WHAT IS AWS CLOUDFORMATION?

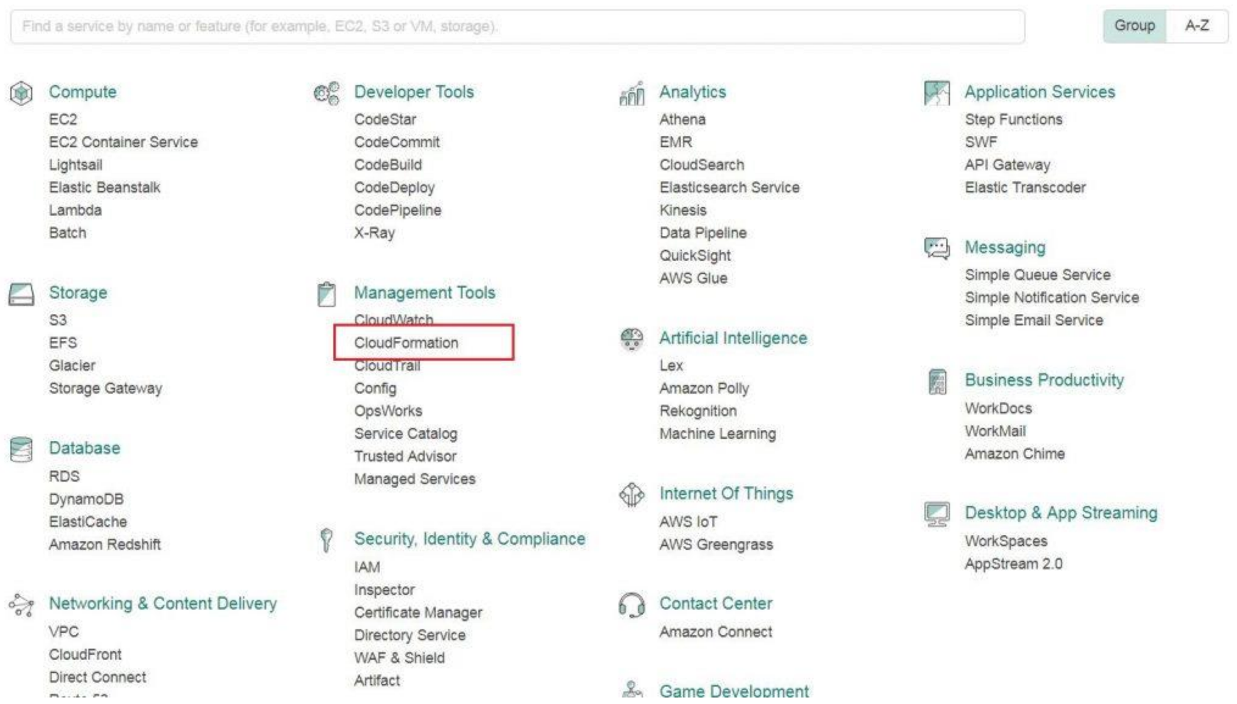
AWS CloudFormation is a service that allows the configuring of AWS resources so that developers can focus more on application development, rather than spending time in creating and managing the infrastructure to run an application. It provides a way for the management, provisioning, and updating of AWS resources in an orderly and predictable fashion. In other words, we can say it is a way to enable **"AWS Resource Automation."**

AWS CloudFormation provides an easy way to create and maintain AWS-related resources like Elastic Beanstalk, EC2 instance, RDS, security groups, and the dependencies between them. It provides a way to define AWS resources, their dependencies, and the connection between the two resources in one template. The AWS formation engine uses that template to create a ready-to-use environment in the cloud. It also provides the flexibility to pass runtime parameters while creating the resources in AWS.

The AWS CloudFormation template is a text file defined in **JavaScript Object Notation (JSON)** format. It also supports YAML format. Cloud formation not only provides a way to create the resources but also offers different ways to configure them.

Pre-requisites: First of all, before using an AWS resource like cloud formation, we need an Amazon Web Services account.

To sign-up for an AWS account, open http://aws.amazon.com/ and then choose "Create an AWS Account." From there, follow the online instructions.



We can deploy the CloudFormation template using:

- CloudFormation console
- AWS Command Line Interface
- Exposed REST APIs

To proceed further, we need to identify the resources that our application requires. AWS provides a lot of built-in templates to use, while also providing the flexibility to define custom templates.

CLOUDFORMATION TEMPLATE:

It is required that CloudFormation follows a pre-defined AWS template. We must define resources in a manner so that AWS identifies these resources and dependencies between them, and create the required infrastructure.

Let’s discuss the template structure and its description to determine its usage and format in AWS CloudFormation.

TEMPLATE STRUCTURE:

An AWS CloudFormation template has six top-level sections, which must be defined in the order given below:

- AWSTemplateFormatVersion
- Description
- Metadata
- Parameters
- Mappings
- Conditions
- Resources
- Output

The AWS template uses JSON format to define the template structure.

```
{
"AWSTemplateFormatVersion":"Version Date",
"Description":"A description of template",
"Metadata":{
},
"Parameters":{
},
"Mappings":{
},
"Conditions":{
},
"Resources":{
},
"Output":{
}
}
```

SECTIONS AND THEIR USES:

AWSTemplateFormatVersion

AWSTemplateFormatVersion is an optional section that takes a literal string value. If this section is present, then it must be the first section of the template. If this section is missing, then AWS automatically takes the latest value defined for it. Currently, it is **"2010-09-09."**

Description

The description is a text line that describes your template. It must follow the format version section.

```
{
"AWSTemplateFormatVersion":"2010-09-09",
"Description":"This template creates infrastructure in AWS cloud"
}
```

