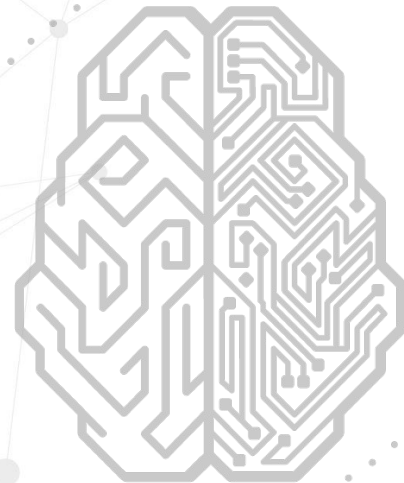




# AWS CloudFormation



# What's in it for you

AWS CloudFormation Deepdive	
S.NO.	AGENDA
1	Why use AWS CloudFormation?
2	What is CloudFormation?
3	How does AWS CloudFormation work?
4	AWS CloudFormation concepts:
5	CloudFormation access control
6	Demo - LAMP stack on EC2 Instance
7	Advanced concepts
8	Demo



# Speakers



**Sanchit Jain**

Lead Architect - AWS at Quantiphi  
AWS APN Ambassador

**FOLLOW ME**



# Why use AWS CloudFormation?

## Support for Different Resources:

AWS CF assists a smorgasbord of resources, which lets you make a highly reliable, available, and scalable or upgradeable AWS infrastructure to cater to your specific application requirements.

## Easy to use:

With CloudFormation, it's easy to classify and station a suite of the secure platform's resources. It allows you to describe any contingencies or special yardsticks to pass in during runtime. You may employ any of the several CloudFormation sample templates verbatim or as a beginning point.

## Flexible and Declarative:

To erect the infrastructure you require, you list out the platform's resources, interconnections, and configuration values you want the template to have and then permit CloudFormation to take care of the rest with some basic clicks in the console

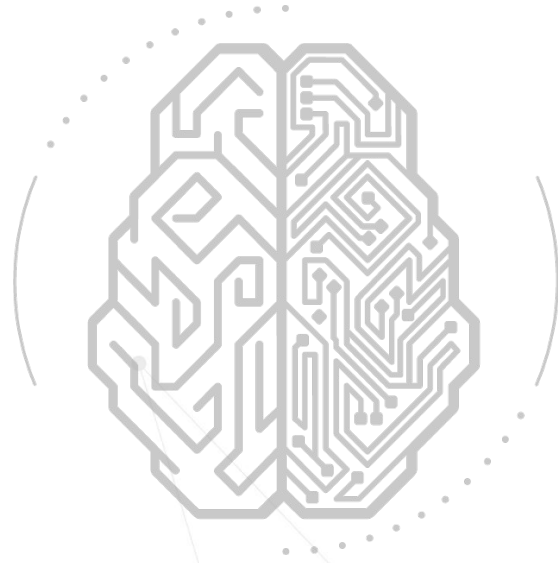
## Customization through parameters:

You may use parameters for customizing specific template aspects at run time. For instance, you could pass the Amazon EC2 instance types, Amazon EBS volume size, RDS database size, server port numbers, and database to the platform when creating a stack.

## Drag and drop UI to visualize and edit:

AWS CloudFormation Designer offers a template diagram with icons denoting the Amazon platform's resources and arrow signs indicating relationships. You could create and modify templates using the interface and then alter template details with the help of the inbuilt JSON text editor.

# What is AWS CloudFormation?

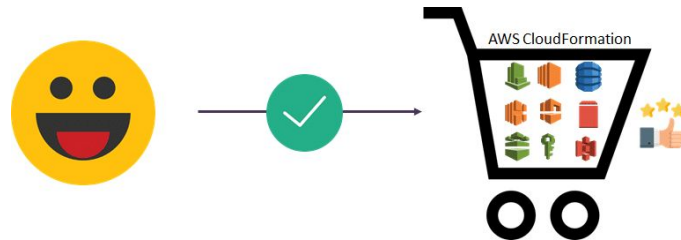


# What is AWS CloudFormation?

AWS CloudFormation provides users a simple way to create and manage a collection of AWS resources by provisioning and updating them in an orderly and predictable way



Create and manage AWS resources



In simple terms, it allows you to create and model your infrastructure and applications without having to perform manual actions

# What is AWS CloudFormation?

AWS CloudFormation provides a simple way to create and manage a collection of AWS resources by provisioning and updating them in an orderly and predictable way



