



AWS Foundation

App Services



Agenda

1

**Introduction to
CloudFormation**

2

**AWS CloudFormation
Concepts**

3

**How does AWS
CloudFormation work?**

4

Infrastructure as Code

5

**CloudFormation Templates
and Template Anatomy**

6

**CloudFormation Designer
and Stacks**

7

**Functions and Pseudo
Parameters**

8

**Simple Notification
Service**

9

Simple Email Service

10

**Simple Queue
Service**

Introduction to CloudFormation

Why CloudFormation?

Managing all the resources of a complex application on AWS becomes a problem when there are a lot resources to be maintained



Introduction to CloudFormation

Why CloudFormation?

CloudFormation allows us to model our entire infrastructure in a file



Why CloudFormation?

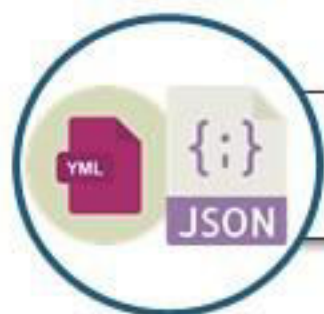
We can create or modify an existing AWS CloudFormation template. A template describes all of our resources and their properties

To make changes or modify, we can simply track the differences in our templates to check changes in our infrastructure, similar to the way developers control revisions to source code

We can also reuse our template to set up our resources consistently and repeatedly. We can just describe our resources once and then provision the same resources over and over to multiple regions

Introduction to CloudFormation

Top Features of CloudFormation



Authoring with JSON/YAML



Authoring with familiar programming languages



Safety controls



Preview changes to our environment

CloudFormation Concepts

CloudFormation Concepts

Template

Stacks

Change Sets

A CloudFormation template is a JSON or a YAML formatted file. We can save these files using the extensions: **.json**, **.yaml**, **.txt**, and **.template**



We can also specify multiple resources in a single template and configure these resources to work together

CloudFormation Concepts

Template

Stacks

Change Sets

We can manage related resources as a single unit called a stack. We can also delete the entire architecture by deleting the stack



We can work with stacks by using the AWS CloudFormation console, API, or AWS CLI

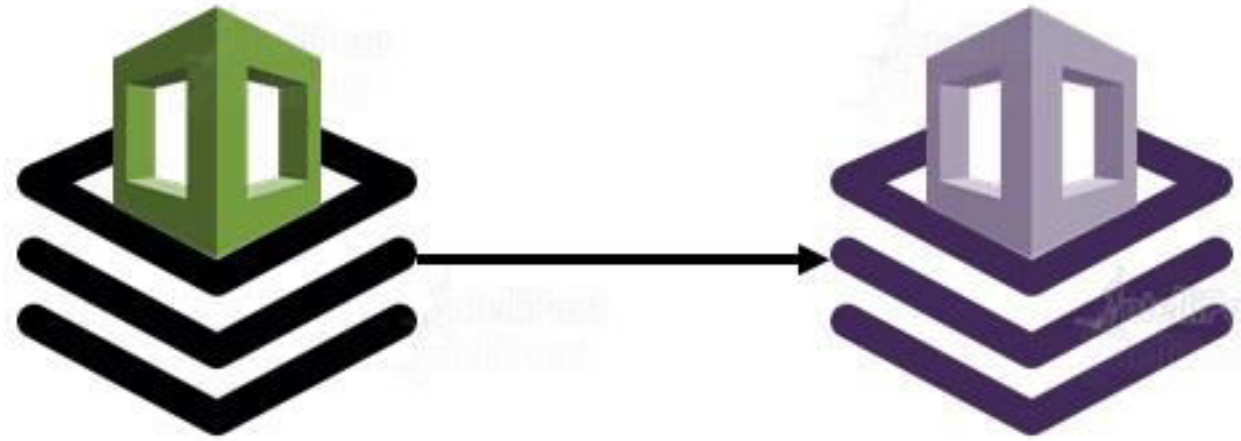
CloudFormation Concepts

Template

Stacks

Change Sets

To change the running resources, we need to update the stack. We can generate a change set, which is a summary of our proposed changes

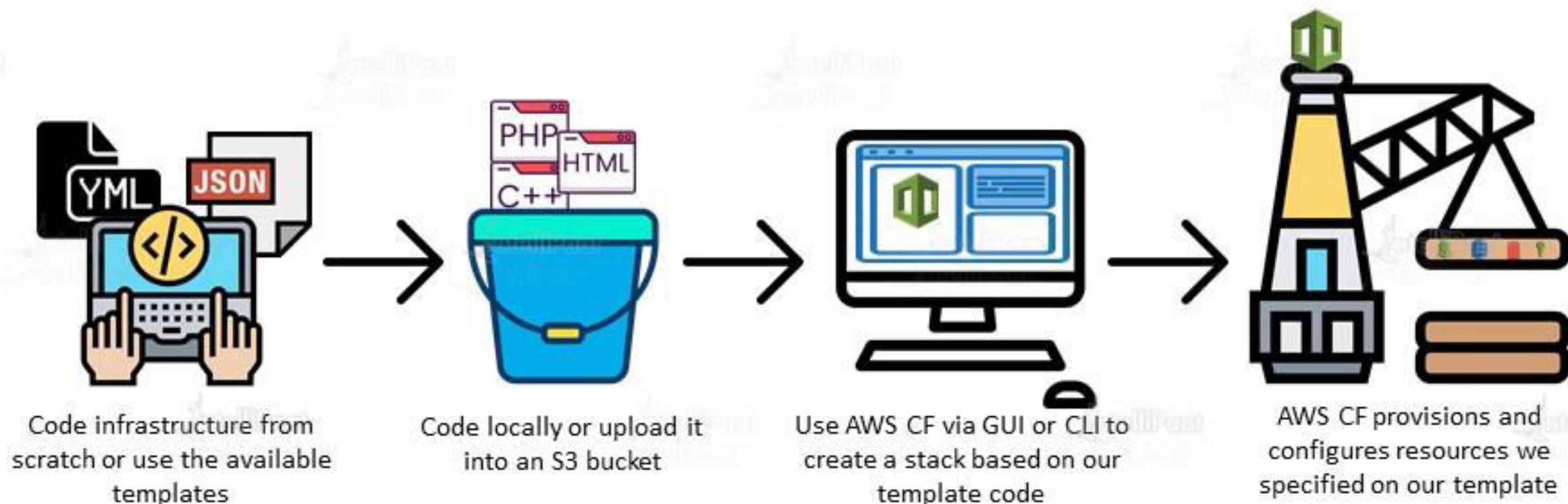


If we generate a change set, we can see how our changes will cause our database to be replaced, and we will be able to plan accordingly

How does AWS CloudFormation work?

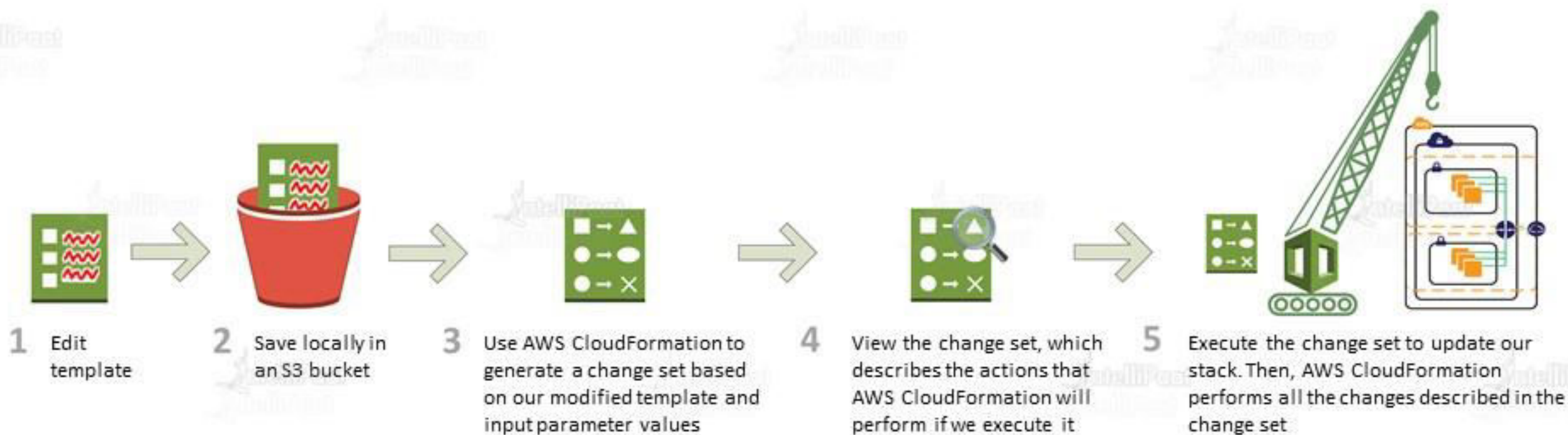
How does AWS CloudFormation work?

How does AWS CF work?



How does AWS CloudFormation work?

