YAML Basics

- YAML is a data serialisation language designed to be directly writable and readable by humans.
- It’s a strict superset of JSON, with the addition of **syntactically significant newlines and indentation**, like Python.
- YAML is case sensitive, the files should have .yaml/.yml as the extension
- YAML does not allow the use of tabs while creating YAML files; spaces are allowed instead.
- **Conventional Block Format : Uses** hyphen+space to begin a new item in a specified list
- **Inline Format :** It is delimited with comma and space
- **Folded Text :** Folded text converts newlines to spaces and removes the leading whitespace
- **Extension:** YAML file should end with extensions like **.yaml** or **.yml**

YAML Basics

- The structure which follows all the basic conventions of YAML is shown below -

```
AWS: [EC2, S3, VPC, IAM]


AWS:

  - EC2

  - S3

  - VPC

  - IAM
```

Synopsis of YAML Basic Elements

- Comments in YAML begins with the (**#**) character.
- Indentation of whitespace is used to denote structure.
- Tabs are not included as indentation for YAML files.
- List members are denoted by a leading hyphen (**-**).
- List members are enclosed in square brackets and separated by “**,**”
- Multiple documents with single streams are separated with 3 hyphens (---).
- YAML always requires colons and commas used as list separators followed by space with scalar(key value) values.

Infrastructure as Code - IAC

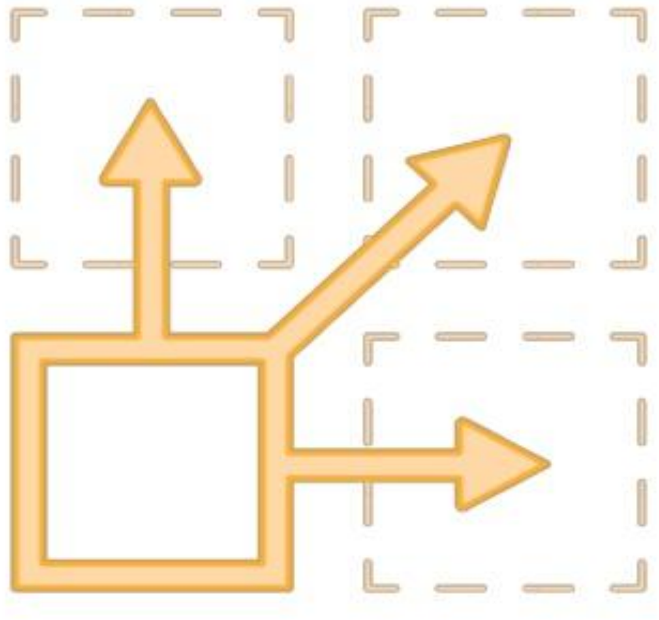
- As of now, we have deployed many Resources in AWS, but **manually (Using AWS Console)**.
- It becomes difficult to reproduce the same set of resources in:
  - In Different Region
  - In Different AWS Account
  - Multiple environments (dev/qa/prod) in multiple regions in multiple AWS account.

- Wouldn't it be great, if all our infrastructure was… code?
  That code would be deployed and **create / update / delete** our infrastructure

What is CloudFormation?

- Helps you model and set up your Amazon Web Services resources, so that you can spend less time managing those resources and more time focusing on your applications that run in AWS.
- **<u>What we do :</u>**
  - Just Create a template(file) that describes all the AWS resources that you want.
- **<u>What CloudFormation does :</u>**
  - AWS CloudFormation takes care of provisioning and configuring those resources for you.

