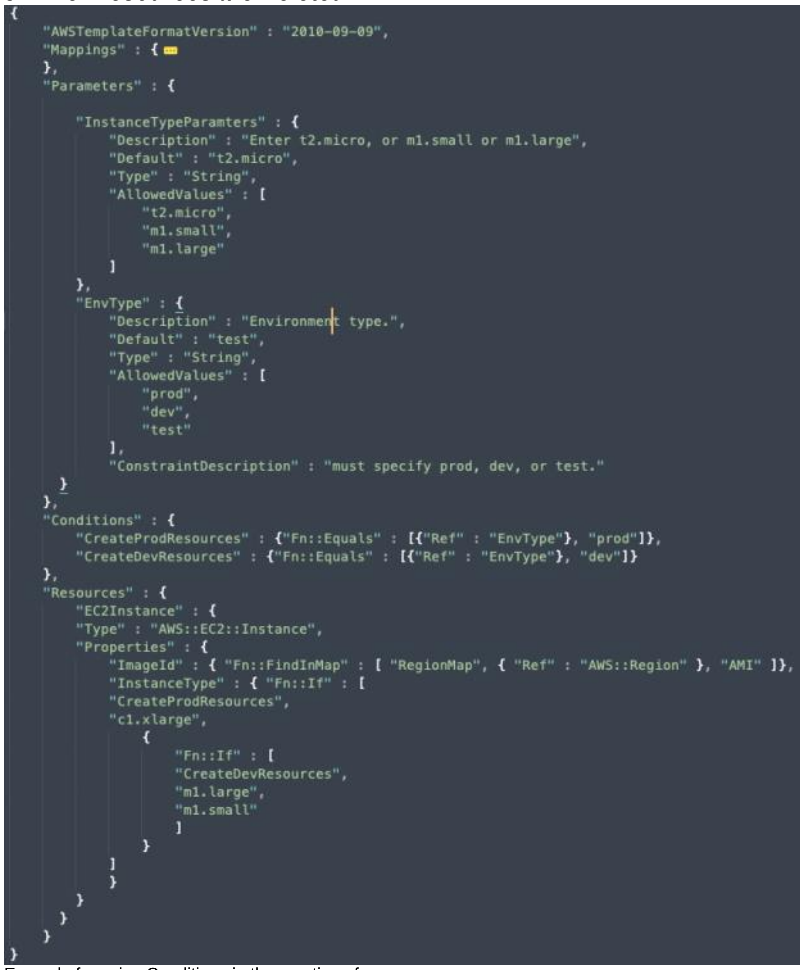
- The optional Conditions section contains statements that define the circumstances under which entities are created or configured.
- For example, you can create a condition and then associate it with a resource or output so that AWS CloudFormation only creates the resource or output if the condition is true.
- Similarly, you can associate the condition with a property so that AWS CloudFormation only sets the property to a specific value if the condition is true. If the condition is false, AWS CloudFormation sets the property to a different value that you specify.
- In your template, you can add an EnvironmentType input parameter, which accepts either **prod** or **test** as inputs. For the production environment, you might include Amazon EC2 instances with certain capabilities; however, for the test environment, you want to use reduced capabilities to save money. With conditions, you can define which resources are created and how they're configured for each environment type.
- To conditionally specify the property, use the conditional function such as: Fn::And, Fn:Equals, Fn::If, Fn::Not or Fn::Or
- In order to create a resource based condition, it is essential to specify the statement in the last three different sections in a template:
- **a. Parameter**: You can define the input value that evaluates whether the input condition is True or False
- **b. Resource**:
- **c. Output**:

*Conditions in a template can be modified **only:***

1. When resources are Added

2. When resources are Modified

3. When resources are Deleted



```
{
    "AWSTemplateFormatVersion" : "2010-09-09",
    "Mappings" : { 
    },
    "Parameters" : {

        "InstanceTypeParamters" : {
            "Description" : "Enter t2.micro, or m1.small or m1.large",
            "Default" : "t2.micro",
            "Type" : "String",
            "AllowedValues" : [
                "t2.micro",
                "m1.small",
                "m1.large"
            ]
        },
        "EnvType" : {
            "Description" : "Environment type.",
            "Default" : "test",
            "Type" : "String",
            "AllowedValues" : [
                "prod",
                "dev",
                "test"
            ],
            "ConstraintDescription" : "must specify prod, dev, or test."
       }
    },
    "Conditions" : {
        "CreateProdResources" : {"Fn::Equals" : [{"Ref" : "EnvType"}, "prod"]},
        "CreateDevResources" : {"Fn::Equals" : [{"Ref" : "EnvType"}, "dev"]}
    },
    "Resources" : {
        "EC2Instance" : {
        "Type" : "AWS::EC2::Instance",
        "Properties" : {
            "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},
            "InstanceType" : { "Fn::If" : [
            "CreateProdResources",
            "c1.xlarge",
                {
                    "Fn::If" : [
                    "CreateDevResources",
                    "m1.large",
                    "m1.small"
                    ]
                }
            ]
            }
        }
       }
    }
}
```

