Cloud Formation Stack Features :

Step 01: Pre-requisites

- Create default VPC if not present
- Create key pairs
    - Cfn-key-1
    - cfn-key-2
- Gather AMI ID

Step 02: Stack Features

- Create stack
    - Just create an instance without a keypair
- Update stack
    - Add keypair to the above stack
- Create change sets: (same like update but has an additional verification mechanism)
    - Change the key-1 to key-2
    - Change the instance type
- Rollback
    - Change the keypair to key-3 which is non existing one.

Resources : (Required)

- Resources are the key components of a CF stack.

  Syntax:

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

  Example:

  ```
  Resources:
    MyEC2Instance:
      Type: “AWS: : EC2 : : Instance”
      Properties:
          ImageID: “ami-1234456ff123”
  ```

!Ref : Intrinsic Function

- Ref returns the value of the specified parameter or resource, so basically it can be used in parameter or resource section of the CF.

- Resource Case: When we specify a resource logical name, it returns a value that we can typically use to refer to that resource.
- Parameter Case : When we specify a parameter logical name, it returns the value of that parameter.

  Syntax :
  - Long Form:
    - Ref: logicalName
  - Short Form:
    - !Ref logicalName

  Example:

```
MyEIP:
  Type: "AWS::EC2::EIP"
  Properties:
        InstanceId: !Ref MyEC2Instance
```

Parameters:

- It enables us to input custom values to our template each time when we create or update a stack.
- Maximum 60 parameters can have a single CFN template.
- Must be given a logical name (logical ID) which must be alphanumeric and unique among all logical names within the template.
- Each parameter must be assigned a parameter type that is supported by AWS CF.
- Each parameter must be assigned a value at runtime for AWS CF to successfully provision the stack. We can optionally specify a default value to unless another value is provided.
- Must be declared and referenced within the same template.
- We can reference parameters from the resources and outputs sections of the template.

  Syntax:

```
Parameters:
  ParameterLogicalId:
        Type: DataType
        ParameterProperty: Value
```

  Examples:

```
Parameters:
  InstanceTypeParameter:
        Type: String
        Default: t2.micro
        AllowedValues:
          - t2.micro
          - m1.small
          - m1.large
```

Description: Please enter the required Instance type.

Parameter Properties:

- AllowedPattern
- AllowedValues
- ConstraintDescription
- Default
- Description
- MaxLength
- MaxValue
- MinLength
- MinValue
- NoEcho

Parameter Types:

- String
- Number
- List
- CommaDelimitedList
- AWS Specific
    - AWS::EC2::Instance::Id
    - AWS::EC2::VPC::Id
    - List<AWS::EC2::Subnet::Id>
- SSM Parameter Type
    - AWS::SSM::Parameter::Name
    - AWS::SSM::Parameter::Value<String>
    - AWS::SSM::Parameter::Value<List<String>>

Mappings:

- Matches a key to a corresponding set of named values.
- For an example, if we wanted to set values based on a region, we can create a mapping that uses region name as a key and contains the values we want to specify for each region.
- We can use Fn::FindInMap intrinsic function to retrieve values in map

Syntax:

```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: value02
    Key03:
```

```
                Name: Value03
```

Examples:

```
Mappings:
  RegionMap:
        us-east-1:
          "HVM64": "ami-11111ddd111"
        eu-west-1:
          "HVM64": "ami-334323ffd222"
        ap-southeast-1:
          "HVM64": "ami-9767ghhh1122"
```

Intrinsic Function: FindInMap :

- FindInMap returns the value corresponding to keys in a **two-level map** that is declared in mappings section.
- **Parameters:**
    - Map Name
    - Top Level Key
    - Second Level Key
    - Return Value

Example:

```
Mappings:
  RegionMap:
        us-east-1:
          "HVM64": "ami-11111ddd111"
          "HVMG2": "ami-5735ggh445"
        eu-west-1:
          "HVM64": "ami-334323ffd222"
          "HVMG2": "ami-557556474t5"
        ap-southeast-1:
          "HVM64": "ami-9767ghhh1122"
          "HVMG2": "ami-96468vdfhsh222"
Resources:
  DevEC2Instance:
        Type: "AWS::EC2::Instance"
        Properties:
          ImageId: !FindInMap
                - RegionMap
                - !Ref 'AWS::Region'
                - HVM64
```

Pseudo Parameters:

- These parameters are predefined by AWS CF.
- Don't need to declare them in our template.
- We can use them the same way as we use parameters as an argument for **Ref** function.
- Usage:

```
Outputs:
  DevStacksRegion:
    Value: !Ref "AWS::Region"
```

Parameters:

- AWS::AccountId
- AWS::NotificationARNs
- AWS::NoValue
- AWS::Partition
- AWS::Region
- AWS::StackId
- AWS::StackName
- AWS::URLSuffix

Conditions:

- It contains statements that define the circumstances under which entities are created or configured.
- **Exmpl-1**: we can create a condition and then associate it with a resource or output so that AWS CF only creates the resource or output if the condition is true.
- **Exmpl-**2: we can associate the condition with a property so that AWS CF only sets the property to a specific value if the condition is true, if the condition is false, AWS CF sets the property to a different value that we specify.
- We will use conditions, when we want to reuse the template in different contexts like dev and prod environments.
- Within each condition we can reference the other condition.
- We can associate these conditions in three places.
  - Resources
  - Resource Properties
  - Outputs
- A stack creation or stack update, AWS CF evaluates all conditions in the template. During stack update, resources that are now associated with a false condition are deleted.