Templated resource provisioning

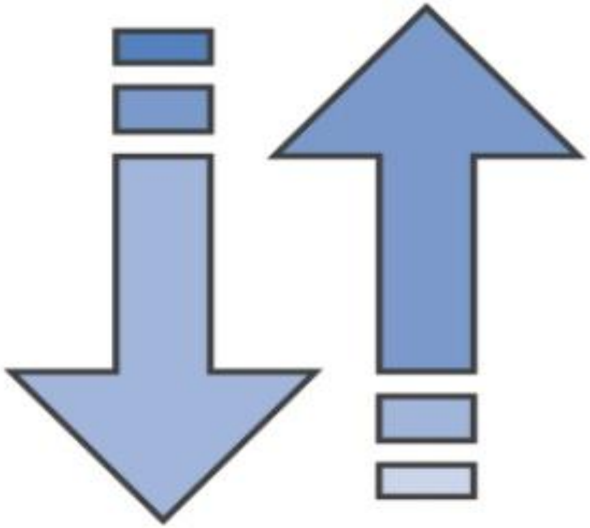


- Create templates to describe the AWS resources used to run your application
- Provision identical copies of a stack

Infrastructure as code

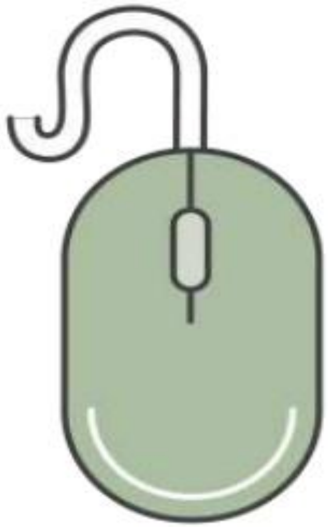
- Templates can be stored in a source control system
- Track all changes made to your infrastructure stack
- Modify and update resources in a controlled and predictable way

Declarative and Flexible



- Just choose the resources and configurations you need
- Customize your template through parameters

Easy to use



- Access through console, CLI, or SDKs
- Start with one of the many sample templates
- Integrate with your development and management tools

How CloudFormation Helps…

- **<u>Simplify Infrastructure Management</u>**
  - When you use that template to create an AWS CloudFormation stack, AWS CloudFormation provisions the EC2, S3 Buckets, RDS etc
  - After the stack has been successfully created, your AWS resources are up and running
  - You can delete the stack just as easily, which deletes all the resources in the stack.

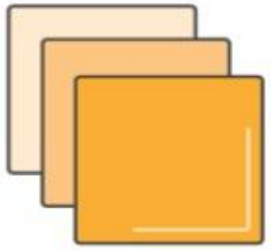
How CloudFormation Helps…

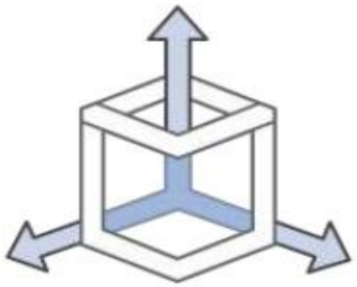
- **Quickly Replicate Your Infrastructure**
  - When you use AWS CloudFormation, you can reuse your template to set up your resources consistently and repeatedly in **different regions**.
- **Easily Control and Track Changes to Your Infrastructure**
  - You can use a version control system with your templates so that you know exactly what changes were made, who made them, and when. If at any point you need to reverse changes to your infrastructure, you can use a previous version of your template.

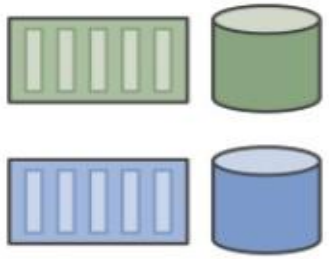
CloudFormation

- Create templates of the infrastructure & applications you want to run on AWS.
- Have CloudFormation automatically provision the required AWS resources and their relationships from the templates.
- Easily version, replicate, or update the infrastructure and applications using the templates.
- Integrates with other development, CI/CD, and management tools

