AWS CloudFormation Template

📌 Module 1: Introduction to AWS CloudFormation

1.1 What is AWS CloudFormation?

- AWS CloudFormation is an **Infrastructure as Code (IaC)** tool that allows you to define AWS resources using **YAML or JSON**.
- Automates deployment of AWS services such as **EC2, S3, IAM, VPC, RDS**, etc.
- Ensures **consistency, repeatability, and automation** in infrastructure management.

1.2 Why Use CloudFormation?

✅ Automates infrastructure deployment
✅ Reduces manual errors
✅ Enables version control for infrastructure
✅ Supports rollback on failures
✅ Works with AWS native and third-party services

📌 Module 2: CloudFormation Components & Concepts

2.1 CloudFormation Building Blocks

1. **Template** – YAML/JSON file defining resources.
2. **Stack** – A group of AWS resources created using a CloudFormation template.
3. **Change Set** – A preview of changes before applying updates.
4. **Drift Detection** – Detects manual changes outside CloudFormation.

2.2 CloudFormation Template Structure

A template consists of the following sections:

```yaml
AWSTemplateFormatVersion: '2010-09-09'  # CloudFormation version
Description: "Sample CloudFormation Template"
Resources:   # Defines AWS resources
  MyBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: my-cloudformation-bucket
```

📌 Module 3: Writing Your First CloudFormation Template

3.1 Defining Resources

- Every AWS service (EC2, S3, RDS, etc.) is defined under the **Resources** section.
- Example: Creating an EC2 instance

```
Resources:
 MyEC2Instance:
  Type: AWS::EC2::Instance
  Properties:
   ImageId: ami-0abcdef1234567890
   InstanceType: t2.micro
```

3.2 Running Your First CloudFormation Stack

- **Using AWS CLI**

```
aws cloudformation create-stack --stack-name MyFirstStack --template-body
file://template.yaml
```

- **Using AWS Console**
  - Navigate to **CloudFormation** > **Create Stack**
  - Upload your template
  - Click **Create Stack**

📌 Module 4: Parameters, Mappings, and Conditions

4.1 Parameters

- Allow user input at runtime.
- Example: Parameterizing EC2 instance type

```
Parameters:
 InstanceType:
  Type: String
  Default: t2.micro
  AllowedValues: [t2.micro, t2.small, t2.medium]
Resources:
 MyEC2Instance:
  Type: AWS::EC2::Instance
  Properties:
   InstanceType: !Ref InstanceType
```

4.2 Mappings

- Store static configuration values.
- Example: Selecting AMI ID based on region

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-123456
    us-west-1:
      AMI: ami-789012
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AMI]
```

4.3 Conditions

- Define conditional resource creation.
- Example: Create an S3 bucket only if the environment is **Production**.

```
Parameters:
  Environment:
    Type: String
    AllowedValues: [Dev, Prod]
Conditions:
  IsProd: !Equals [!Ref Environment, "Prod"]
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Condition: IsProd
```

📌 Module 5: Outputs, Intrinsic Functions, and Dependencies

5.1 Outputs

- Export values from a stack.

```
Outputs:
 BucketName:
  Value: !Ref MyS3Bucket
  Description: "The created S3 bucket name"
```

5.2 Intrinsic Functions

- Built-in functions to reference and manipulate resources.
- **Common Functions**:
    - `!Ref` → Reference resources/parameters
    - `!Sub` → String substitution
    - `!Join` → Concatenate strings
    - `!GetAtt` → Get an attribute of a resource

Example:

```
Resources:
 MyBucket:
  Type: AWS::S3::Bucket
Outputs:
 BucketArn:
  Value: !GetAtt MyBucket.Arn
```

5.3 Handling Dependencies

- CloudFormation automatically determines dependencies, but you can explicitly define them using **DependsOn**.

```
Resources:
 MyBucket:
  Type: AWS::S3::Bucket
 MyLambda:
  Type: AWS::Lambda::Function
  DependsOn: MyBucket
```

📌 Module 6: Advanced CloudFormation Features

6.1 Nested Stacks

- Used for **modular templates** (reusable CloudFormation templates).
- Example: Importing a child stack inside a parent stack.

```
Resources:
  MyNestedStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: "https://s3.amazonaws.com/mybucket/mystack.yaml"
```

6.2 Stack Policies

- Restrict stack modifications.

```
{
 "Statement": [
  {
   "Effect": "Deny",
   "Action": "Update:Delete",
   "Principal": "*",
   "Resource": "*"
  }
 ]
}
```

📌 Module 7: CloudFormation Best Practices

✅ **Use Parameters & Mappings** for flexible, region-independent templates.
✅ **Enable Stack Rollback** to recover from failed deployments.
✅ **Use Nested Stacks** for reusable, modular architecture.
✅ **Leverage AWS IAM** to restrict CloudFormation permissions.
✅ **Validate templates** before deploying using:

```
aws cloudformation validate-template --template-body file://template.yaml
```

📌 Module 8: Real-World CloudFormation Project

Project: Deploy a Scalable Web Application

1. **VPC Setup** – Define subnets, security groups, and route tables.
2. **EC2 Instances** – Launch web servers in an Auto Scaling Group.
3. **Load Balancer** – Distribute traffic across multiple EC2 instances.
4. **RDS Database** – Deploy a PostgreSQL/MySQL database for backend storage.
5. **S3 Bucket** – Store static content.
6. **CloudWatch** – Set up monitoring and logging.

📌 Module 9: CloudFormation vs. Other IaC Tools

Feature	CloudFormation	Terraform	Ansible
Type	AWS-native IaC	Multi-cloud IaC	Configuration Management
Language	YAML / JSON	HCL	YAML
State Management	Managed by AWS	Requires Terraform state files	Not stateful
Multi-Cloud Support	AWS Only	AWS, Azure, GCP, etc.	Works on servers, not cloud-native

---