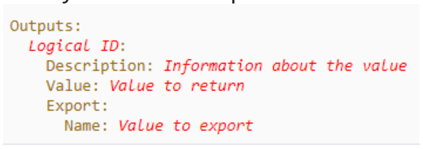
Example for using Conditions in the creation of resource

3.8 What is Transforms in CFTs?

- The optional Transform section specifies **one or more macros that AWS CloudFormation uses to process your template**.
- The Transform section builds on the simple, declarative language of AWS CloudFormation with a powerful macro system.
- **1. AWS::Serverless**: This specify version of an AWS Serverless Application Model (SAM) It helps to deploy AWS Lambda functions
- **2. AWS::Include**: This helps to include a separate template snippets

3.9 What is Outputs in CFTs?

- The optional Outputs section declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console.
- Syntax of the output



```
Outputs:
  Logical ID:
    Description: Information about the value
    Value: Value to return
    Export:
      Name: Value to export
```

Description and export are optional parameters

- You can't delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack
- Note: Export key is used to export the value
- The values are getting imported in other stacks using **FN::ImportValue** function

4. What is CloudFormer

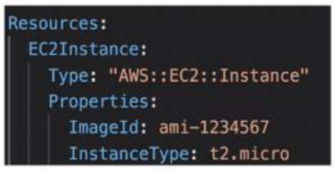
It can automatically generate a CFT from an existing AWS resource in your AWS account. Stores the CFT in S3 bucket you specified

5. What are Intrinsic Functions in CFTs?

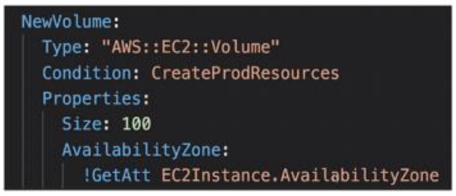
Use intrinsic functions in your template to **assign values to properties that are not available until runtime**.

Example:

1. **Fn::Ref**: The intrinsic function Ref returns the value of the specified parameter or resource.
2. When you specify a **parameter's logical name**, it returns the **value of the parameter.**
3. When you specify a **resource's logical name**, it returns a value that you can typically use to refer to that resource, such as a **physical ID**.
4. The shorthand for this in YAML is !Ref
5. **FN::GetAtt:** The GetAtt intrinsic function returns the value of an attribute from a resource in the template.
6. Attributes are attached to any resources you create



```
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: ami-1234567
      InstanceType: t2.micro
```



```
NewVolume:
  Type: "AWS::EC2::Volume"
  Condition: CreateProdResources
  Properties:
    Size: 100
    AvailabilityZone:
      !GetAtt EC2Instance.AvailabilityZone
```

FN::FindInMap: To return a named value from a specific key

!FindInMap [MapName, TopLevelKey, SecondLevelKey]

Fn::ImportValue: Import values that are exported in the other templates

Fn::Join: Join values with delimiter

! join [deliter, [comma-delimited values]]

Example: To create "a:b:c"

!Join [":", [a, b, c]]

Fn::Sub: Substitute values from a text

String must contain **${**variableName**}** and will substitute them

Condition Functions:

Only to create resources based on the condition. We can use the following logical operations.

(Fn::And, Fn::If, Fn::Not, Fn::Equals, Fn::Or) : it's covered in the above

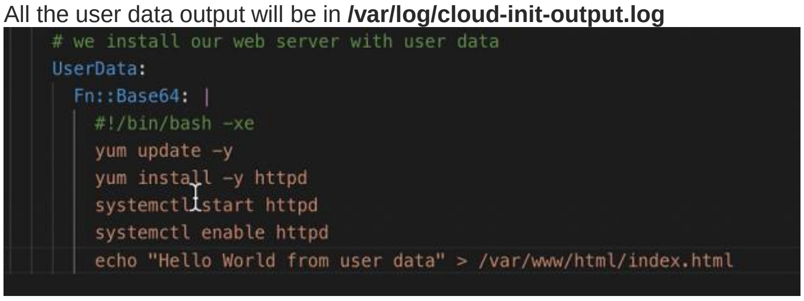
Fn::GetAZs: Returns an array of availability zone strings

Fn::Select : Receives an array and an index to return a single element

6. How to Pass User Data to EC2 Instances using CloudFormation?

Method 1: Using Fb::Base64 : We can pass the entire script for the user data through the function **Fn::Base64**

All the user data output will be in **/var/log/cloud-init-output.log**



```
  # we install our web server with user data
  UserData:
    Fn::Base64: |
      #!/bin/bash -xe
      yum update -y
      yum install -y httpd
      systemctl start httpd
      systemctl enable httpd
      echo "Hello World from user data" > /var/www/html/index.html
```

CloudFormation : 1:20 Mins
Terraform : 40-50 Mins
API Gateway
Lambda
CloudFront
AWS Organization
Single Sign On (Microsoft Azure Integration with AWS)
Kinesis