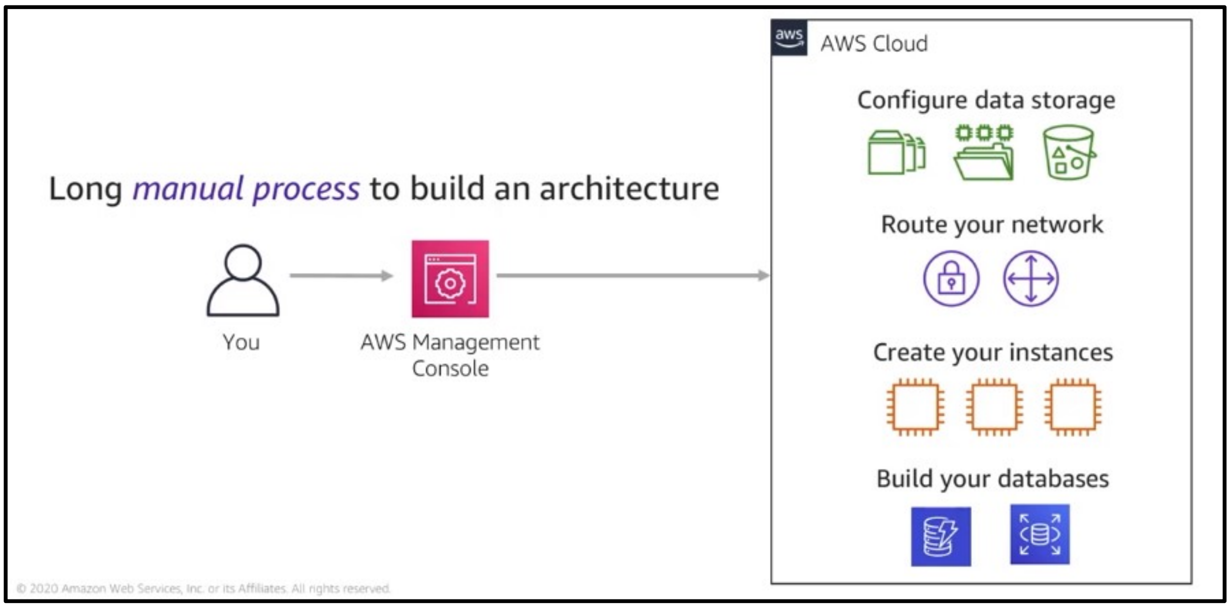
AWS CloudFormation

Introduction


Long *manual process* to build an architecture
You
AWS Management Console
AWS Cloud
Configure data storage
Route your network
Create your instances
Build your databases
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

- It takes significant time and energy to build a large-scale computing environment.
- Many organizations will start using AWS by manually creating an Amazon Simple Storage Service (Amazon S3) bucket, or launching an Amazon Elastic Compute Cloud (Amazon EC2) instance and running a web server on it.
- Then, over time, they manually add more resources as they find that expanding their use of AWS can meet additional business needs.
- Soon, however, it can become challenging to manually manage and maintain these resources.

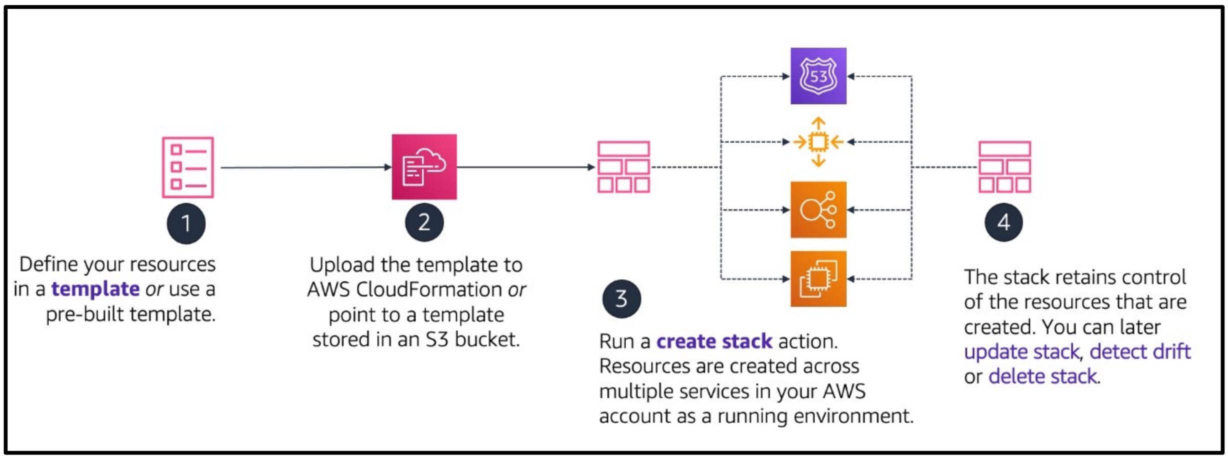
AWS CloudFormation

- You can automate the provisioning of these AWS resources with CloudFormation.
- AWS CloudFormation is a fully managed service that provides a common language for you to describe and provision all of the infrastructure resources in your cloud environment.
- CloudFormation creates, updates,and deletes the resources for your applications in environments called stacks.
- CloudFormation is all about automated resource provisioning—it simplifies the task of repeatedly and predictably creating groups of related resources that power your applications.
- AWS CloudFormation provisions resources in a repeatable manner.
- It enables you to build and rebuild your infrastructure and applications without needing to perform manual actions or write custom scripts.
- With AWS CloudFormation, you author a document that describes what your infrastructure should be, including all the AWS resources that should be a part of the deployment.

