AWS CloudFormation

is a service that allows you to manage, configure and provision your AWS infrastructure as code.

AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment.

Resources are defined using a CloudFormation template.

CloudFormation interprets the template and makes the appropriate API calls to create the resources you have defined.

Supports YAML or JSON.

CloudFormation can be used to provision a broad range of AWS resources.

Think of CloudFormation as deploying infrastructure as code.

CloudFormation has some similarities with AWS Elastic Beanstalk though they are also quite different as detailed in the table below:

| **CloudFormation** | **Elastic Beanstalk** |
| --- | --- |
| “Template-driven provisioning” | “Web apps made easy” |
| Deploys infrastructure using code | Deploys applications on EC2 (PaaS) |
| Can be used to deploy almost any AWS service | Deploys web applications based on Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker |
| Uses JSON or YAML template files | Uses ZIP or WAR files |
| Similar to Terraform | Similar to Google App Engine |

## 

## **Key Benefits**

Infrastructure is provisioned consistently, with fewer mistakes (human error).

Less time and effort than configuring resources manually.

You can use version control and peer review for your CloudFormation templates.

Free to use (you’re only charged for the resources provisioned).

It can be used to manage updates and dependencies.

It can be used to rollback and delete the entire stack as well.

## **Key Concepts**

The following table describes the key concepts associated with AWS CloudFormation:

| **Component** | **Description** |
| --- | --- |
| Templates | The JSON or YAML text file that contains the instructions for building out the AWS environment |
| Stacks | The entire environment described by the template and created, updated, and deleted as a single unit |
| StackSets | AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation |
| Change Sets | A summary of proposed changes to your stack that will allow you to see how those changes might impact your existing resources before implementing them |
| Templates | The JSON or YAML text file that contains the instructions for building out the AWS environment |

## 

## **Templates**

A template is a YAML or JSON template used to describe the end-state of the infrastructure you are either provisioning or changing.

After creating the template, you upload it to CloudFormation directly or using Amazon S3.

CloudFormation reads the template and makes the API calls on your behalf.

The resulting resources are called a “Stack”.

Logical IDs are used to reference resources within the template.

Physical IDs identify resources outside of AWS CloudFormation templates, but only after the resources have been created.

### **Template elements**

Mandatory:

* List of resources and associated configuration values.

Not mandatory:

* Template parameters (limited to 60).
* Output values (limited to 60).
* List of data tables.

### **Template components**

**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.

* Mandatory.
* Represent AWS components that will be created.
* Resources are declared and can reference each other.

The following example YAML code declares an EC2 instance as a resource:

Resources:

MyEC2Instance:

Type: "AWS::EC2::Instance"

Properties:

ImageId: "ami-0ff8a91507f77f867"

### 

### **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.

* Provide inputs to your CloudFormation template.
* Useful for template reuse.

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:*** *the 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.*

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.

### **Pseudo Parameters**

Pseudo parameters are parameters that are predefined by AWS CloudFormation. You do not declare them in your template. Use them the same way as you would a parameter, as the argument for the Ref function.

Examples include:

* AWS::AccountId – Returns the AWS account ID of the account in which the stack is being created.
* AWS::NotificationARNs – Returns the list of notification Amazon Resource Names (ARNs) for the current stack.
* AWS::Region – Returns a string representing the AWS Region in which the encompassing resource is being created.
* AWS::StackId – Returns the ID of the stack as specified with the aws cloudformation create-stack command.

### 

### **Mappings**

The *optional* Mappings section matches a key to a corresponding set of named values.

* Fixed variables.
* Good for differentiating between regions, environments, AMIs etc.
* Need to know the values in advance.
* For user-specific values use parameters instead.

The following example has region keys that are mapped to two sets of values: one named HVM64 and the other HVMG2.

RegionMap:

us-east-1:

HVM64: ami-0ff8a91507f77f867

HVMG2: ami-0a584ac55a7631c0c

us-west-1:

HVM64: ami-0bdb828fd58c52235

HVMG2: ami-066ee5fd4a9ef77f1

***Exam tip:*** *with mappings you can, for example, 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.*

### 

### **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.

* Outputs can be imported into other stacks.
* Can view the outputs in the console or using the AWS CLI.
* Cannot delete a Stack if its outputs are being referenced by another CloudFormation Stack.

In the following example YAML code, the output named StackVPC returns the ID of a VPC, and then exports the value for cross-stack referencing with the name VPCID appended to the stack’s name

Outputs:

StackVPC:

Description: The ID of the VPC

Value: !Ref MyVPC

Export:

Name: !Sub "${AWS::StackName}-VPCID"

### **Conditions**

The *optional* Conditions section contains statements that define the circumstances under which entities are created or configured.