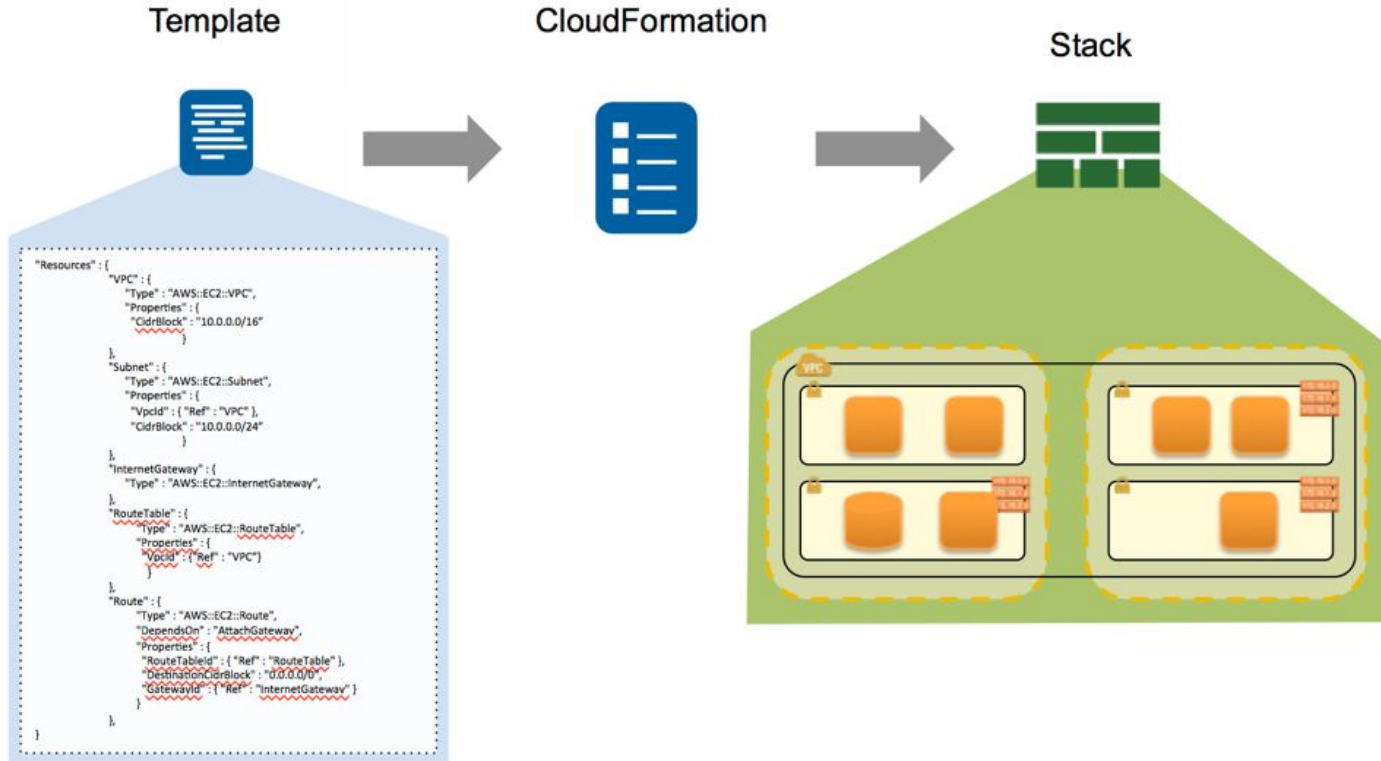
With AWS CloudFormation, Expedia is able to deploy and easily manage its entire front and backend AWS resources into its cloud environment



Create and manage AWS resources

In simple terms, it allows you to create and model your infrastructure and applications without having to perform manual actions

# What is AWS CloudFormation?





# What is AWS CloudFormation?



All the resources required by user in an application can be deployed easily using templates

Also, you can reuse the template to replicate your infrastructure in multiple environments

To make the templates reusable, use the parameters, mappings, and conditions sections, in the template so that you can customize your stacks when you create them

# How AWS CloudFormation work?



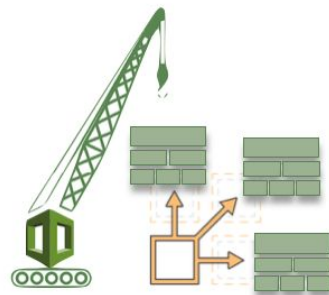
*Code your infrastructure from scratch with the CloudFormation template language, in either YAML or JSON format, or start from many available sample templates*