AWS CloudFormation

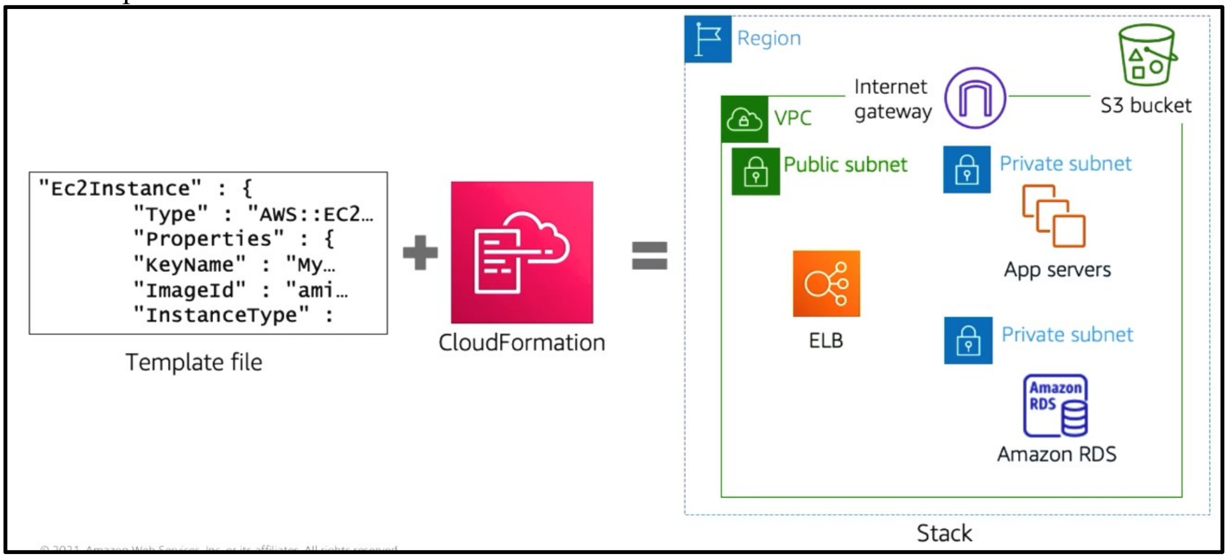
- You can think of this document as a model. You then use the model to create the reality, because AWS CloudFormation can actually create the resources in your account.
- When you use AWS CloudFormation to create resources, it is called an AWS CloudFormation stack.
- You create a stack, update a stack, or delete a stack. Thus, you can provision resources in an orderly and predicable way.
- Using AWS CloudFormation enables you to treat your infrastructure as code (IaC). Author it with any code editor, check it into a version control systemsuch as GitHub or AWS CodeCommit, and review files with team members before you deploy into the appropriate environments.
- If the AWS CloudFormation document that you create to model your deployment is checked in to a version control system, you could always delete a stack, check out an older version of the document, and create a stack from it. With version control, you can use essential rollback capabilities
- CloudFormation supports the infrastructure needs of many types of applications, such as existing enterprise applications, legacy applications, applications built using a variety of AWS resources, and container-based solutions (including those built with AWS Elastic Beanstalk)

AWS Cloud Formation Overview

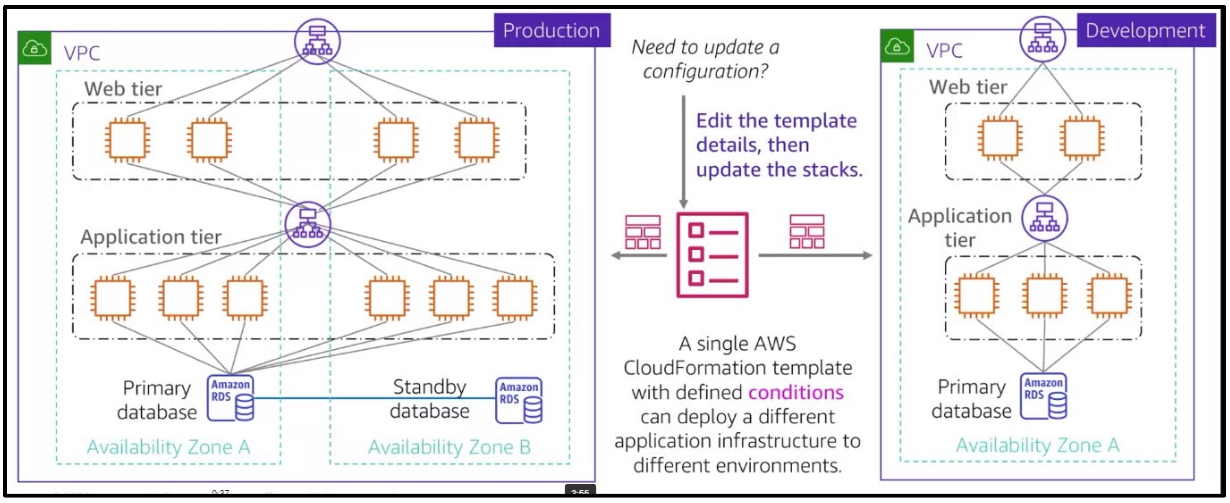

1
Define your resources in a **template** *or* use a pre-built template.
2
Upload the template to AWS CloudFormation *or* point to a template stored in an S3 bucket.
3
Run a **create stack** action. Resources are created across multiple services in your AWS account as a running environment.
4
The stack retains control of the resources that are created. You can later update stack, detect drift or delete stack.