Syntax:

```
Conditions:
  Logical ID:
    Intrinsic function
```

Example:

```
Conditions:
```

```
CreateEIPForProd:
      Fn::Equals:
        - !Ref EnvironmentName
        - Prod
```

Conditions - Intrinsic Functions :

- Below are the list of conditions which can be used in AWS CF.
    - Fn::And
    - Fn::Equals
    - Fn::If
    - Fn::Not
    - Fn:Or

Outputs: (Optional)

- It declare output values that we can use as below
    - Import into another stack (to create cross-stack references)
    - When using Nested Stacks we can see how outputs of a nested stack are used in Root Stack.
    - We can view outputs on the CF Console.
    - We can declare maximum of 60 outputs in a CF template

Syntax:

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

Example:

```
Outputs:
  InstanceId:
        Description: Instance ID
        Value: !Ref MyVMInstance
        Export:
          Name: !Sub "${AWS::StackName}-InstanceId"
MyInstAvailabilityZone:
        Description: Instance Availability Zone
        Value: !GetAtt MyVMInstance.AvailabilityZone
        Export:
          Name: !Sub "${AWS::StackName}-InstanceAz"
```

Export (Optional) :

- Contains resource output used for cross stack reference.
- For each AWS account Export name must be unique within the region. As it should be unique we can use the export name as **“AWS::StackName”-ExportName**
- We can't create cross stack references across region.
- We can use the intrinsic function Fn::ImportValue to import value that have been exported within the same region.
- For outputs the value of the name property of na export cant use Ref or GetAtt functions that depend on a resource.
- We can't delete a stack if another stack references one of its output.
- We can't modify or remove an output value that is referenced by another stack.
- We can use outputs in combination with Conditions.

Metadata:

- Metadata provides details about the cfn template.

  Syntax:
  ```
          Instances:
            Description: “Info about the Instances”
          Databases:
            Description: “Info about the Databases”
  ```

  Example:
  ```
          AWSTemplateFormatVersion: 2010-09-09
          Metadata:
            Instances:
                  Description: My VM Instance
  ```

- We have three types of metadata keys which are listed below
  - AWS::CloudFormation::Designer
    - Auto generated during resources drag and drop to canvas
      - Designer, visually depicts how our resources are laid out.
      - Automatically add information when we use it to create, view and update templates. Its a system generated metadata.
      - Defines the info about the resources such as their Size and Relative position in template metadata. All layout info is stored in designer metadata.
      - We drag and drop the resources.
      - When we create template,It enforces some basic relationships between resources to help us create valid template.
      - Example: we can not directly add EC2 Instances in VPC, we must add a subnet in a VPC.
      - We can validate templates directly in designer.

        - We can validate our manually written template using Designer validate template.
        - Conversion of JSON to YAML or vice versa is possible in designer.
    - AWS::CloudFormation::Interface
        - Used for parameter grouping
            - When we create or update stacks in the console, the console lists input parameters in alphabetical order by their logical IDs.
            - By Using this Key, we can define our own parameter grouping and ordering so that users can efficiently specify parameter values.
            - We can also define labels for parameters.
            - A label is friendly name or description that the console displays instead of a parameter's logical ID which helps users understand the values to specify for each parameter.

              Syntax:

```
Metadata:
  AWS::CloudFormation::Interface
        ParameterGroups:
          - ParameterGroup
        ParameterLabels:
          - ParameterLabel
```

              Example:

```
Metadata:
  AWS::CloudFormation::Interface
        ParameterGroups:
          - Label:
                default: "EC2 Instance configuration"
            Parameters:
                - InstanceType
                - KeyName
          - Label:
                Default: "Environment Configuration"
            Parameters:
                - EnvironmentName
        ParameterLabels:
          EnvironmentName:
                Defaults: "which env we are planning to"
```

    - AWS::CloudFormation::Init
        - Used for application installation and configurations on AWS Compute EC2 instances

EC2UserData:

- We can use User Data in CF template for EC2.

- We need to use intrinsic function Fn::Base64 with UserData in CFN template. This function returns the BAse64 representation of input string. It passes encoded data to EC2 Instance.
- YAML Pipe (|) : Any intended text that follows should be interpreted as a multiline scalar value which means value should be interpreted literally in such a way that preserves newlines.
- UserData Cons:
  - By default UserData scripts and cloud-init directives run only during the boot cycle when we first launch an instance.
  - We can update our configuration to ensure that our userdata scripts and cloud init directives run every time we restart our instance. (Reboot of server required)

Sample:

```
UserData:
  Fn::Base64:
        #!/bin/bash
        Sudo yum update
        Sudo yum -y erase java-1.7.0-openjdk.x86_64
        Sudo yum -y install java-1.8.0-openjdk.x86_64
        Sudo yum -y install java-1.8.0-openjdk-devel
        Sudo yum -y install tomcat8
        Service tomcat8 start
        Mkdir /usr/share/tomcat8/webapps/ROOT
        Touch /usr/share/tomcat8/webapps/ROOT/index.html
        Echo "Hello World" > /usr/share/tomcat8/webapp/ROOT/Index.html
```