Updating the Stack with Change Sets



Infrastructure as Code in CloudFormation

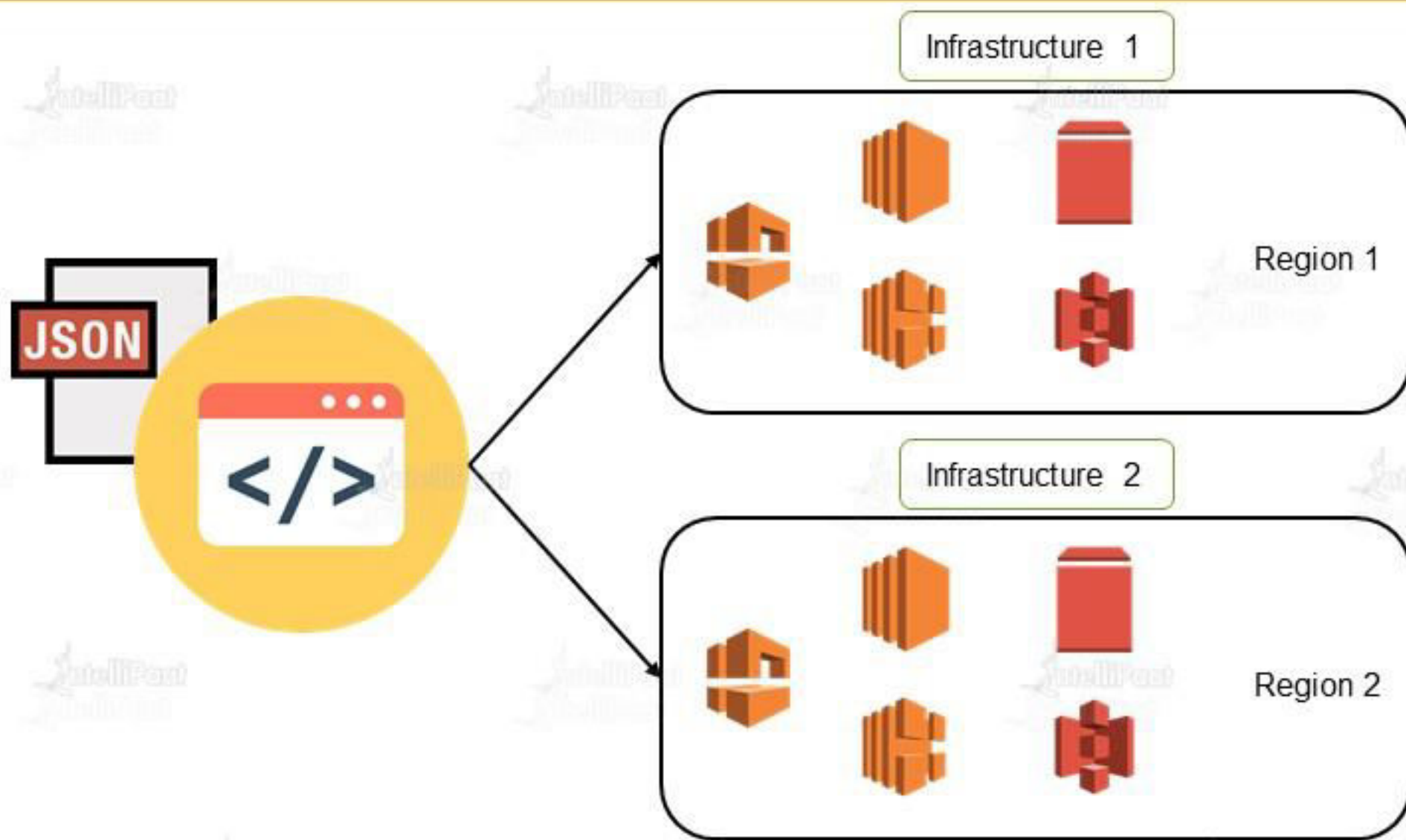
Infrastructure as Code in CloudFormation

Instead of creating multiple instances, databases, and other resources manually, we can create a single piece of code, which we can use to create multiple infrastructures



CloudFormation not only creates resources on our AWS account but also waits for them to stabilize while they start. It verifies whether provisioning was successful, and if there is a failure, then it can gracefully roll the infrastructure back to a past known good state

Infrastructure as Code in CloudFormation



CloudFormation Templates

CloudFormation Templates

Templates describe the resources that we want to provision to our AWS CloudFormation stacks. We can use AWS CloudFormation Designer or any text editor to create and save templates

We can author AWS CloudFormation templates in JSON or YAML formats. All AWS CloudFormation features and functions, including CloudFormation Designer, support both formats



CloudFormation Templates

If we add comments to the YAML template created in Designer, they will not be preserved while converting the template into JSON. However, we can add comments to templates we create locally

```
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template
Resources:
  MyEC2Instance: #An inline comment
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0ff8a91507f77f867" #Another comment -- This is a Linux AMI
      InstanceType: t2.micro
      KeyName: testkey
      BlockDeviceMappings:
        -
          DeviceName: /dev/sdm
          Ebs:
            VolumeType: io1
            Iops: 200
            DeleteOnTermination: false
            VolumeSize: 20
```

There are certain specifications of JSON and YAML that CloudFormation supports

JSON

CloudFormation works with the ECMA-404 JSON standard. For more information on this format, check out this documentation:

<https://www.json.org/json-en.html>

YAML

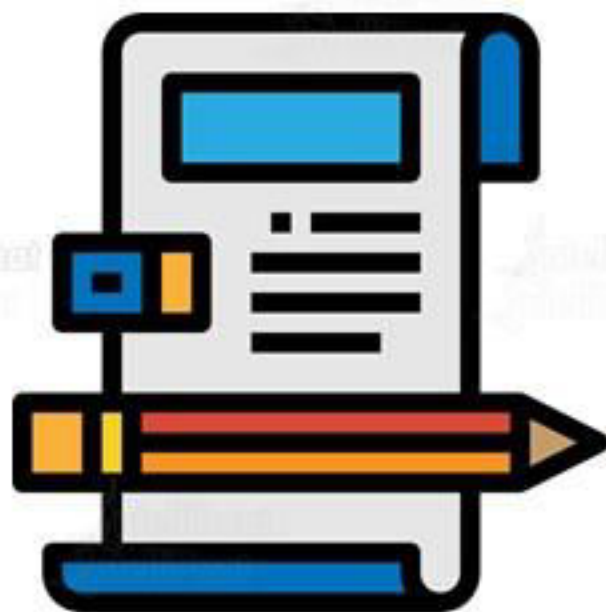
CloudFormation supports the YAML version 1.1 with a few exceptions as given below:

1. Binary, omap, pairs, set, and timestamp tags
2. Aliases
3. Hash merges

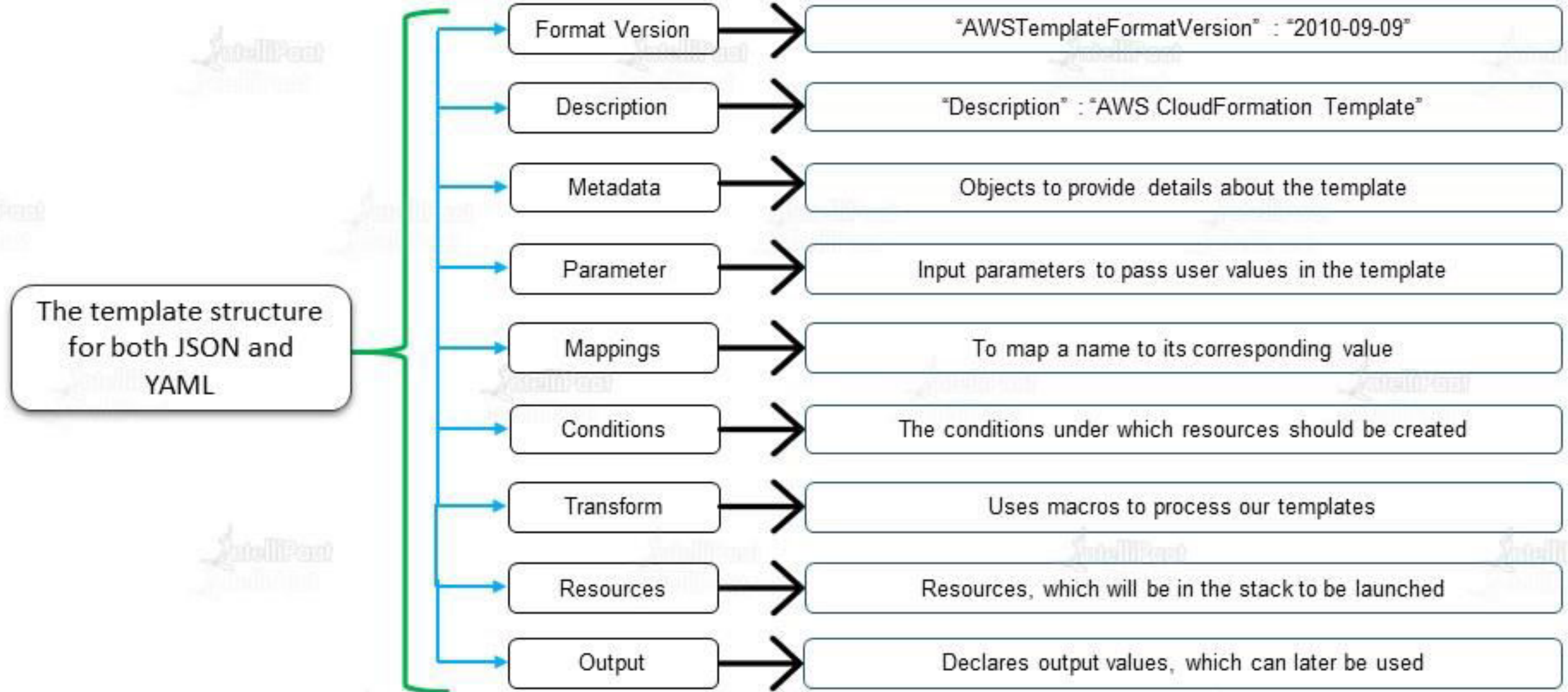
For more information: <https://yaml.org/spec/1.1/>