- This diagram demonstrates how AWS CloudFormation works.
- First, you define the AWS resources that you want to create. In the example here, it creates a few EC2 instances, a load balancer, an Auto Scaling group, and an Amazon Route 53 hosted zone.
- You define the resources in an AWS CloudFormation template. You can create the template from scratch, or you can use a pre-built template. Many sample templatesare also available.
- Although AWS CloudFormation offers broad support for AWS services, not all resources can be created by AWS CloudFormation.
- Next, you upload the template to AWS CloudFormation. Alternatively, you can store the template on Amazon S3 and point AWS CloudFormation to the location where it is stored.

- Third, you run the create stack action. When you do this, the AWS CloudFormation service reads through what is specified in the template and creates the desired resources in your AWS account.
- A single stack can create and configure resources in a single Region across multiple AWS services.
- Finally, you can observe the progress of the stack-creation process. After the stack has successfully completed, the AWS resources that it created exist in your account.
- The stack object remains, and it acts like a handle to all the resources that it created. This is helpful when you want to take actions later. For example, you might want to update the stack (to create additional AWS resources or modify existing resources) or delete the stack (which will clean up and delete the resources that were created by the stack).
- Example:



```
"Ec2Instance" : {
        "Type" : "AWS::EC2…
        "Properties" : {
        "KeyName" : "My…
        "ImageId" : "ami…
        "InstanceType" :
```


VPC
Production
Web tier
Application tier
Primary database
Standby database
Availability Zone A
Availability Zone B
Need to update a configuration?
Edit the template details, then update the stacks.
A single AWS CloudFormation template with defined conditions can deploy a different application infrastructure to different environments.
VPC
Development
Web tier
Application tier
Primary database
Availability Zone A

- You can use the same AWS CloudFormation template to create both your production environment and development environment.

- This approach can help ensure that (for example) the same application binaries, same Java version, and same database version are used in both development and production. Thus, the template can help ensure that your application behaves in production the same way that it behaved in the development environment.
- In the example, you see that both the production and development environments are created from the same template.
- However, the production environment is configured to run across twoAvailability Zones and the development environment runs in a single Availability Zone. These kinds of deployment-specific differences can be accomplished by using conditions.
- You can use a Conditionsstatement in AWS CloudFormation templates to help ensure that—though they are different in size and scope—development, test, and production environments are otherwise configured identically.
- You might need several testing environments for functional testing, user acceptance testing, and load testing. Creating those environments manually is risky.
- However, creating them with AWS CloudFormation helps ensure consistency and repeatability.

CloudFormation Templates

- A template is a declaration of the AWS resources that make up a stack.
- The template is stored as a text file whose format complies with the JavaScript Object Notation (JSON) or YAML standard. Because they're text files, you can create and edit them in any text editor.
- In the template, you declare the AWS resources you want to create and configure.
- A CloudFormation template is a JSON-formatted or YAML-formatted text file that describes your AWS infrastructure.
- If you provision yourenvironment by using templates, your templates become a form of documentation for your environment.
- CloudFormation reads template files, which provide instructions for what resources need to be provisioned.
- CloudFormation constructs the resources listed in the template file. The output of this process is your environment, or stack.
- You can create a template that creates a single resource stack or a stack with hundreds of resources.
- CloudFormation template contains following sections:
    - Resources (Required)
    - Parameters
    - Conditions
    - Mappings
    - Metadata
    - Rules
    - Outputs etc.