*Check out your template code locally, or upload it into an S3 bucket*

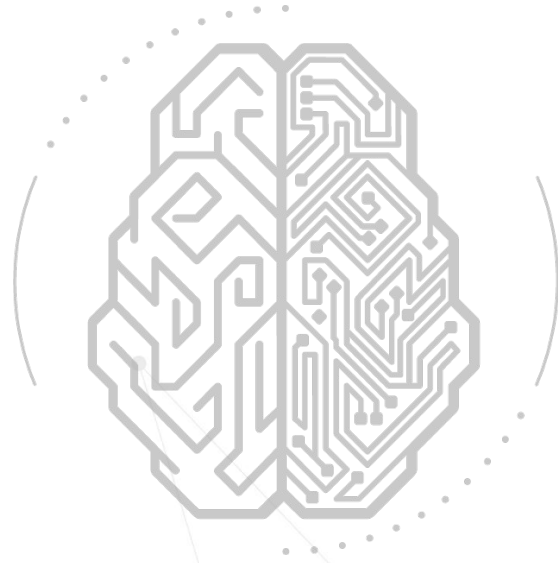


*Use AWS CloudFormation via the browser console, command line tools or APIs to create a stack based on your template code*



*AWS CloudFormation provisions and configures the stacks and resources you specified on your template*

# AWS CloudFormation Concepts



# AWS CloudFormation Concepts



Template

JSON/YAML formatted file

Parameter definition

Resource creation

Configuration actions



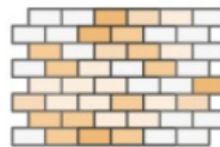
CloudFormation

Framework

Stack creation

Stack updates

Error detection and rollback



Stack

Configured AWS resources

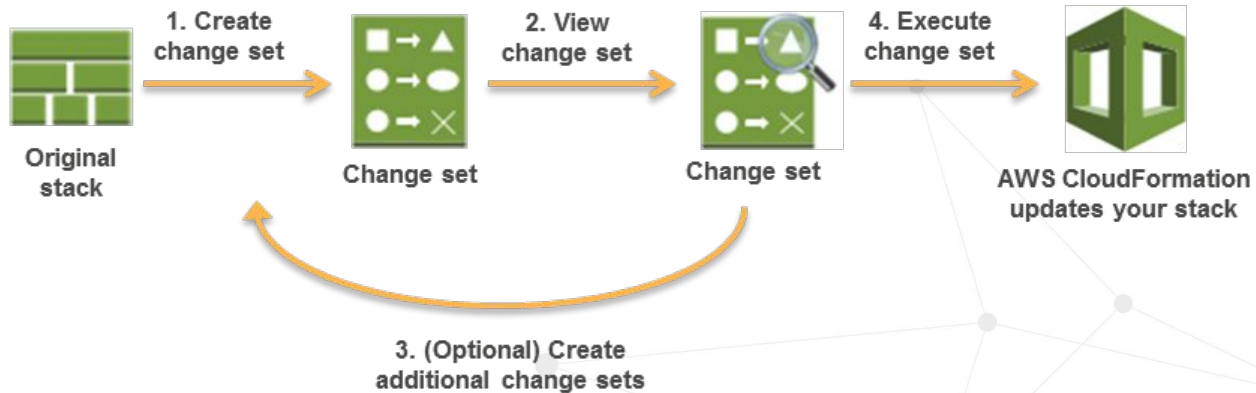
Comprehensive service support

Service event aware

Customizable

# Templates

- A template in AWS CloudFormation is a formatted text file in JSON or YAML language that describes your AWS infrastructure
- To create, view and modify templates you can use AWS CloudFormation Designer or any text editor tool



# Templates

- A template in AWS CloudFormation is a formatted text file in JSON or YAML language that describes your AWS infrastructure
- To create, view and modify templates you can use AWS CloudFormation Designer or any text editor tool
- Templates in AWS CloudFormation consists of 9 main objects