Common Use Cases


Stack replication


Infrastructure scale out

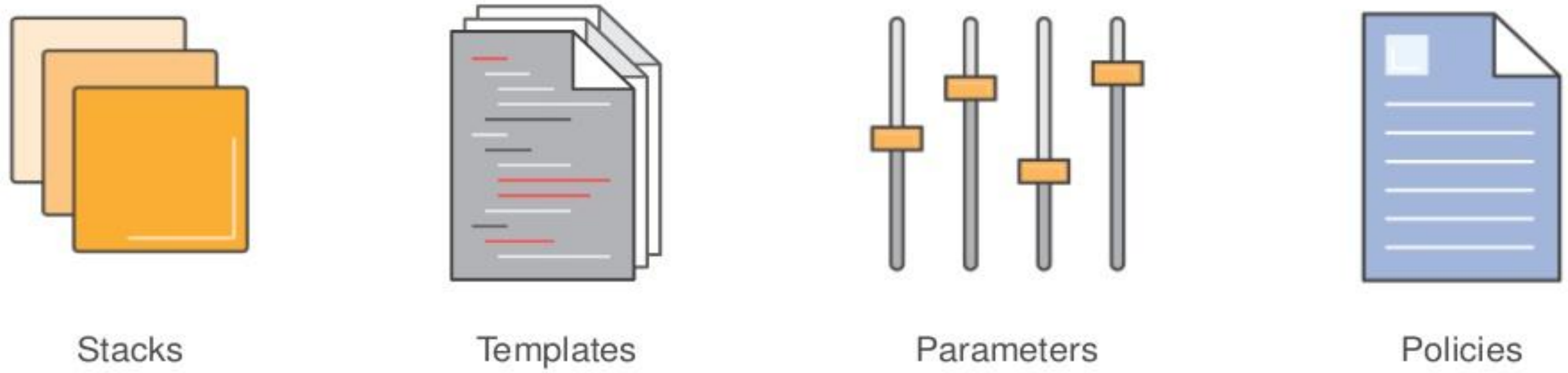

Blue/green deployments

Infrastructure as code

Pricing

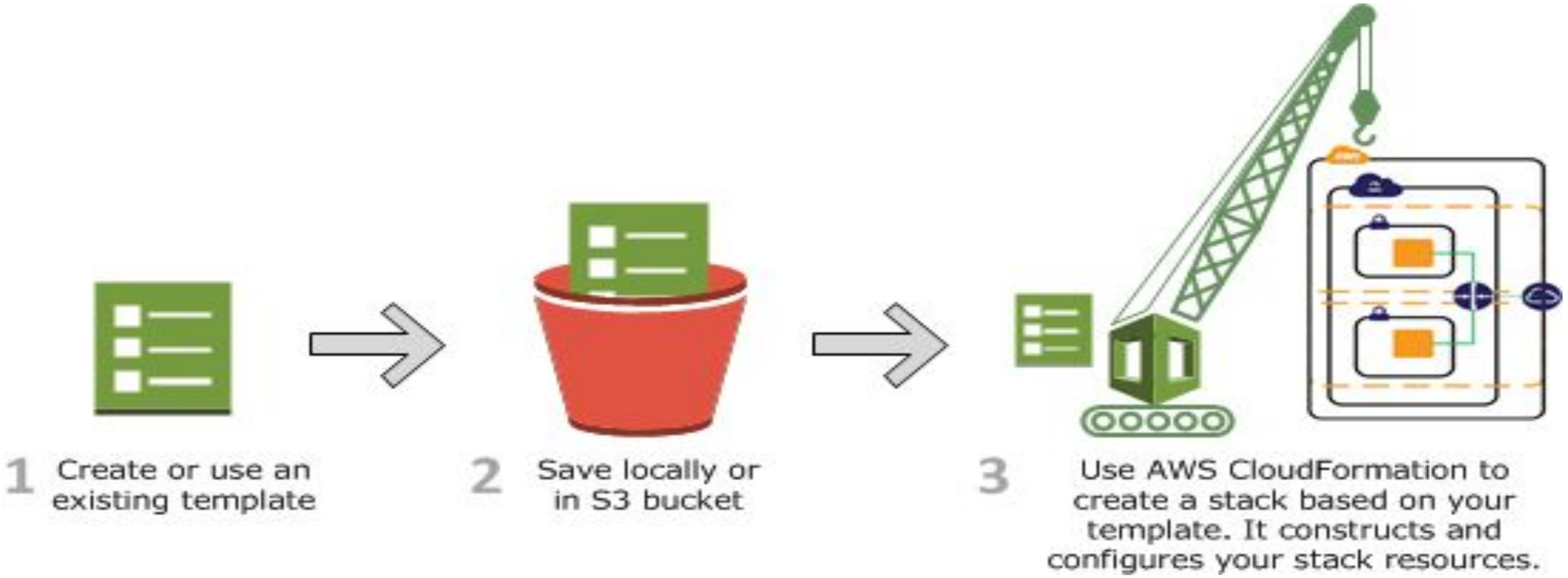
- <u>**No additional charge for CloudFormation.**</u>
- Customers pay only for the AWS resources (e.g EC2 instances, EBS Volumes, RDS instances etc) that are created using CloudFormation.