* Control the creation of resources based on a condition.
* Applied to resources and outputs.

In the sample YAML code below, resources are created only if the EnvType parameter is equal to prod:

Conditions:

CreateProdResources: !Equals [ !Ref EnvType, prod ]

### 

### **Transform**

The optional Transform section specifies one or more macros that AWS CloudFormation uses to process your template.

The transform section can be used to reference additional code stored in S3, such as Lambda code or reusable snippets of CloudFormation code.

The AWS::Serverless transform, which is a macro hosted by AWS CloudFormation, takes an entire template written in the AWS Serverless Application Model (AWS SAM) syntax and transforms and expands it into a compliant AWS CloudFormation template.

In the following example, the template uses AWS SAM syntax to simplify the declaration of a Lambda function and its execution role:

Transform: AWS::Serverless-2016-10-31

Resources:

MyServerlessFunctionLogicalID:

Type: AWS::Serverless::Function

Properties:

Handler: index.handler

Runtime: nodejs8.10

CodeUri: 's3://testBucket/mySourceCode.zip'

### 

### **Intrinsic Functions**

AWS CloudFormation provides several built-in functions that help you manage your stacks. Use intrinsic functions in your templates to assign values to properties that are not available until runtime.

***EXAM TIP:*** *At a minimum, know the intrinsic functions listed below for the exam. The full list can be found at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html*

**Ref**

* Fn::Ref (or !Ref in YAML),
* 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, such as a physical ID.

The following resource declaration for an Elastic IP address needs the instance ID of an EC2 instance and uses the Ref function to specify the instance ID of the MyEC2Instance resource:

MyEIP:

Type: "AWS::EC2::EIP"

Properties:

InstanceId: !Ref MyEC2Instance

**Fn::GetAtt**

* The Fn::GetAtt intrinsic function returns the value of an attribute from a resource in the template.
* Full syntax (YAML): Fn::GetAtt: [ logicalNameOfResource, attributeName ]
* Short form (YAML): !GetAtt logicalNameOfResource.attributeName

The following example template returns the SourceSecurityGroup.OwnerAlias and SourceSecurityGroup.GroupName of the load balancer with the logical name myELB.

*AWSTemplateFormatVersion: 2010-09-09*

*Resources:*

*myELB:*

*Type: AWS::ElasticLoadBalancing::LoadBalancer*

*Properties:*

*AvailabilityZones:*

*- eu-west-1a*

*Listeners:*

*- LoadBalancerPort: '80'*

*InstancePort: '80'*

*Protocol: HTTP*

*myELBIngressGroup:*

*Type: AWS::EC2::SecurityGroup*

*Properties:*

*GroupDescription: ELB ingress group*

*SecurityGroupIngress:*

*- IpProtocol: tcp*

*FromPort: '80'*

*ToPort: '80'*

*SourceSecurityGroupOwnerId: !GetAtt myELB.SourceSecurityGroup.OwnerAlias*

*SourceSecurityGroupName: !GetAtt myELB.SourceSecurityGroup.GroupName*

**Fn::FindInMap**

* The intrinsic function Fn::FindInMap returns the value corresponding to keys in a two-level map that is declared in the Mappings section.
* Full syntax (YAML): Fn::FindInMap: [ MapName, TopLevelKey, SecondLevelKey ]
* Short form (YAML): !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]

The following example shows how to use Fn::FindInMap for a template with a Mappings section that contains a single map, RegionMap, that associates AMIs with AWS regions:

Mappings:

RegionMap:

us-east-1:

HVM64: "ami-0ff8a91507f77f867"

HVMG2: "ami-0a584ac55a7631c0c"

us-west-1:

HVM64: "ami-0bdb828fd58c52235"

HVMG2: "ami-066ee5fd4a9ef77f1"

Resources:

myEC2Instance:

Type: "AWS::EC2::Instance"

Properties:

ImageId: !FindInMap

- RegionMap

- !Ref 'AWS::Region'

- HVM64

InstanceType: m1.small

**Fn::ImportValue**

* The intrinsic function Fn::ImportValue returns the value of an output exported by another stack.
* You typically use this function to create cross-stack references.

Fn::ImportValue:

!Sub "${NetworkStackName}-SecurityGroupID"

**Fn::Join**

* Full syntax (YAML): Fn::Join: [ delimiter, [ comma-delimited list of values ] ]
* Short form (YAML): !Join [ delimiter, [ comma-delimited list of values ] ]

The following example uses Fn::Join to construct a string value. It uses the Ref function with the Partition parameter and the AWS::AccountId pseudo parameter.