Format version



Description



Metadata



Parameters



Mappings



Conditions



Transform



Resources



Outputs

# Template Structure



```
{
  "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
  }
}
```

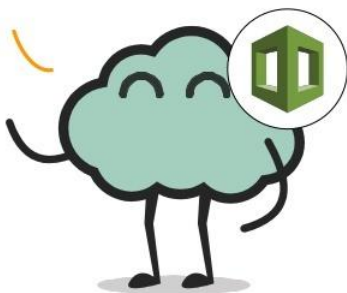
JSON

```
---
AWSTemplateFormatVersion: "version date"
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
```

YAML

# Template Structure

Now, let's discuss each and every object of template structure



```
{
  "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
  }
}
```

JSON

```
---
AWSTemplateFormatVersion: "version date"
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
}
```

YAML



# Template Structure

```
1 AWSTemplateFormatVersion: 2010-09-09
2 Description: >=
3 Launch an EC2 Instance running apache and expose
4 default page to the internet. Harcoded to work
5 only with US-EAST-1 (N. Virginia)
6 Parameters:
7   InstanceType:
8     Description: WebServer EC2 instance type
9     Type: String
10    Default: t2.micro
11    AllowedValues:
12      - t2.nano
13      - t2.micro
14 Resources:
15   WebServer:
16     Type: 'AWS::EC2::Instance'
17     Properties:
18       Tags:
19         -
20           Key: Name
21           Value: Apache Default WebServer
22       InstanceType: !Ref InstanceType
23       # You shouldn't hardcode, just show here for example
24       ImageId: 'ami-0b898040803850657'
25       SecurityGroupIds:
26         - !GetAtt SecurityGroup.GroupId
27       UserData:
28         'Fn::Base64':
29           !Sub |
30             #!/usr/bin/env bash
31             su ec2-user
32             sudo yum install httpd -y
33             sudo service httpd start
34   SecurityGroup:
35     Type: 'AWS::EC2::SecurityGroup'
36     Properties:
37       GroupDescription: Enable internet users to access.
38       SecurityGroupIngress:
39         - IpProtocol: tcp
40           FromPort: 80
41           ToPort: 80
42           CidrIp: 0.0.0.0/0
43 Outputs:
44   PublicIp:
45     Value: !GetAtt WebServer.PublicIp
46
```

## Template Sections

- MetaData** Additional information about the template
- Description** A description of what this template is suppose to do
- Parameters** Values to pass to your template at runtime
- Mappings** A lookup table. Maps keys to values so you change your values to something else
- Conditions** Whether resources are created or properties are assigned
- Transform** Applies macros (like applying a mod which change the anatomy to be custom)
- Resources\*** A resource you want to create eg. IAM Role, EC2 Instance, Lambda, RDS
- Outputs** Values that returned eg. an ip-address of new server created.

CloudFormation Templates **requires** you to **at least list one resource**.

# Metadata

Metadata can be used in the template to provide further information using JSON or YAML objects

```
Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      - Label:
          default: 'Amazon EC2
            Configuration'
        Parameters:
          - InstanceType

    ParameterLabels:
      InstanceType:
        default: 'Type of EC2 Instance'
```



Json/YAML

# Mapping

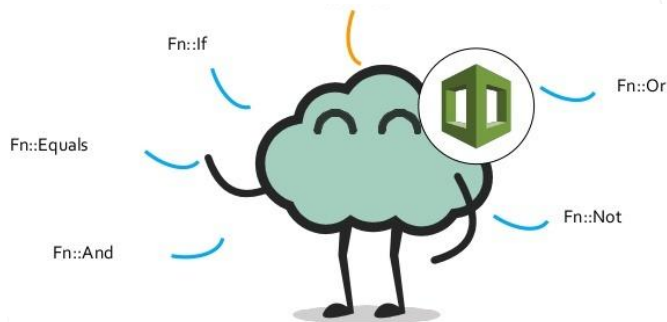
- For Example : Based on a region, you can set values. In a template, you can create a mapping that uses a key and holds the values that you want to specify for each specific region

```
{
  ...
  "Mappings" : {
    "RegionMap" : {
      "us-east-1" : { "32" : "ami-6411e20d", "64" : "ami-7a11e213" },
      "us-west-1" : { "32" : "ami-c9c7978c", "64" : "ami-cfc7978a" },
      "eu-west-1" : { "32" : "ami-37c2f643", "64" : "ami-31c2f645" }
    },
    "Resources" : {
      "myEC2Instance" : {
        "Type" : "AWS::EC2::Instance",
        "Properties" : {
          "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "32"] },
          "InstanceType" : "m1.small"
        }
      }
    }
  }
}
```