Template Anatomy

A template is a JSON or YAML text file, and it should be in a certain format to work with



Template Anatomy



Format Version

The latest template format version is **2010-09-09**, and it is currently the only valid value

JSON

```
"AWSTemplateFormatVersion" : "2010-09-09"
```

YAML

```
AWSTemplateFormatVersion: "2010-09-09"
```

Parameters

They enable us to input custom values to our template each time we create or update a stack

JSON

```
"Parameters" : {  
  "ParameterLogicalID" : {  
    "Type" : "DataType",  
    "ParameterProperty" : "value"  
  }  
}
```

```
"Parameters" : {  
  "InstanceTypeParameter" : {  
    "Type" : "String",  
    "Default" : "t2.micro",  
    "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],  
    "Description" : "Enter t2.micro, m1.small, or m1.large. Default is t2.micro."  
  }  
}
```

Parameters

They enable us to input custom values to our template each time we create or update a stack

YAML

Parameters:

ParameterLogicalID:

Type: DataType

ParameterProperty: value

Parameters:

InstanceTypeParameter:

Type: String

Default: t2.micro

AllowedValues:

- t2.micro
- m1.small
- m1.large

Description: Enter t2.micro, m1.small, or m1.large. Default is t2.micro.

Resources

The required Resources section declares the AWS resources that we want to include in the stack, such as an Amazon EC2 instance or an Amazon S3 bucket

JSON

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

YAML

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


Resources

JSON

```
"Resources" : {  
  "MyEC2Instance" : {  
    "Type" : "AWS::EC2::Instance",  
    "Properties" : {  
      "ImageId" : "ami-0ff8a91507f77f867"  
    }  
  }  
}
```

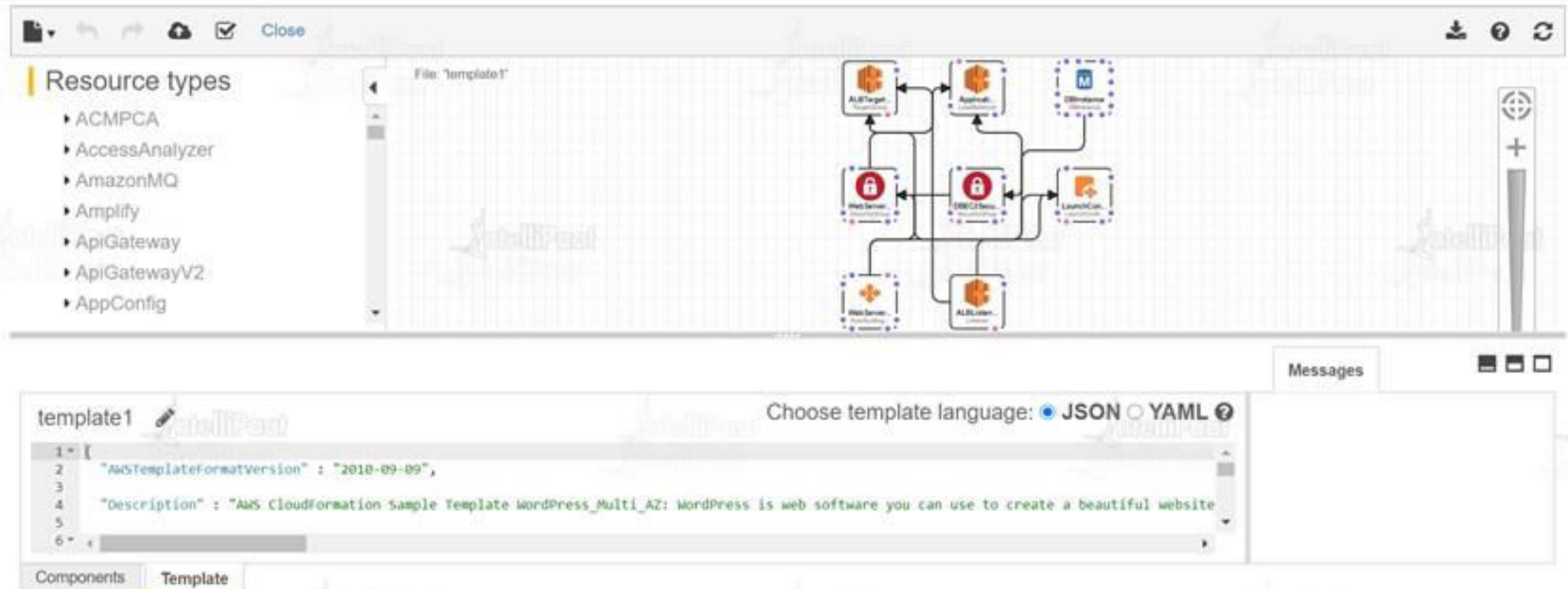
YAML

```
Resources:  
  MyEC2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: "ami-0ff8a91507f77f867"
```

CloudFormation Designer

CloudFormation Designer

AWS CloudFormation Designer is a graphic tool for creating, viewing, and modifying AWS CloudFormation templates. We can design our template resources using a drag-and-drop interface and then edit their details using the integrated JSON and YAML editor



The screenshot displays the AWS CloudFormation Designer interface. On the left, a 'Resource types' sidebar lists various AWS services like ACMPCA, AccessAnalyzer, AmazonMQ, Amplify, ApiGateway, ApiGatewayV2, and AppConfig. The main workspace shows a visual diagram of a template named 'template1', with resources represented by icons and connected by dependency lines. At the bottom, a text editor shows the JSON template code for 'template1', including fields for 'AWSTemplateFormatVersion' and 'Description'. A 'Choose template language' dropdown is set to 'JSON'. The interface also includes a 'Messages' panel on the right and a 'Components' tab at the bottom left.

```
1 {
2   "AWSTemplateFormatVersion" : "2010-09-09",
3
4   "Description" : "AWS CloudFormation Sample Template WordPress_Multi_AZ: WordPress is web software you can use to create a beautiful website
5
6 }
```

CloudFormation Designer



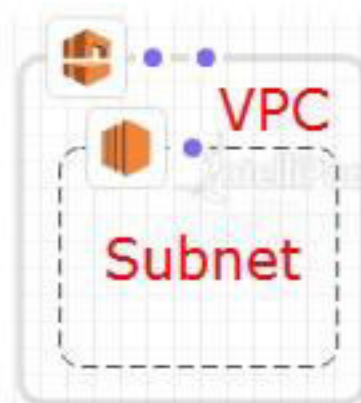
Containers - VPC



Square objects - instances



Multiple containers



CloudFormation Designer



Integrated JSON and YAML editor

template1

Choose template language: ☒ JSON ☐ YAML

```
1 {
2   "AWSTemplateFormatVersion": "2010-09-09",
3   "Description": "AWS CloudFormation Sample Template VPC_AutoScaling_and_ElasticLoadBalancer: Create a load balanced, Auto Scaled sample website",
4   "Parameters": {
5     "VpcId": {
6       "Type": "AWS::EC2::VPC::Id",
7       "Description": "VpcId of your existing Virtual Private Cloud (VPC)",
8       "ConstraintDescription": "must be the VPC Id of an existing Virtual Private Cloud."
9     },
10    "Subnets": {
11      "Type": "List<AWS::EC2::Subnet::Id>",
12      "Description": "The list of SubnetIds in your Virtual Private Cloud (VPC)",
13      "ConstraintDescription": "must be a list of an existing subnets in the selected Virtual Private Cloud."
14    },
15    "AZs": {
16      "Type": "List<String>",
17
```

Components Template

Autocomplete feature in the editor



CloudFormation Stacks

CloudFormation Stacks

A stack, for instance, can include all the resources required to run a web application, such as a web server, a database, and networking rules. If we no longer require a web application, we can simply delete the stack, and all of its related resources will be deleted



If a resource cannot be created, AWS CloudFormation rolls the stack back and automatically deletes any resources that were created

We can use AWS Management Console or AWS CLI to launch CloudFormation stacks

When using the console, we can start by clicking on the **Create stack** button

Create a CloudFormation stack

Use your own template or a sample template to quickly get started.