CloudFormation terminology



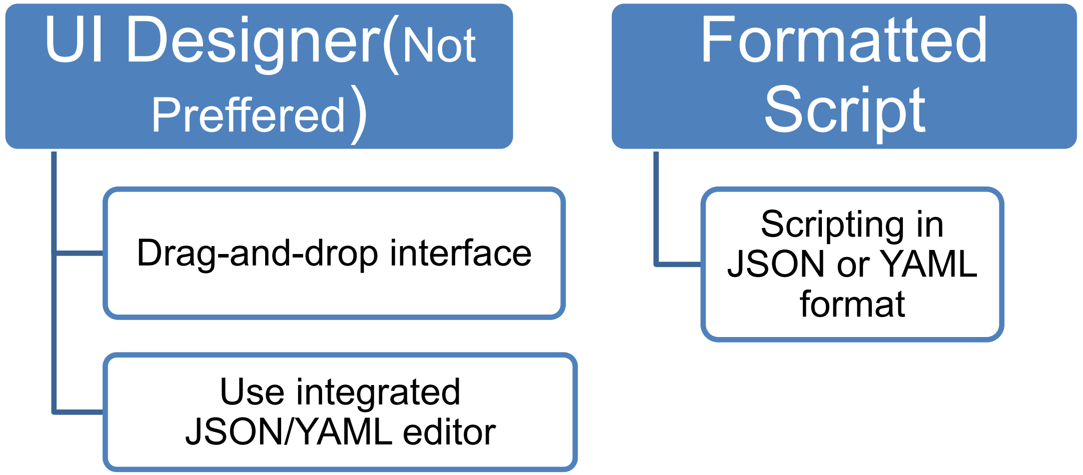
AWS CloudFormation Basics

- **Template:** JSON or YAML formatted text file which is a blueprint of your AWS resources.
- **Stack:** In CloudFormation, you manage resources as a single unit called a Stack. All the resources in a stack are defined by the stack's Template.


1 Create or use an existing template
2 Save locally or in S3 bucket
3 Use AWS CloudFormation to create a stack based on your template. It constructs and configures your stack resources.

CF Template :

- A template is a **JSON or YAML** formatted text file that describes what resources are contained in the Stack.
- It contains information about each resource, its configuration and how it may be connected or dependent on other resources.
- A template can be developed using two methods:


UI Designer(Not Preffered)
Drag-and-drop interface
Use integrated JSON/YAML editor
Formatted Script
Scripting in JSON or YAML format

CF Template Sections:

- Templates include several major sections.
- The Resources section is the only required section.
- Some sections in a template can be in any order.
- However, as you build your template, it can be helpful to use the logical order shown in the following list because values in one section might refer to values from a previous section.