AWS CloudFormation

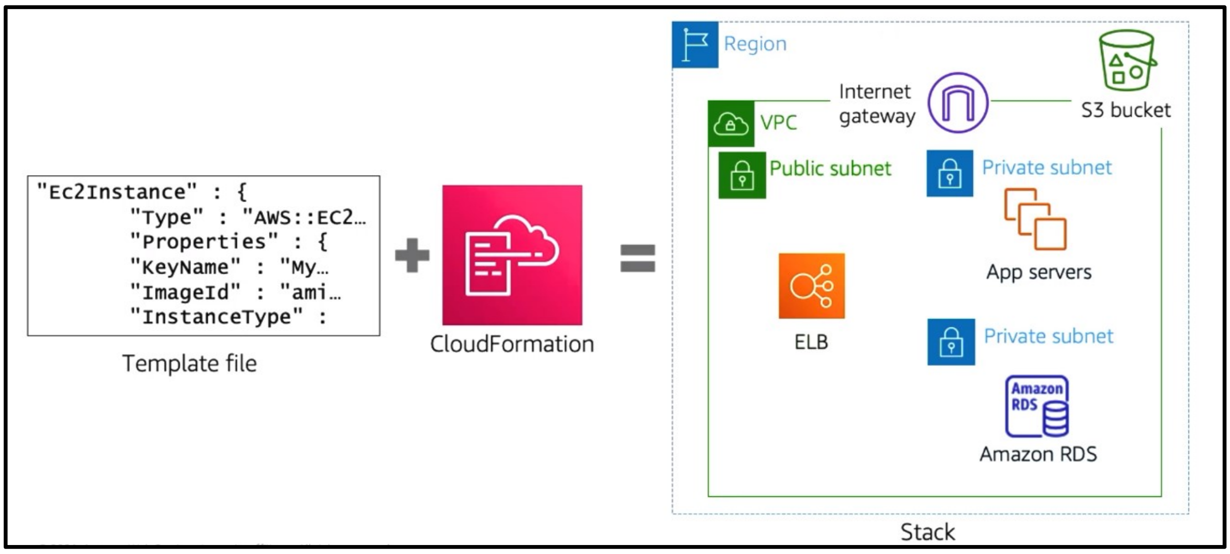
- **Template Anatomy:**

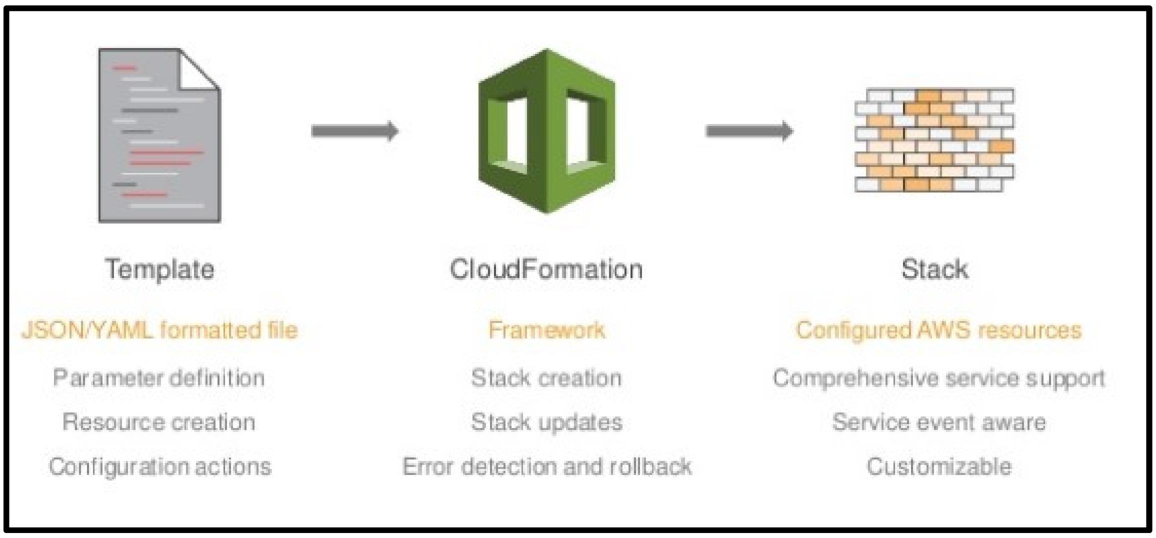
JSON Format	**YAML Format**
{	---
"AWSTemplateFormatVersion" : "version date",	AWSTemplateFormatVersion: "version date"
"Description" : "JSON string",	Description:
	String
"Metadata" : {	
template metadata	Metadata:
},	template metadata
"Parameters" : {	Parameters:
set of parameters	set of parameters
},	
	Rules:
"Rules" : {	set of rules
set of rules	
},	Mappings:
	set of mappings
"Mappings" : {	
set of mappings	Conditions:
},	set of conditions
"Conditions" : {	Transform:
set of conditions	set of transforms
},	
	Resources:
"Transform" : {	set of resources
set of transforms	
},	Outputs:
	set of outputs
"Resources" : {	
set of resources	
},	
"Outputs" : {	
set of outputs	
}	
}	

AWS CloudFormation

CloudFormation stack

- Stacks are the resources generated by a template.
- A stack is also a unit of deployment.





- You can create stacks, update stacks by running a modified template, and delete stacks.
- When you delete a stack, all of the resources in the stack are deleted by default, though this behavior can be reconfigured and overridden.
- When you author templates and create stacks, you might need to consider the quotas (formerly referred to as limits) that AWS has by default.
- If you know these quotas in advance, you can avoid limitation errors that could require you to redesign your templates or stacks.
- If you need additional stacks beyond your quota, you can request an increase through the Support Center in the console.

CloudFormation Parameters

- Use the optional Parameters section to customize your templates. Parameters enable you to input custom values to your template each time you create or update a stack.
- You can have a maximum of 200 parameters in an AWS CloudFormation template.
- Parameters must be declared and referenced from within the same template. You can reference parameters from the Resources and Outputs sections of the template.
- Defining a Parameter in a Template: The following example declares a parameter named InstanceTypeParameter. This parameter lets you specify the Amazon EC2 instance type for the stack to use when you create or update the stack.
- Note that InstanceTypeParameter has a default value of t2.micro. This is the value that AWS CloudFormation uses to provision the stack unless another value is provided.