Create stack

We can use AWS Management Console or AWS CLI to launch CloudFormation stacks

When using the CLI, we need to follow the syntax and link our template file

```
aws cloudformation create-stack --stack-name myteststack --template-body file:///home/testuser/mytemplate.json --parameters
ParameterKey=Parm1,ParameterValue=test1 ParameterKey=Parm2,ParameterValue=test2
{
  "StackId" : "arn:aws:cloudformation:us-west-2:123456789012:stack/myteststack/330b0120-1771-11e4-af37-50ba1b98bea6"
}
```

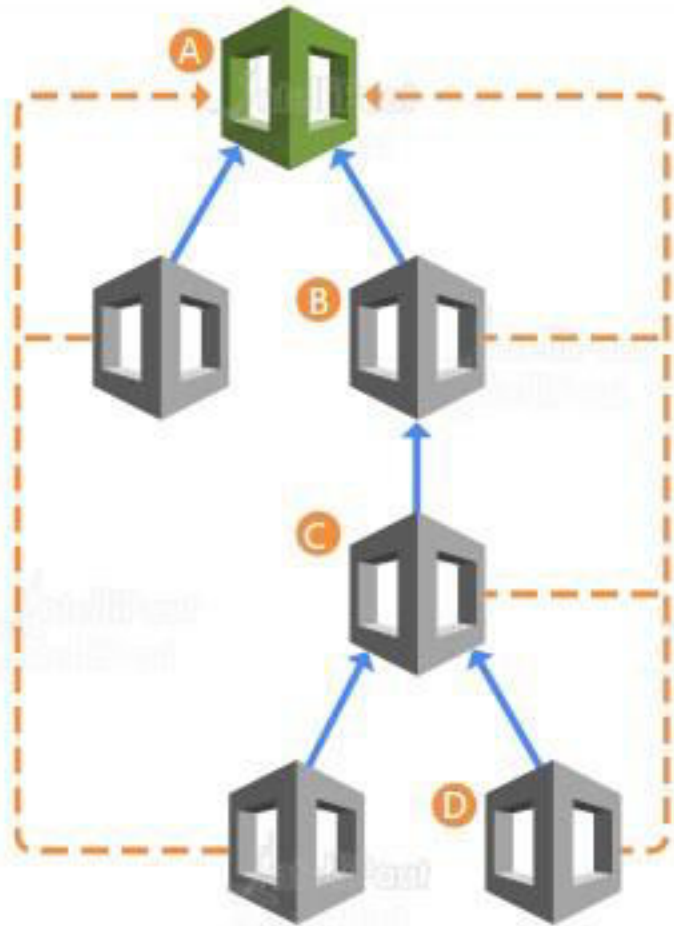

CloudFormation Stacks: Nested Stacks

Nested stacks are stacks created as part of other stacks. We can create a nested stack within another stack by using the `AWS::CloudFormation::Stack` resource



Let's say, we need a load balancer. Instead of copying and pasting the same configurations into our templates, we can create a dedicated template for this. Then, we just need to use the resource to reference that particular template from within other templates

CloudFormation Stacks: Nested Stacks

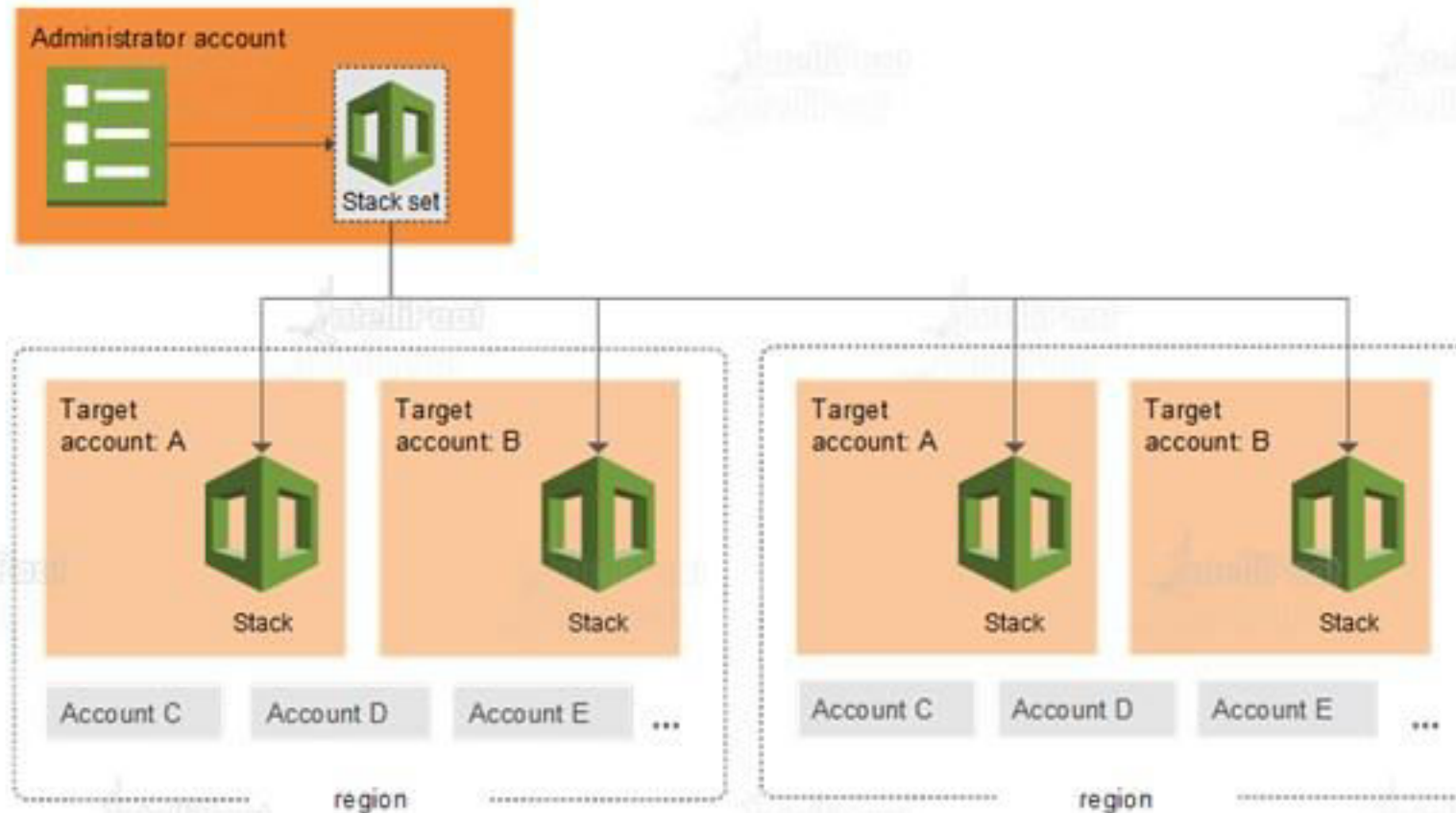


- For the first level of nested stacks, the root stack is also the parent stack
- Stack A is the root stack for all the other nested stacks in the hierarchy
- For Stack B, Stack A is both the parent stack and the root stack
- For Stack D, Stack C is the parent stack; while for Stack C, Stack B is the parent stack

CloudFormation Stack Sets

CloudFormation StackSets

StackSets extends the functionality of stacks by enabling us to create, update, or delete stacks across multiple accounts and regions with a single operation



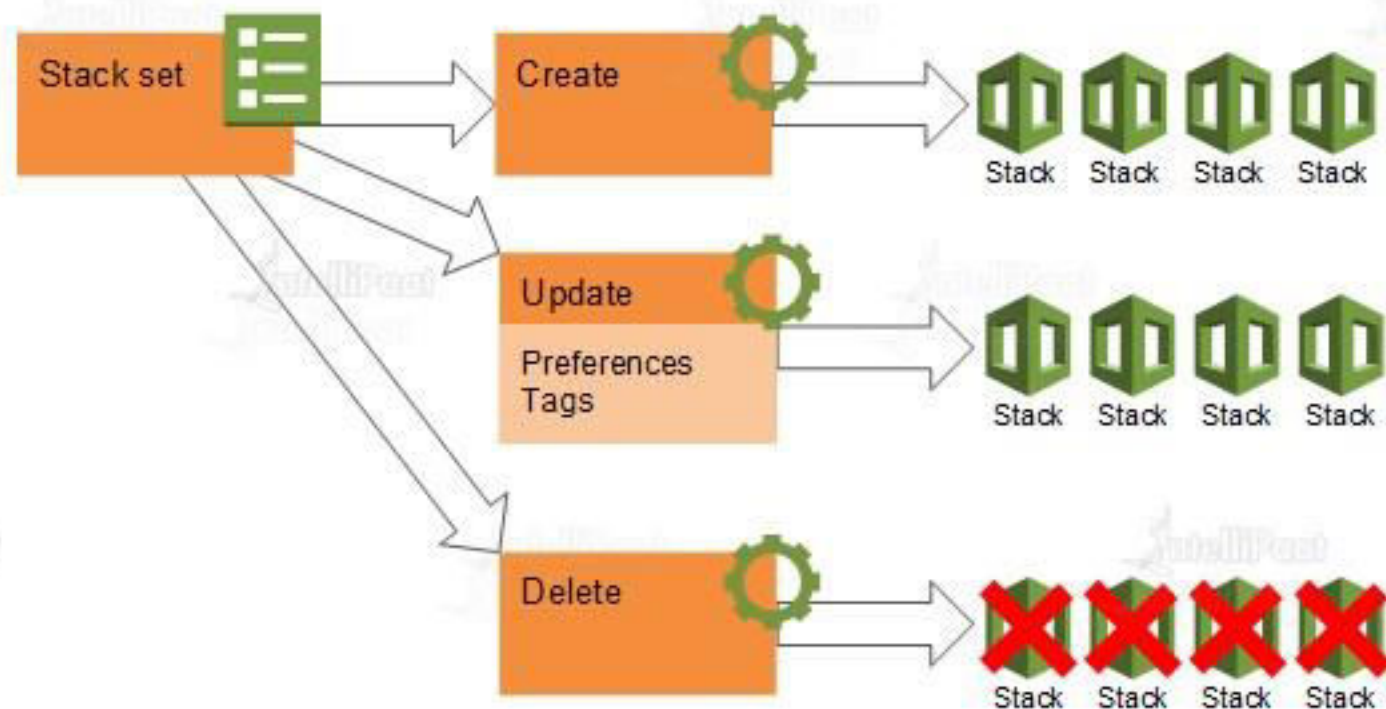
An **administrator account** is the AWS account in which we create stack sets