Metadata

Metadata is an optional section used to provide the details of the template. In this section, we can define the details about the different resources we are using, like the database we are configuring, the instance we are using, and so on. It gives an inside view of the infrastructure that will be created using this template, without seeing the entire formation template to get the idea.

```
{
"Metadata":{
"Instances":{
"Description":"Information about the instances"
},
"Databases":{
"Description":"Information about the databases"
}
}
}
```

Parameters

Parameters is an optional section used to pass parameters to the template variables at runtime. We can refer to these parameters while defining resources or inn the output section of the template.

```
{
"Parameters":{
"InstanceType":{
"Description":"EC2 instance type",
"Type":"String",
"Default":"t2.micro",
"AllowedValues":[
"t1.micro","t2.micro","t2.small","t2.medium","m1.small",
"m1.medium","m1.large","m1.xlarge","m2.xlarge","m2.2xlarge",
"m2.4xlarge","m3.medium","m3.large","m3.xlarge","m3.2xlarge"
],
"ConstraintDescription":"Provide a valid EC2 instance type."
}
}
}
```

In the above JSON, we define a parameter **"InstanceType"** that will be passed at runtime and used to identify an EC2 instance type. Here we provide an array of instance types (**AllowedValues)** and a default value (**Default)** of the instance. CloudFormation will check the value provided at runtime by the user against this array. This parameter will only take one value that is defined and if it is not provided, the default value (i.e. "**t2.micro**") will be used. We can use this value to refer this parameter in any section.

```
{"Ref" : "InstanceType"}
```

Mappings

Mappings is an optional section used to define named-value pairs. For example, if we want to set a named-value pair by region name, then we can provide a mapping. Here the region name would be a key and provide a value for that key. Using this mapping, the user can choose their region to create the resources.

Note: We can't use parameters, pseudo parameters, or intrinsic function in mapping section. Keys and values in the mapping section must be a literal string.

```
{
"Mappings":{
"RegionMap":{
"us-east-1":{ "32":"ami-6411e20d"},
"us-west-1":{ "32":"ami-c9c7978c"},
"eu-west-1":{ "32":"ami-37c2f643"},
"ap-southeast-1":{ "32":"ami-66f28c34"},
"ap-northeast-1":{ "32":"ami-9c03a89d"}
}
}
}
```

The above example defines a Mapping to map a 32-bit Image ID to a user region. We can retrieve a value from this map using the **"Fn::FindInMap"** intrinsic function, seen below:

```
{
"ImageId":{
"Fn::FindInMap":["RegionMap",{"Ref":"AWS::Region"},"32"]
}
}
```

Suppose if you are working in the "**us-west-1**" region and it returns the ImageID as **"ami-c9c7978c."** We can correlate this with Map in Java or in other programming languages.

Conditions

This section defines the conditions that will be used by the CloudFormation. It is like **"If"** in any programming language. We can combine multiple conditions with the **COMMA(,)** delimiter. Conditions are evaluated with the input parameter that we defined in the "Parameters" section.

We might use the same template for multiple environments like testing and production, and from the environment, we decide which resource should be created. Like in a production environment, we want a large EC2 instance, but in a testing environment, we may need lesser capabilities. We can use these conditions in the "Resource" section when defining resources.

We can use some intrinsic functions to define the conditions. Some are:

- Fn::And
- Fn::If
- Fn:Not
- Fn::Or
- Fn::Equals

```
{
"Conditions":{
"CreateProdResources":{
"Fn::Equals":[ {"Ref":"EnvType"},"production"]
}
}
}
```

Resources

Resources is a **required** section of any CloudFormation template. This section is used to define the resources that are required for our infrastructure. Most of all, this section is the only required section in any template. We must define all of the resources separately.

```
"Resources":{
"LogicalID":{
"Type":"Resource Type",
"Properties":{
Set of properties
}
}
}
```

Logical ID

This is an alphanumeric key, and it must be unique within the resource section because it plays a vital role within a template. A resource is identified by this ID and we can reference this resource using its ID within the template.

Type

The type is used to declare the type of resource, i.e. "AWS::EC2::Instance." It is used by the CloudFormation engine to create the resource of a defined type.

Properties

This is an additional option to define the resource properties. This is an optional section.

Note: We can use conditions, or we can also define some attributes like **"DependsOn"** to instruct the CloudFormation engine to define the sequence of the resource creation.

```
{
"Resources":{
"MyInstance":{
"Type":"AWS::EC2::Instance",
"Properties":{
"AvailabilityZone":"us-west-2b",
"ImageId": { "Fn::FindInMap" : [ "RegionMap", { "Ref" :   "AWS::Region" }, { "Fn::FindInMap" :
[    "AWSInstanceType2Arch",                         { "Ref" : "EC2InstanceType" }, "Arch" ] } ] },
}
}
}
}
```

In the above example, "**MyInstance**" is the logical ID of the resource. We can provide any shell script to the EC2 instance that will run when the instance is ready and configure any application in it. Therefore, we are required to define all of the resources in this section that are needed.

Output

This section is used to get output values from the CloudFormation engine, i.e. EC2 instance physical ID. The output will be shown on the CloudFormation console. This is also an optional segment of the template.

```
{
"Outputs":{
"InstanceID":{
"Description":"The EC2 Instance ID",
"Value":{
"Ref":"MyInstance"
}
}
}
}
```

CONCLUSION

By combining all the sections, or by using only the Resource section, we can create and define a CloudFormation template that will be used by the CloudFormation engine to create an infrastructure and configure resources. In addition to creating or configuring, we can use templates to provide user data in the resource section so that while instantiating any EC2 instance, it will also install any particular server, like Apache Tomcat. Other ways to configure these resources, like the Elastic Beanstalk script, the AWS code deploy, etc., can also be used.