JSON

```
"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."
  }
}
```

YAML

```
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.
```

- **Referencing a parameter within a template:** You use the Ref intrinsic function to reference a parameter, and AWS CloudFormation uses the parameter's value to provision the stack. You can reference parameters from the Resources and Outputs sections of the same template.
- In the following example, the InstanceType property of the EC2 instance resource references the InstanceTypeParame

JSON

```
"Ec2Instance" : {
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
    "InstanceType" : { "Ref" : "InstanceTypeParameter" },
    "ImageId" : "ami-0ff8a91507f77f867"
  }
}
```

YAML

```
Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType:
      Ref: InstanceTypeParameter
    ImageId: ami-0ff8a91507f77f867
```

CloudFormation Resources

- The required Resources section declares the AWS resources that you want to include in the stack, such as an Amazon EC2 instance or an Amazon S3 bucket.
- **Syntax:**

JSON	YAML

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

- The logical ID must be alphanumeric (A-Za-z0-9) and unique within the template. Use the logical name to reference the resource in other parts of the template.
- The resource type identifies the type of resource that you are declaring. For example, AWS::EC2::Instance declares an EC2 instance.
- Resource properties are additional options that you can specify for a resource. For example, for each EC2 instance, you must specify an Amazon Machine Image (AMI) ID for that instance.
- **Example:**

JSON

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

YAML

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

CloudFormation Mappings

- The optional Mappings section matches a key to a corresponding set of named values.
- For example, if you want to set values based on a region, you can create a mapping that uses the region name as a key and contains the values you want to specify for each specific region.
- You use the Fn::FindInMap intrinsic function to retrieve values in a map.
- Syntax:

JSON

```
"Mappings" : {
  "Mapping01" : {
    "Key01" : {
      "Name" : "Value01"
    },
    "Key02" : {
      "Name" : "Value02"
    },
    "Key03" : {
      "Name" : "Value03"
    }
  }
}
```

YAML

```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
```

- Return a value from a mapping: You can use the Fn::FindInMap function to return a named value based on a specified key. The following example template contains an Amazon EC2 resource whose ImageId property is assigned by the FindInMap function.
- Example:

JSON

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Mappings" : {
    "RegionMap" : {
      "us-east-1"       : {"HVM64" : "ami-0ff8a91507f77f867},
      "us-west-1"       : {"HVM64" : "ami-0bdb828fd58c52235},
    "eu-west-1"       : {"HVM64" : "ami-047bb4163c506cd98"},
    "ap-northeast-1"  : {"HVM64" : "ami-06cd52961ce9f0d85"},
    "ap-southeast-1"  : {"HVM64" : "ami-08569b978cc4dfa10"}
   }
  },

  "Resources" : {
   "myEC2Instance" : {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
     "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "HVM64"]},
     "InstanceType" : "m1.small"
    }
   }
  }
}
```

YAML

```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      HVM64: ami-0ff8a91507f77f867
    us-west-1:
      HVM64: ami-0bdb828fd58c52235
    eu-west-1:
      HVM64: ami-047bb4163c506cd98
    ap-northeast-1:
      HVM64: ami-06cd52961ce9f0d85
    ap-southeast-1:
      HVM64: ami-08569b978cc4dfa10
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]
      InstanceType: m1.small
```

CloudFormation Outputs

- The optional Outputs section declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console. For example, you can output the S3 bucket name for a stack to make the bucket easier to find.
- In the following example, the output named BackupLoadBalancerDNSName returns the DNS name for the resource with the logical ID BackupLoadBalancer only when

the CreateProdResources condition is true. (The second output shows how to specify multiple outputs.)

JSON

```
"Outputs" : {
  "BackupLoadBalancerDNSName" : {
    "Description": "The DNSName of the backup load balancer",
    "Value" : { "Fn::GetAtt" : [ "BackupLoadBalancer", "DNSName" ]},
    "Condition" : "CreateProdResources"
  },
  "InstanceID" : {
    "Description": "The Instance ID",
    "Value" : { "Ref" : "EC2Instance" }
  }
}
```

YAML

```
Outputs:
  BackupLoadBalancerDNSName:
    Description: The DNSName of the backup load balancer
    Value: !GetAtt BackupLoadBalancer.DNSName
    Condition: CreateProdResources
  InstanceID:
    Description: The Instance ID
    Value: !Ref EC2Instance
```

CloudFormation Conditions

- The optional Conditions section contains statements that define the circumstances under which entities are created or configured.
- For example, you can create a condition and then associate it with a resource or output so that AWS CloudFormation only creates the resource or output if the condition is true.
- Similarly, you can associate the condition with a property so that AWS CloudFormation only sets the property to a specific value if the condition is true. If the condition is false, AWS CloudFormation sets the property to a different value that you specify.
- Example: The following sample template includes an EnvType input parameter, where you can specify prod to create a stack for production or test to create a stack for testing.
- For a production environment, AWS CloudFormation creates an Amazon EC2 instance and attaches a volume to the instance. For a test environment, AWS CloudFormation creates only the Amazon EC2 instance.
- The CreateProdResources condition evaluates to true if the EnvType parameter is equal to prod. In the sample template, the NewVolume and MountPoint resources are

associated with the CreateProdResources condition. Therefore, the resources are created only if the EnvType parameter is equal to prod.

JSON