A **target account** is the account into which we create, update, or delete one or more stacks in our stack set

A **stack instance** is a reference to a stack in a target account within a region. A stack instance can exist without a stack; e.g., if the stack could not be created for some reason, the stack instance shows the reason for the stack creation failure

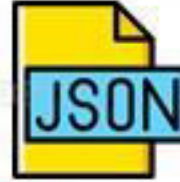
CloudFormation StackSets

Here is the logical relationship between stack sets, stack operations, and stacks. When we update a stack set, all associated stack instances are updated throughout all accounts and regions



Functions and Pseudo Parameters

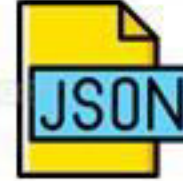
Mappings



```
"Mappings" : {  
  "mapping-name" : {  
    "key-1" : {  
      "Name" : "value-1"  
    },  
    "key-2" : {  
      "Name" : "value-2"  
    },  
    "key-3" : {  
      "Name" : "value-3"  
    }  
  }  
}
```

```
{ "Fn:FindInMap" : [ "mapping-name", "key-1", "Name" ] }
```

Mappings



```
"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"  
    },  
    "ap-southeast-1" : {  
      "32" : "ami-66f28c34", "64" : "ami-60f28c32"  
    },  
  },  
}
```

```
{ "Fn:FindInMap" : [ "mapping-name", "key-1", "Name" ] }
```

Functions

"Fn::FindInMap"

```
{ "Fn::FindInMap" : [ "MapName", "TopLevelKey", "SecondLevelKey" ] }
```

"Fn::GetAZs"

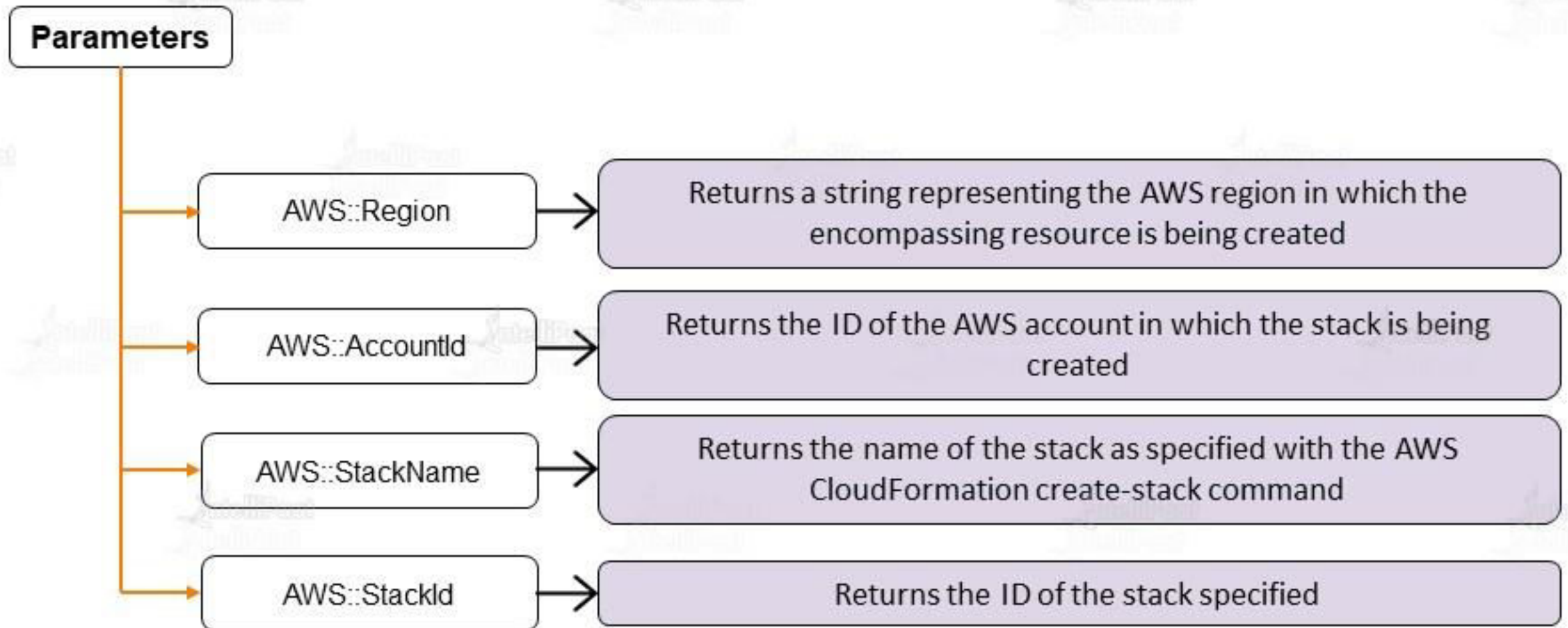
```
{ "Fn::GetAZs" : "region" }
```

"Fn::GetAtt"

```
{ "Fn::Join" : [ "delimiter", [ comma-delimited list of values ] ] }
```

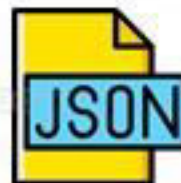
"Fn::Join"

```
{ "Fn::GetAtt" : [ "LogicalNameOfResource", "attributeName" ] }
```