CF Template Sections:

- **AWSTemplateFormatVersion (**optional**):** Section identifies the capabilities of the template. The latest template format version is 2010–09–09 and is currently the only valid value.
- **Description(**optional**):** Describe about purpose and use case of this particular template
- **Resources (Required):** Resources is the minimum required section for a template, section contains reference information for all AWS resources that are supported by AWS CloudFormation.
- **Metadata(**optional**):** Describe the about each resources used in the template, Some AWS CloudFormation features retrieve settings or configuration information that you define from the Metadata section.
- **Parameters(**optional**):** Values to pass to your template at runtime (when you create or update a stack). You can refer to parameters from the **Resources** and **Outputs** sections of the template..

CF Template Sections:

- **Mappings(optional):** A mapping of keys and associated values that you can use to specify conditional parameter values, similar to a lookup table. You can match a key to a corresponding value by using the **Fn::FindInMap** intrinsic function in the **Resources** and **Outputs** sections
- **Conditions(optional):** this section includes statements that define when a resource is created or when a property is defined. For example, you can compare whether a value is equal to another value. Based on the result of that condition, you can conditionally create resources.
- **Outputs (optional):** Describes the values that are returned whenever you view your stack's properties. For example, you can declare an output for an S3 bucket name and then call the aws cloudformation describe-stacks AWS CLI command to view the name.

CloudFormation Resources

- Resources are the core of your CloudFormation template (MANDATORY)
- They represent the different AWS Components that will be created and configured
- Resources are declared and can reference each other
- CloudFormation has over 224 types of resources (!)
- AWS Resource types identifiers are of the form:
  AWS::aws-service-name::data-type-name

- All CloudFormation Resources can be found here

CloudFormation Parameters

- CF parameters types
- **AWS-specific parameter types**

- All CloudFormation Resources can be found here

Resource Section (Required)

```
"Resources" : {
    "Logical ID" : {
        "Type" : "Resource type",
        "Properties" : {
            Set of properties
        }
    }
}
```

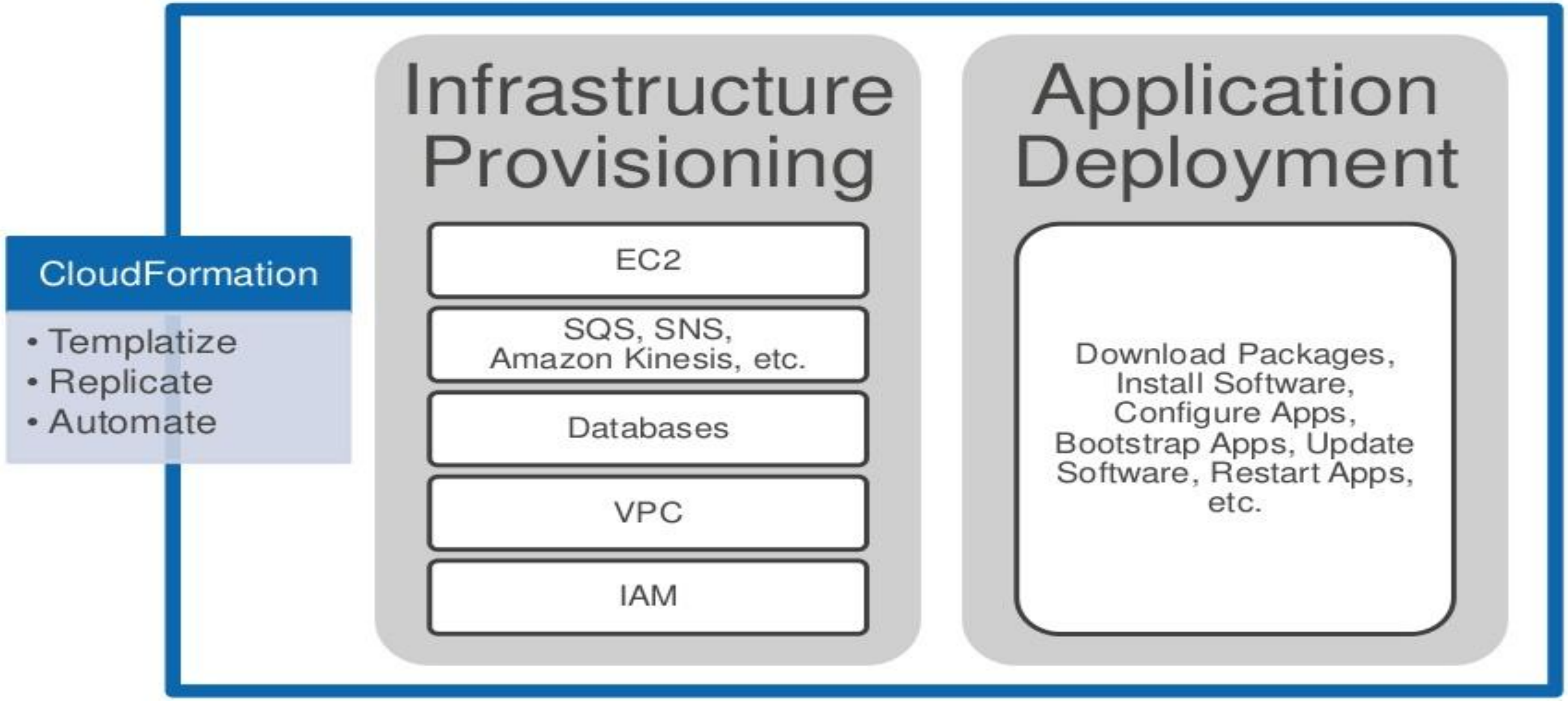
```
Resources:
  Logical ID:
    Type: Resource type
    Properties:
      Set of properties
```