# Conditions

- Conditions can be used when you want to reuse the templates by creating resources in different context
- In a template, during stack creation, all the conditions in your template are evaluated
- All resources that are associated with a true condition are created and the invalid conditions are ignored automatically

You can use ***intrinsic functions*** to define conditions



# Conditions

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "EnvType": {
      "Description": "Environment type.",
      "Default": "test",
      "Type": "String",
      "AllowedValues": ["prod", "test"],
      "ConstraintDescription": "must specify prod or test."
    },
  },
  "Conditions": {
    "CreateProdResources": {
      "Fn::Equals": [
        {
          "Ref": "EnvType"
        },
        "prod"
      ]
    },
  },
  "Resources": {
    "EC2Instance": {
      "Type": "AWS::EC2::Instance",
      "Properties": {
        "ImageId": "ami-0ff8a91507f77f867"
      }
    }
  }
}
```

# Transform

- Transform build a simple declarative language for CloudFormation and enables reuse of template components
- Here, you can declare a single transform or multiple transforms within a template

*// Start of processable content for MyMacro and AWS::Serverless*

*Transform:*

- MyMacro
- 'AWS::Serverless'

*Resources:*

*WaitCondition:*

*Type: 'AWS::CloudFormation::WaitCondition'*

*MyBucket:*

*Type: 'AWS::S3::Bucket'*

*Properties:*

*BucketName: MyBucket*

*Tags: [{"key": "value"}]*

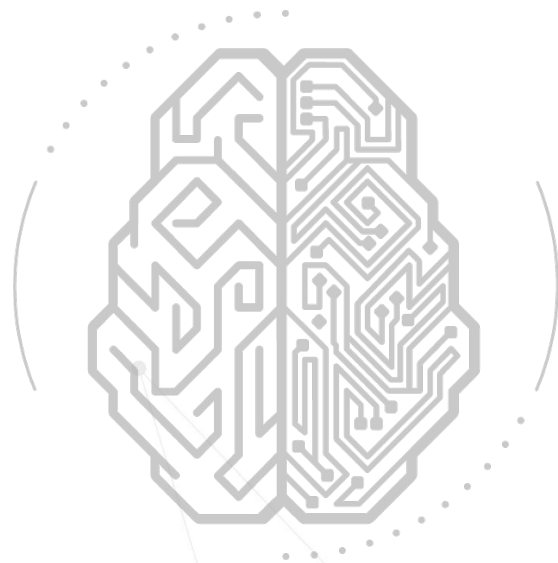
*CorsConfiguration:[]*

*// End of processable content for MyMacro and AWS::Serverless*



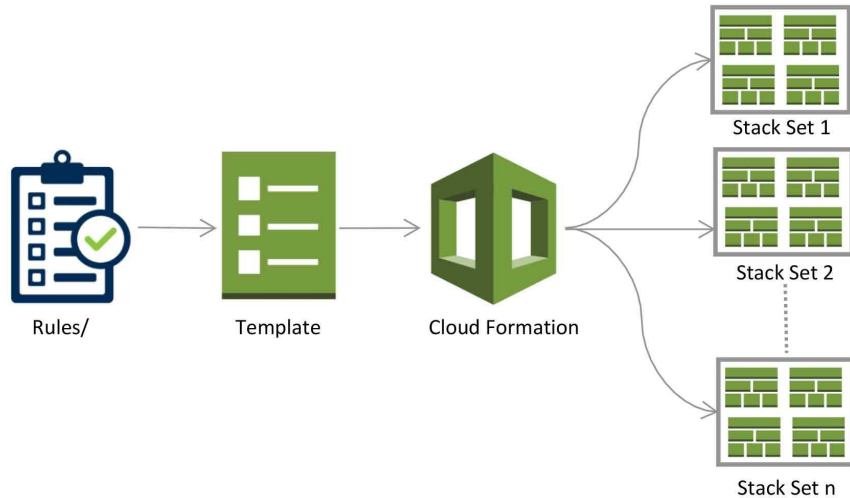
Reuse of template

# Stacks in AWS CloudFormation



# Stack

- A collection of AWS resources is called a Stack and it can be managed in a single unit