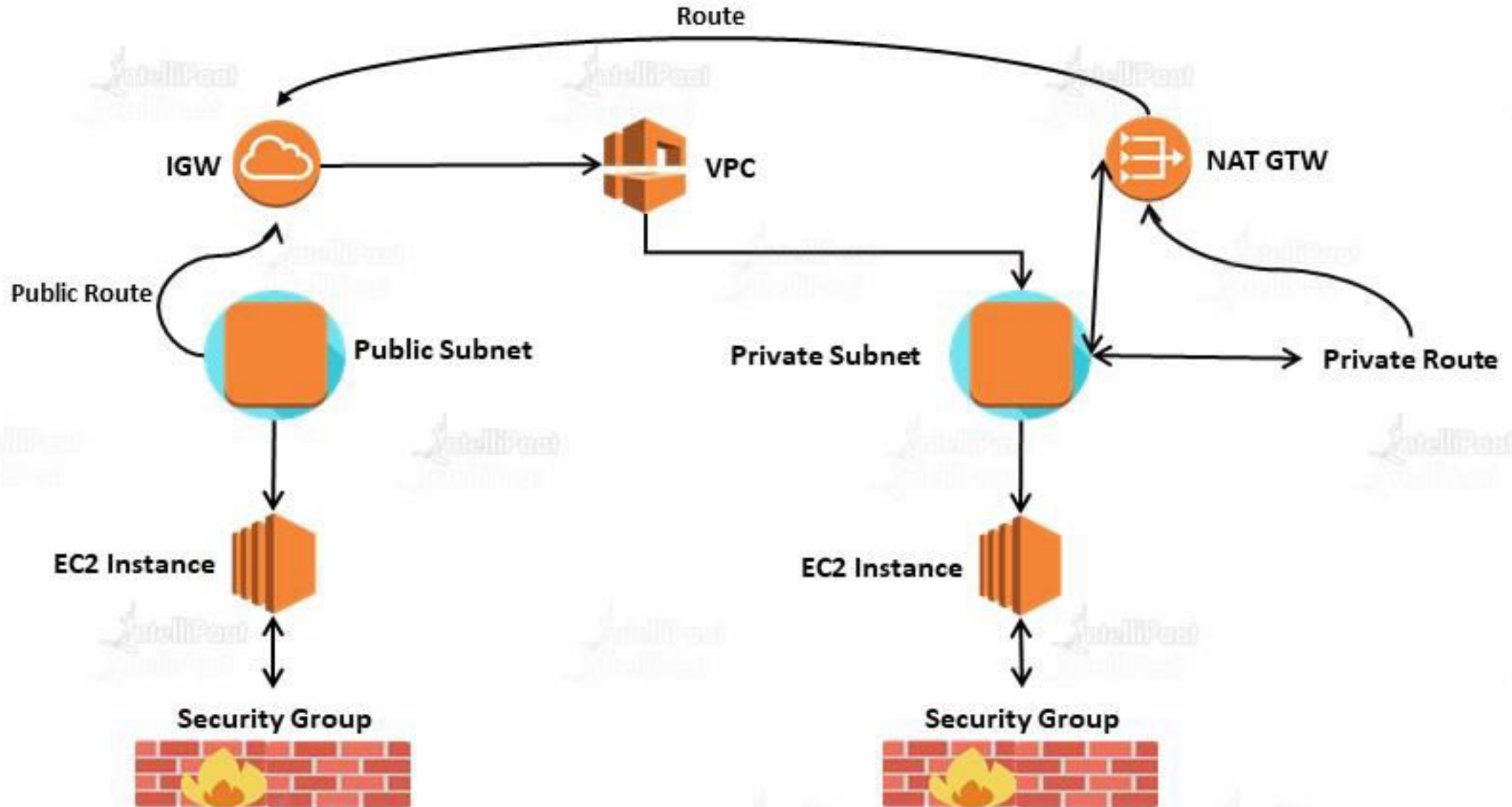
Demo: Functions and Pseudo Parameters

Functions and Pseudo Parameters



- Create a VPC with two subnets, private and public
- Launch one instance in the public subnet (inst1) and another in the private subnet (inst2)
- We should be able to ssh into inst1 from anywhere
- Inst2 should allow ssh only from inst1
- Get the following input from the user:
 - VPC CIDR block
 - CIDR block for public and private subnets
 - EC2 instance type
- Use CloudFormation to automate the deployment

CF Concepts



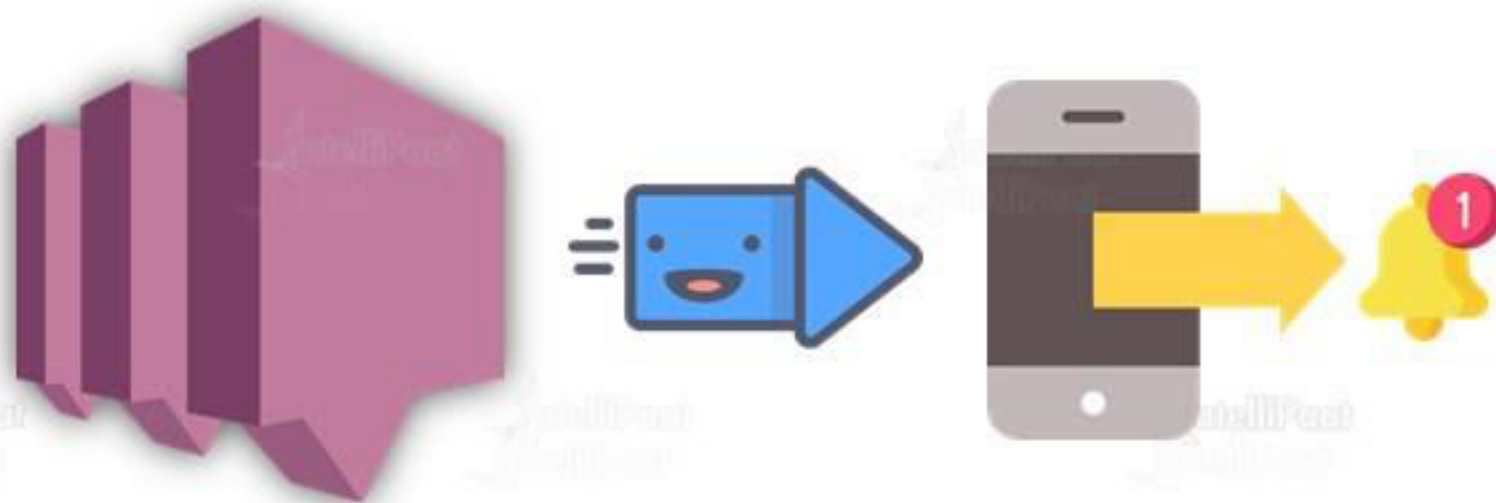
CloudFormation Best Practices

- Use the life cycle and ownership of our AWS resources to get help for deciding what resources should go in each stack
- Use IAM with AWS CloudFormation to specify what AWS CloudFormation actions users can perform, such as viewing stack templates, creating stacks, or deleting stacks
- Use nested stacks to reuse common template patterns
- Rather than embedding sensitive information like credentials in AWS CloudFormation templates, use dynamic references in our stack template
- Create change sets before updating. Change sets allow us to see how the proposed changes to a stack might impact our running resources before we implement them
- Update our Amazon EC2 Linux instances regularly

Simple Notification Service

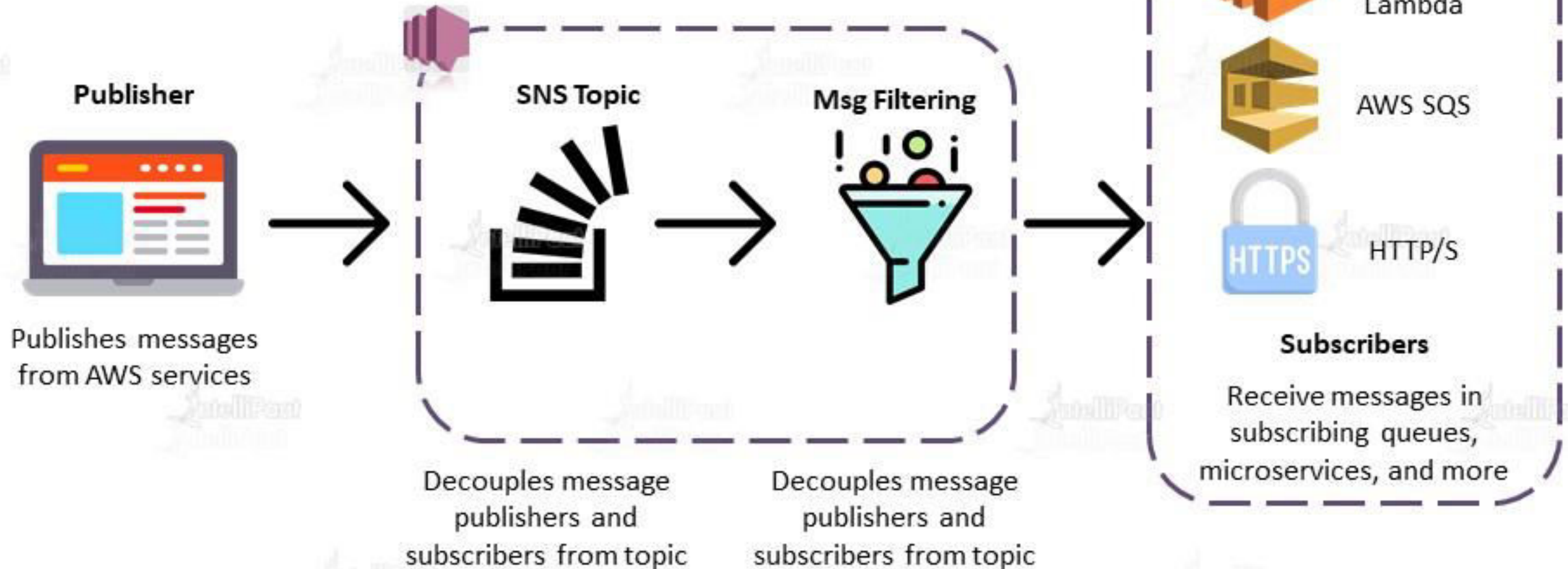
Simple Notification Service

Amazon Simple Notification Service (SNS) is a cloud-based notification service. It provides push-based and many-to-many messaging. It is easy to set up, operate, and send notifications from the cloud