- **Logical ID:** Logical name of the resource just to use inside the template to refer other part of the template.
- **Type:** The resource type identifies the type of resource that you are declaring. For example, **AWS::EC2::Instance** declares an EC2 instance.

Steps to be followed…

- Create or Use an existing template.
- Upload a template to CloudFormation.
- Specify parameter values.
- Set up tags or notification options.
- Review and create.

Application Deployment as Code



CloudFormation
• Templatize
• Replicate
• Automate
Infrastructure Provisioning
EC2
SQS, SNS, Amazon Kinesis, etc.
Databases
VPC
IAM
Application Deployment
Download Packages, Install Software, Configure Apps, Bootstrap Apps, Update Software, Restart Apps, etc.

Intrinsic Functions

- Ref
- Fn::Join
- Fn::FindInMap
- Fn::GetAtt
- Fn::GetAZs
- Fn::Select
- Fn::Sub
- Condition Functions
- Fn::Cidr
- Fn::ImportValue
- Fn::Split
- Fn::Transform

Supports wide range of AWS Services

- ✓ Auto Scaling
- ✓ Amazon CloudFront
- ✓ AWS CloudTrail
- ✓ Amazon CloudWatch
- ✓ Amazon DynamoDB
- ✓ Amazon EC2
- ✓ AWS Elastic Beanstalk
- ✓ Amazon ElastiCache
- ✓ Elastic Load Balancing
- ✓ AWS Identity and Access Management (IAM)
- ✓ Amazon Kinesis
- ✓ AWS OpsWorks
- ✓ Amazon RDS
- ✓ Amazon Redshift
- ✓ Amazon Route 53
- ✓ Amazon S3
- ✓ Amazon SNS
- ✓ Amazon SQS
- ✓ Amazon VPC

Intrinsic Functions – YAML Syntax

- Resource Reference

- !Ref logicalName **or** Ref: logicalName

- Parameter Reference

- !Ref parameterName **or** Ref: parameterName

- Fn::Join

!Join [delimiter, [comma-delimited list of values]]

- Fn::FindInMap

!FindInMap [MapName, TopLevelKey, SecondLevelKey]