- CloudFormation Template defines a stack in which the resources can be created, deleted, or updated in a predictable way
- A stack can have all the resources (web server, database etc) that can be required to run a web application





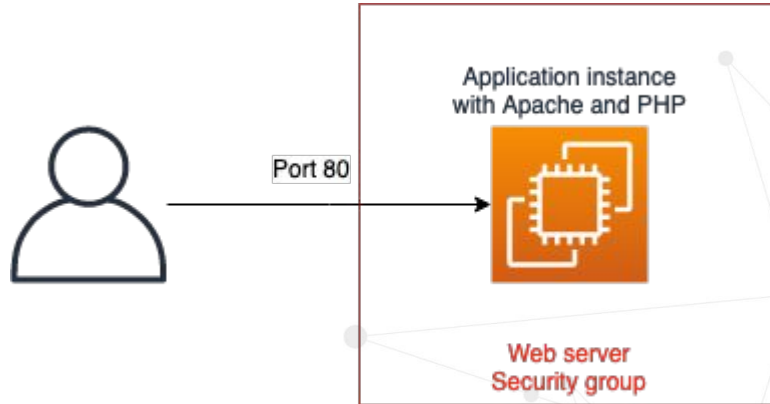
The background of the image is a light gray abstract network. It consists of numerous small circular nodes connected by thin, light gray lines. The nodes are distributed across the entire frame, with some appearing as larger, darker gray circles, possibly representing hubs or more significant nodes in the network. The overall effect is a complex, interconnected web of lines and dots.

**DEMO**

# Demo

- In this lab you will deploy an Apache Web server with a simple PHP application via UserData property.
  - First, you will bootstrap EC2 instance to install web server and content.
  - Then you will create an EC2 Security Group and allow access on port 80 to the instance.
  - Finally, you will view the content served by the web server.

The following diagram provides a high-level overview of the architecture you will implement.

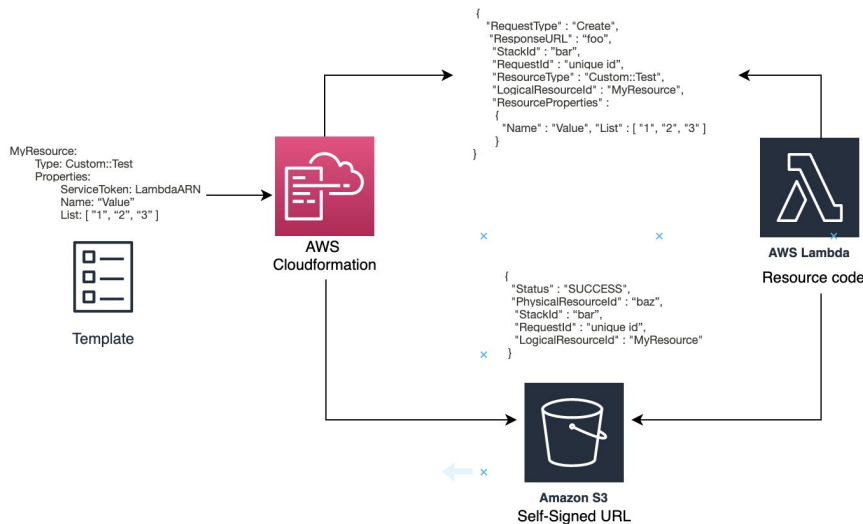


# Advanced Concepts



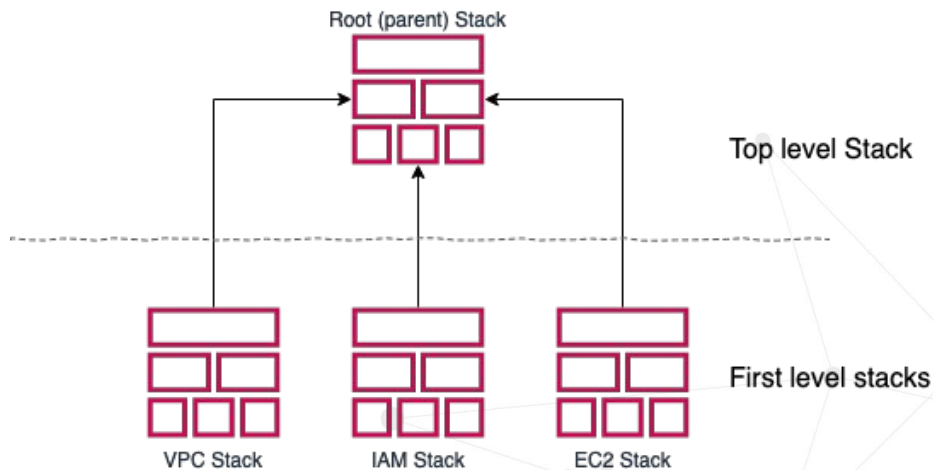
# Custom Resource

- A custom resource is a custom type powered by a Lambda function to execute a task that CloudFormation cannot do, for example, retrieving an AMI ID or the CIDR block for a VPC.
- Custom resources have a “request type” included with the request, allowing the custom resource to create, update and delete whatever it is doing.
- Outside to just normal lookup, there are more capabilities an AWS CloudFormation Custom Resource can provide



# Nested Stack

- Nested stacks are stacks created as part of other stacks. We create a nested stack within another stack by using the `AWS::CloudFormation::Stack` resource