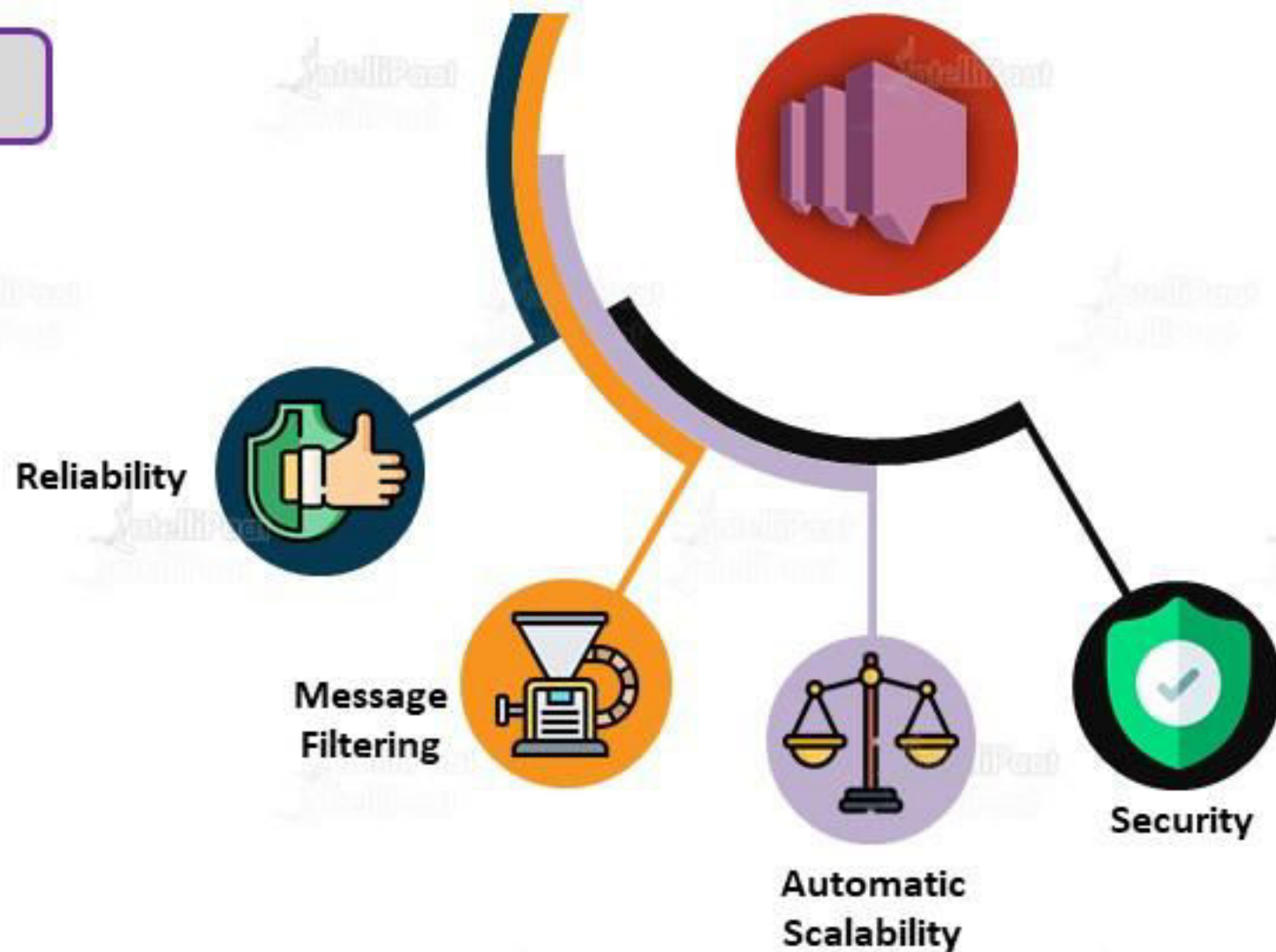
Simple Notification Service

How does SNS work?



Simple Notification Service

Benefits



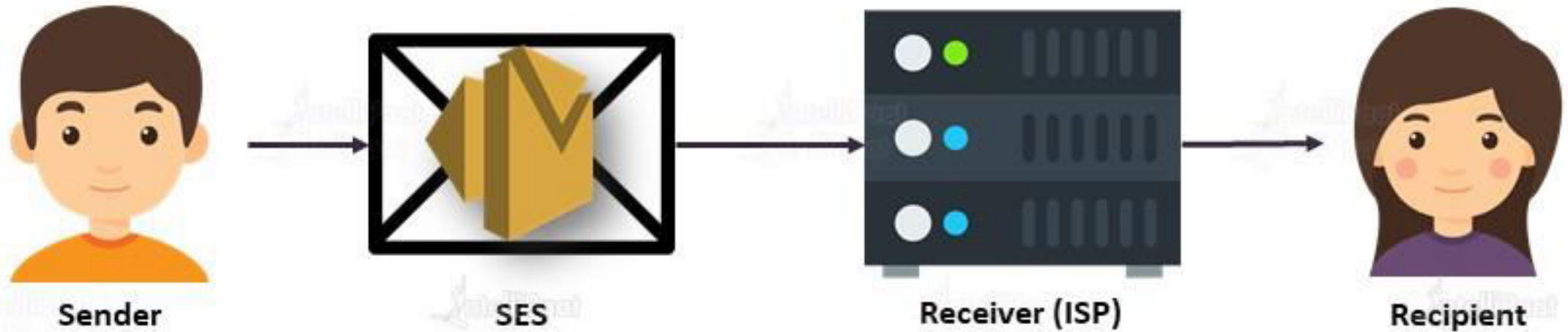
Simple Email Service

Simple Email Service

Amazon Simple Email Service (SES) is a cloud-based email sending service, which is designed to help digital marketers and developers send appropriate emails. It is reliable and cost-effective



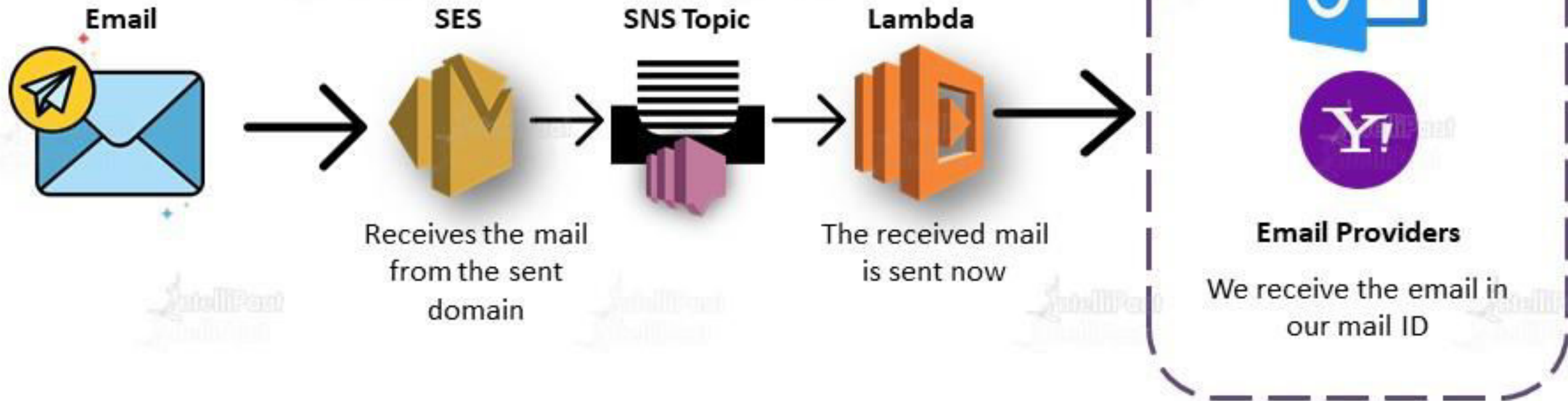
Simple Email Service



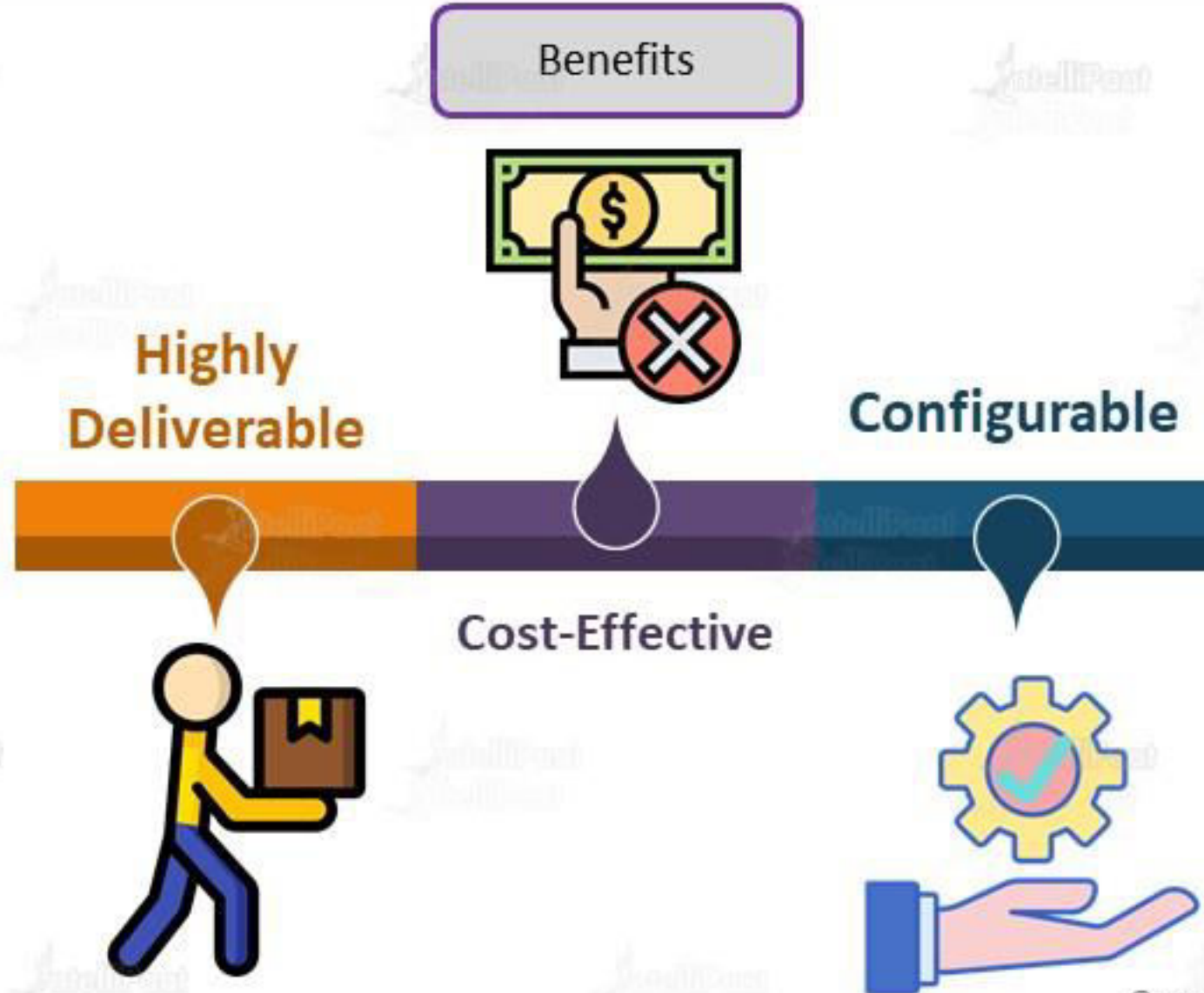
In simple terms, a sender sends a mail. SES collects it and pushes to the receiver (say, ISP), and the receiver finally sends it to the recipient