- Fn::GetAtt

Fn::GetAtt: [logicalNameOfResource, attributeName]

or

!GetAtt logicalNameOfResource.attributeName

Intrinsic Functions – YAML Syntax

- **Fn::GetAZs**

!GetAZs region

- Here **region** value can be **us-east-1** or “**AWS::Region**”, empty value is same as “**AWS::Region**”

- **Fn::Select**

!Select [index, listOfObjects]

- **Fn::GetAtt**

Fn::GetAtt: [logicalNameOfResource, attributeName]   OR
!GetAtt logicalNameOfResource.attributeName

- **Fn::Sub**

!Sub String
!Sub 'arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:vpc/${vpc}‘   <- Example

Conditions – YAML Syntax

- **Fn::If**

!If [condition_name, value_if_true, value_if_false]

- **Fn::Equals**

!Equals [!Ref EnvironmentType, prod]

- **Fn::Not**

!Not [condition]

CF - Pseudo Paramters

- **AWS::Region**
  - Returns a string representing the Region in which the encompassing resource is being created, such as us-west-2.
- **AWS::AccountId**
  - Returns the AWS account ID of the account in which the stack is being created, such as 123456789012.
- **AWS::StackName**
  - Returns the name of the stack as specified, such as **teststack**

CloudFormation - Drift Detection

- Drift detection enables you to detect whether a stack’s actual configuration differs, or has drifted, from its expected configuration.
- Use CloudFormation to detect drift on an entire stack, or on individual resources within the stack.
- A resource is considered to have drifted if any if its **actual property values** differ from the **expected property values** (Property values written in the template file).
- A stack is considered to have drifted if one or more of its resources have drifted.

CF - Cross Stack Reference

- CloudFormation allows us to reference resources from one CloudFormation stack and use those resources on another stack.
- This is called **cross-stack reference**.
- It allows for a layering of stacks, which is useful for separating your resources based on your services.
- Instead of putting all resources on one stack, you can create resources from one stack and reference those resources on other CloudFormation stacks.
- This also allows you to re-use the same CloudFormation stacks so that you can build faster if you need a new environment with minimal changes.
- <u>Use Cases:</u>

❑ **Network stack** – contains VPC, public and private subnets, and security groups.

❑ **Web server stack** – contains webserver and referencing the public subnets and security groups from the network stack

❑ **Database stack** – contains your database server and referencing the private subnets and security groups from the network stack.

CF - Cross Stack Reference

Note:

For each AWS account, Export names must be unique within a region.

- You can't create cross-stack references across regions.
- You can use the intrinsic function **Fn::ImportValue** to import only values that have been exported within the same region.
- For outputs, the value of the Name property of an Export can't use Ref or GetAtt functions that depend on a resource.
- You can't delete a stack if another stack references one of its outputs.
- You can't modify or remove an output value that is referenced by another stack.

CloudFormation Best Practices

- **Planning and organizing**

  Organize Your Stacks By Lifecycle and Ownership

  Use Cross-Stack References to Export Shared Resources

  Reuse Templates to Replicate Stacks in Multiple Environments

  Verify Quotas for All Resource Types

- **Creating Templates**

  Do Not Embed Credentials in Your Templates

  Use AWS-Specific Parameter Types

  Use Parameter Constraints

  Validate Templates Before Using Them

CloudFormation Best Practices

- **Managing stacks**

  Manage All Stack Resources with AWS CloudFormation

  Create Change Sets Before Updating Your Stacks

  Use Stack Policies

  Use Cross-Stack References to Export Shared Resources

  Reuse Templates to Replicate Stacks in Multiple Environments

  Verify Quotas for All Resource Types

CloudFormation Tips

- Turn on Termination Protection on all of your stacks to avoid costly accidents!
- Avoid hardcoding resource parameters that can potentially change.
- Use stack parameters as much as you can, and resort to default parameter values.
- Always keep CF template generalized that can work in any region or any other AWS account.