- Nested stacks can themselves contain other nested stacks, resulting in a hierarchy of stacks, as in the diagram below. The root stack is the top-level stack to which all the nested stacks ultimately belong.



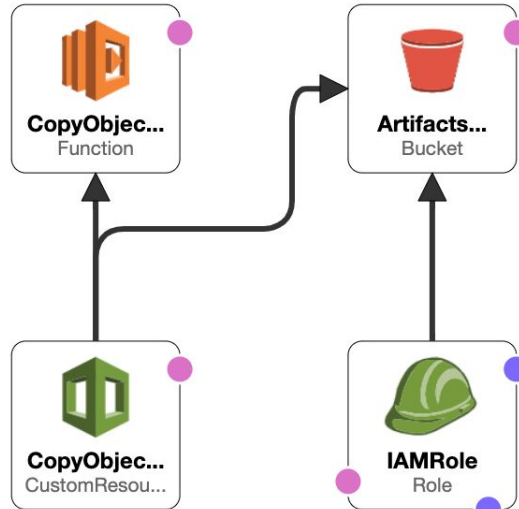
The background of the image is a light gray abstract network. It consists of numerous small circular nodes connected by thin, light gray lines. The nodes are distributed across the entire frame, with some appearing as larger, darker gray circles. The connections between the nodes form a complex, web-like pattern that fills the background.

**DEMO**

# Demo - Custom Resource

- In this demo, we will leverage Cloudformation custom resource to copy data from one S3 bucket to another S3 bucket, and post successful copy the cloudformation stack will be completed

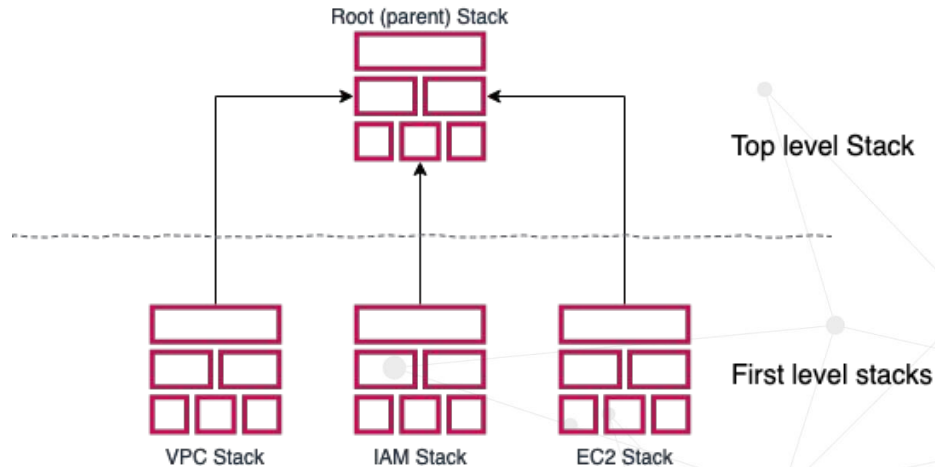
The following diagram provides a high-level overview of the architecture you will implement.



# Demo - Layered Stack

- In this lab you will deploy an Apache Web server with a simple PHP application via UserData property.
  - First, you will bootstrap EC2 instance to install web server and content.
  - Then you will create an EC2 Security Group and allow access on port 80 to the instance.
  - Finally, you will view the content served by the web server.

The following diagram provides a high-level overview of the architecture you will implement.





The background of the image features a complex, abstract network of thin, light gray lines connecting numerous small, dark gray circular nodes. These nodes are scattered across the entire frame, creating a web-like pattern that suggests a global or digital network. The lines vary in length and orientation, forming a dense yet delicate structure.

**THANK YOU**