Simple Email Service

How does SES work?



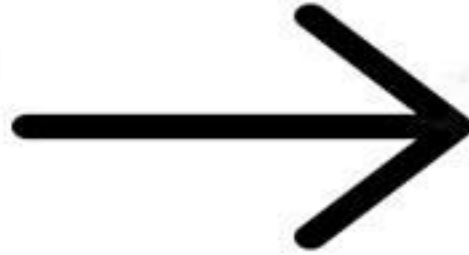
Simple Email Service



Simple Queue Service

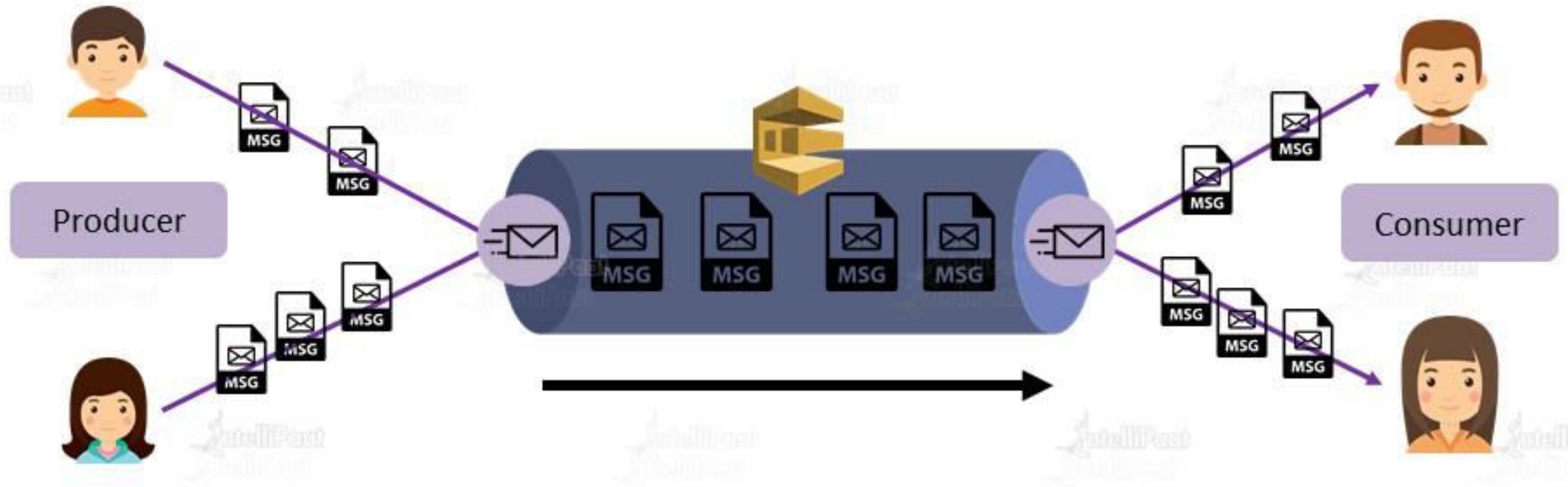
Simple Queue Service

Amazon Simple Queue Service (SQS) is a cloud-based message queuing service. It enables us to decouple and scale microservices, distributed systems, and serverless applications



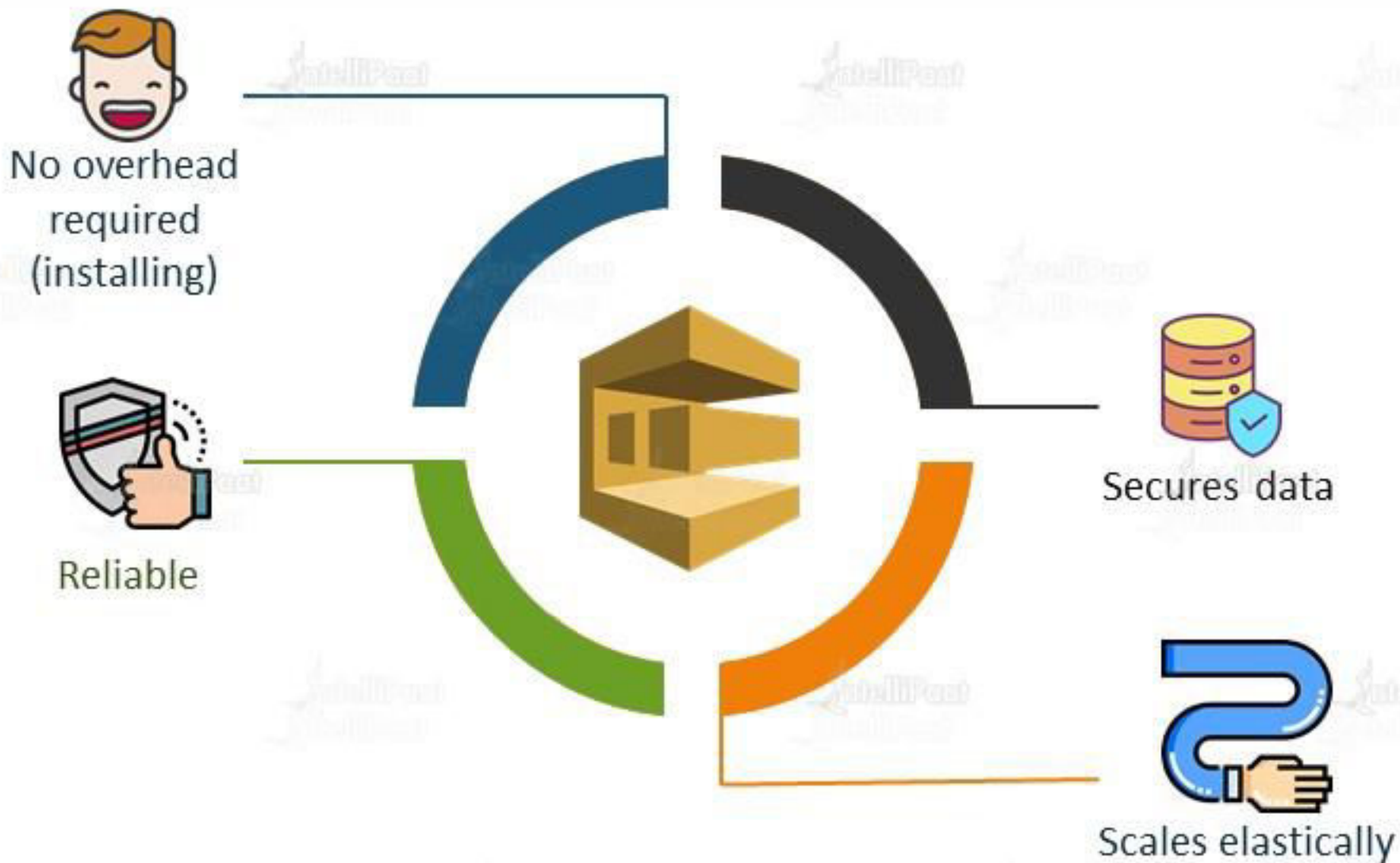
Simple Queue Service

How does SQS work?



Simple Queue Service

Benefits





India : +91-7847955955

US : 1-800-216-8930 (TOLL FREE)



support@intellipaate.com



24/7 Chat with Our Course Advisor