```
{
   "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"
         }
      },
      "MountPoint": {
         "Type": "AWS::EC2::VolumeAttachment",
         "Condition": "CreateProdResources",
         "Properties": {
            "InstanceId": {
               "Ref": "EC2Instance"
            },
            "VolumeId": {
               "Ref": "NewVolume"
            },
            "Device": "/dev/sdh"
            }
          },
          "NewVolume": {
            "Type": "AWS::EC2::Volume",
            "Condition": "CreateProdResources",
            "Properties": {
              "Size": 100,
              "AvailabilityZone": {
                "Fn::GetAtt": [
                  "EC2Instance",
                  "AvailabilityZone"
                ]
              }
            }
          }
        }
      }
```

CloudFormation Intrinsic Functions

- AWS CloudFormation intrinsic functions help you manage cloud formation stacks by assigning values that are required but are only available at run time (i.e., when the stack is launched).
- For example, let's say that one of your resources depends on another resource's attributes, which are not defined at the time the template is written.
- We can use CloudFormation to dynamically access those attributes at run time.
- Here's a list of the intrinsic functions:
    - **Fn::GetAZs** – returns an array that lists Availability Zones for a specified region
    - **Fn::Select** – returns a single object from a list of objects by index
    - **Fn::Ref** – returns the value of the specified parameter or resource
    - **Fn::GetAtt** – returns the value of an attribute from a resource in the template
    - **Fn::Sub** – substitutes variables in an input string with values that you specify
    - **Fn::Join** – appends a set of values into a single value, separated by the specified delimiter

- **Ref:** The intrinsic function Ref returns the value of the specified *parameter* or *resource*. When you specify a parameter's logical name, it returns the value of the parameter. When you specify a resource's logical name, it returns a value that you can typically use to refer to that resource.
  Examples:
  JSON:
  ```
  { "Ref" : "logicalName" }
  ```
  YAML:
  ```
  Ref: logicalName
  ```

- **The Fn::GetAtt intrinsic function** returns the value of an attribute from a resource in the template. Implementing the GetAtt function you not only provide the logical name of the resource like you did for the ref function but you also have to provide a second value for your **desired attribute name.**

  Examples:

  JSON:

  ```
  { “Fn::GetAtt” : [ “logicalNameOfResource”, “attributeName” ] }
  ```

  YAML:

  ```
  Fn::GetAtt: [ logicalNameOfResource, attributeName ]
  ```

- **The intrinsic function Fn::FindInMap** returns the value corresponding to keys in a two-level map that's declared in the Mappings section.

  Example:

  JSON

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

  YAML

  ```
  Fn::FindInMap: [ MapName, TopLevelKey, SecondLevelKey ]
  ```

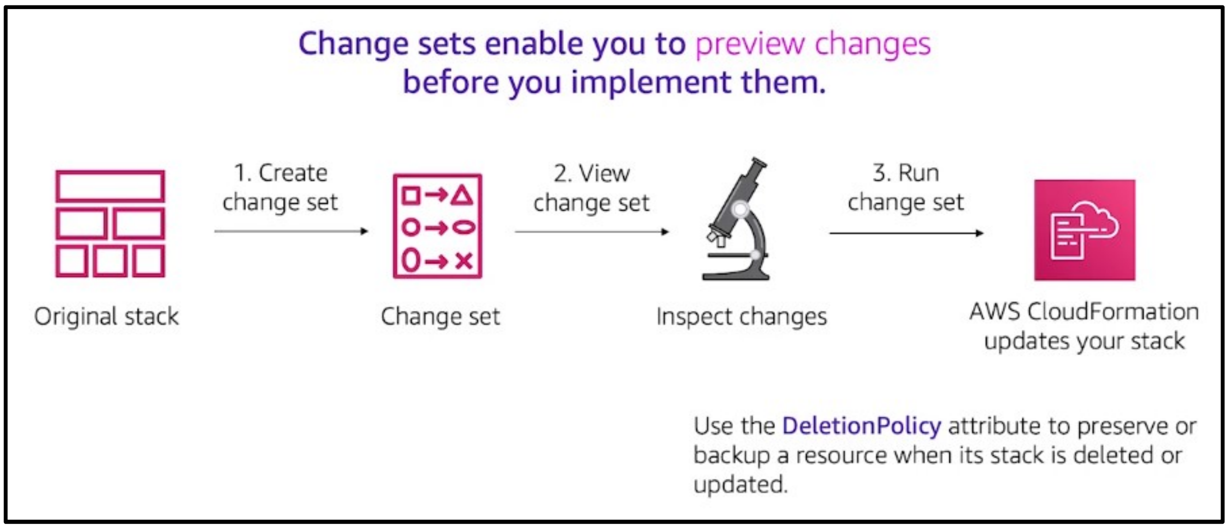
  Syntax for the short form:

  ```
  !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]
  ```

CloudFormation Rollbacks

- Rollback triggers enable you to have AWS CloudFormation monitor the state of your application during stack creation and updating, and to roll back that operation if the application breaches the threshold of the alarms you've specified.
- For each rollback trigger you create, you specify the CloudWatch alarm that CloudFormation should monitor.
- CloudFormation monitors the specified alarms during the stack create or update operation, and for the specified amount of time after all resources have been deployed.
- If any of the alarms goes to ALARM state during the stack operation or the monitoring period, CloudFormation rolls back the entire stack operation.

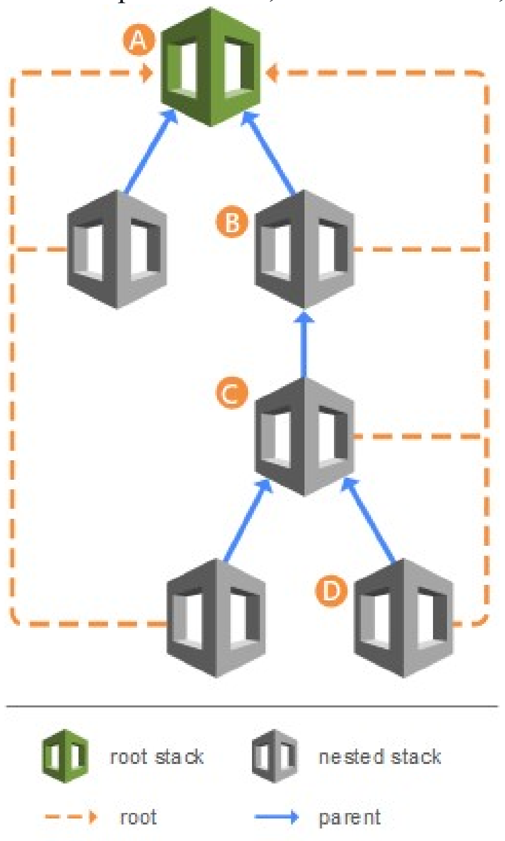
Change sets


Change sets enable you to preview changes before you implement them.
1. Create change set
2. View change set
3. Run change set
Original stack
Change set
Inspect changes
AWS CloudFormation updates your stack
Use the DeletionPolicy attribute to preserve or backup a resource when its stack is deleted or updated.

- One way to update a stack (and thus update your AWS resources) is to update the AWS CloudFormation template that you used to create the stack, and then run the Update Stack option.
- However, you might want to gain additional insight about the specific changes that AWS CloudFormation will implement if you run that command—before you actually run an update.
- If you want this type of insight, you can use a change set.
- Change sets enable you to preview the changes, verify that they align with your expectations, and then approve the updates before you proceed.
- Follow this basic workflow to use AWS CloudFormation change sets:
    1. Create a change set by submitting changes for the stack that you want to update.
    2. View the change set to see which stack settings and resources will change. If you want to consider other changes before you decide which changes to make, create additional change sets.
    3. Run the change set. AWS CloudFormation updates your stack with those changes.
- If you use change sets, you might want to set a deletion policy on some resources.
- The DeletionPolicy attribute can be used to preserve (or, in some cases, back up) a resource when its stack is deleted or updated. If a resource has no DeletionPolicy attribute, AWS CloudFormation deletes the resource.

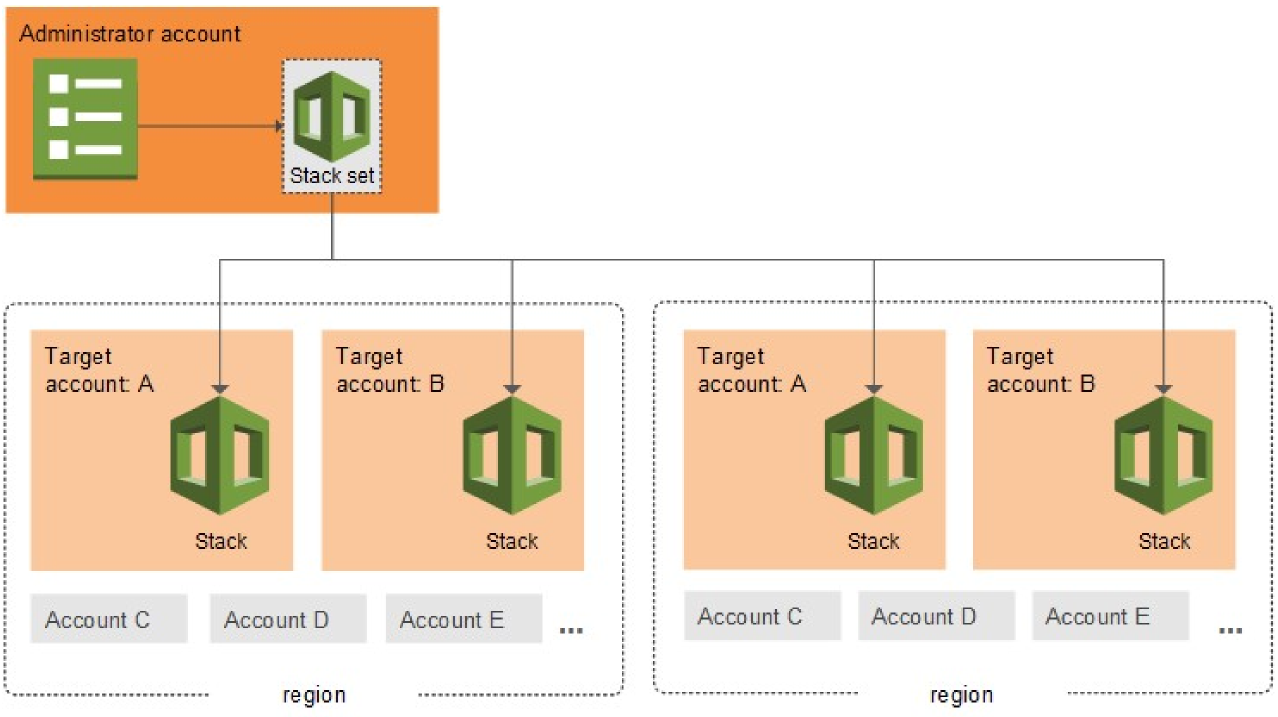
CloudFormation Nested Stacks

- *Nested stacks* are stacks created as part of other stacks.
- You create a nested stack within another stack by using the AWS::CloudFormation::Stack resource.
- *F*or example, assume that you have a load balancer configuration that you use for most of your stacks. Instead of copying and pasting the same configurations into your templates, you can create a dedicated template for the load balancer.
- Then, you just use the resource to reference that template from within other templates.
- 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. In addition, each nested stack has an immediate *parent stack*. For the first level of nested stacks, the root stack is also the parent stack.
- In the diagram below, for example:
    - 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.

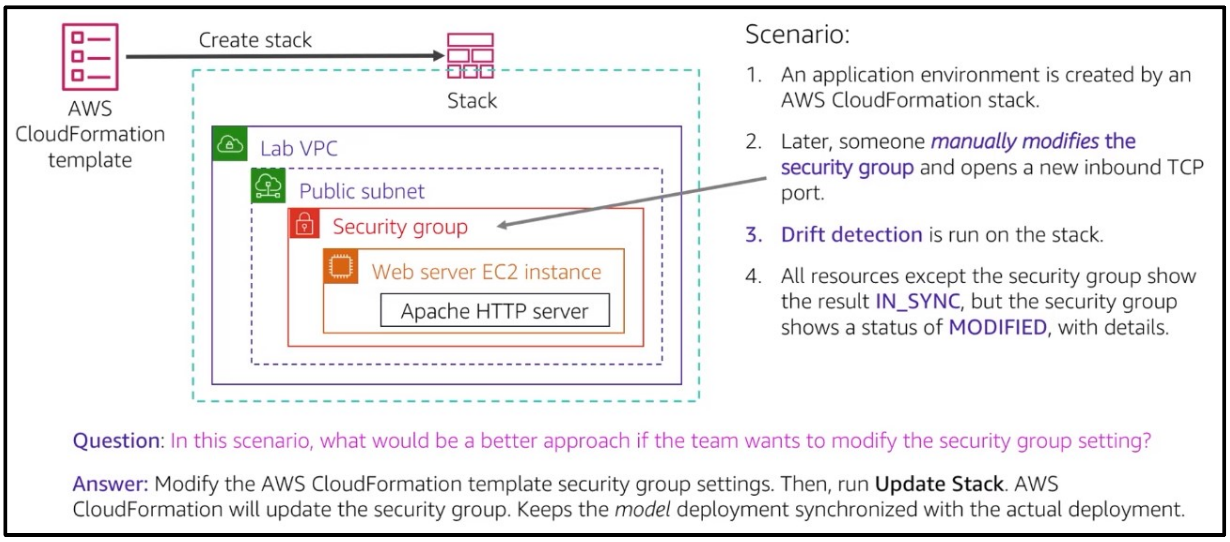


CloudFormation StackSets

- AWS CloudFormation StackSets extends the capability of stacks by enabling you to create, update, or delete stacks across multiple accounts and AWS Regions with a single operation.
- Using an administrator account, you define and manage an AWS CloudFormation template, and use the template as the basis for provisioning stacks into selected target accounts across specified AWS Regions.



CloudFormation Drift Detection



Scenario:

1. An application environment is created by an AWS CloudFormation stack.
2. Later, someone *manually modifies* the security group and opens a new inbound TCP port.
3. Drift detection is run on the stack.
4. All resources except the security group show the result IN_SYNC, but the security group shows a status of MODIFIED, with details.

Question: In this scenario, what would be a better approach if the team wants to modify the security group setting?

Answer: Modify the AWS CloudFormation template security group settings. Then, run **Update Stack**. AWS CloudFormation will update the security group. Keeps the *model* deployment synchronized with the actual deployment.

- Consider this scenario. An application environment is created by running an AWS CloudFormation stack.
- Then, someone decides to manually modify the deployed environment settings.
- They create a new inbound rule in the security group that was created by the stack.However, they make this change outside the context of AWS CloudFormation—for example, by using the Amazon EC2 console.
- As the architect of this application, you would want to know that your deployed environment no longer matches the model environment that is defined in the AWS CloudFormation template.
- How would you know which resources were modified so that they no longer exactly conform with the specifications in the stack?
- Drift detection can be run on a stack by choosing Detect Drift from the Stack actions menu in the console.
- Performing drift detection on a stack shows you whether the stack has drifted from its expected template configuration.
- Drift detection returns detailed information about the drift status of each resource that supports drift detection in the stack.
- When you delete a stack that has drift, the drift is not handled by the AWS CloudFormation resource cleanup process.
- If the stack has unresolved resource dependencies, they might cause the delete stack action to fail.
- In such cases, you might need to manually resolve the issue.