!Join

- ''

- - 'arn:'

- !Ref Partition

- ':s3:::elasticbeanstalk-\*-'

- !Ref 'AWS::AccountId'

**Fn::Sub**

* The intrinsic function Fn::Sub substitutes variables in an input string with values that you specify.
* In your templates, you can use this function to construct commands or outputs that include values that aren’t available until you create or update a stack.

The following example uses a mapping to substitute the ${Domain} variable with the resulting value from the Ref function:

Name: !Sub

- www.${Domain}

- { Domain: !Ref RootDomainName }

## 

## 

## **Stacks and Stack Sets**

### **Stacks**

Deployed resources based on templates.

Create, update, and delete stacks using templates.

Deployed through the Management Console, CLI or APIs.

Stack creation errors:

* Automatic rollback on error is enabled by default.
* You will be charged for resources provisioned even if there is an error.

Updating stacks:

* AWS CloudFormation provides two methods for updating stacks: direct update or creating and executing change sets.
* When you directly update a stack, you submit changes and AWS CloudFormation immediately deploys them.
* Use direct updates when you want to quickly deploy your updates.
* With change sets, you can preview the changes AWS CloudFormation will make to your stack, and then decide whether to apply those changes.

### **Stack Sets**

AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and 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 regions.

An administrator account is the AWS account in which you create stack sets.

A stack set is managed by signing in to the AWS administrator account in which it was created.

A target account is the account into which you create, update, or delete one or more stacks in your stack set.

Before you can use a stack set to create stacks in a target account, you must set up a trust relationship between the administrator and target accounts.

### 

### **Nested Stacks**

Nested stacks allow re-use of CloudFormation code for common use cases.

For example standard configuration for a load balancer, web server, application server etc.

Instead of copying out the code each time, create a standard template for each common use case and reference from within your CloudFormation template.

## **Best Practices**

AWS provides Python “helper scripts” which can help you install software and start services on your EC2 instances.

* Use CloudFormation to make changes to your landscape rather than going directly into the resources.
* Make use of Change Sets to identify potential trouble spots in your updates.
* Use Stack Policies to explicitly protect sensitive portions of your stack.
* Use a version control system such as CodeCommit or GitHub to track changes to templates.

## **Serverless Application Model (SAM)**

Use SAM for deploying serverless applications using CloudFormation.

SAM is an extension to CloudFormation used to define serverless applications.

Simplified syntax for defining serverless resources: APIs, Lambda Functions, DynamoDB Tables etc.

Use the SAM CLI to package your deployment code, upload it to S3 and deploy your serverless application.

## **User data with EC2**

User data can be included in CloudFormation.

The script is passed into Fn::Base64

The user data script logs are stored in /var/log/cloud-init-output.log

Binary is available on Amazon EC2 at /opt/aws/bin/cfn-init

## 

## **CloudFormation Helper Scripts**

cfn-init:

* The cfn-init helper script reads template metadata from the AWS::CloudFormation::Init key and acts accordingly to:
* Fetch and parse metadata from AWS CloudFormation
* Install packages
* Write files to disk
* Enable/disable and start/stop services
* cfn-init does not require credentials, so you do not need to use the –access-key, –secret-key, –role, or –credential-file options.
* Logs go to /var/log/cfn-init.log

cfn-signal:

* The cfn-signal helper script signals AWS CloudFormation to indicate whether Amazon EC2 instances have been successfully created or updated.
* If you install and configure software applications on instances, you can signal AWS CloudFormation when those software applications are ready.
* You use the cfn-signal script in conjunction with a [**CreationPolicy**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-creationpolicy.html) or an Auto Scaling group with a [**WaitOnResourceSignals**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html) update policy.
* When AWS CloudFormation creates or updates resources with those policies, it suspends work on the stack until the resource receives the requisite number of signals or until the timeout period is exceeded.
* You can signal a creation policy ([**CreationPolicy**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-creationpolicy.html)) or a wait condition handle ([**WaitOnResourceSignals**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html)).

Troubleshooting errors:

* Make sure the AMI has the CloudFormation helper scripts included.
* Check that the cfn-init and cfn-signal commands have run successfully.
* Verify internet connectivity.

## 

## **Creation Policies and Wait Conditions**

* CreationPolicy attribute:
* Use the CreationPolicy attribute when you want to wait on resource configuration actions before stack creation proceeds.
* You can associate the CreationPolicy attribute with a resource to prevent its status from reaching create complete until AWS CloudFormation receives a specified number of success signals, or the timeout period is exceeded.
* To signal a resource, you can use the [**cfn-signal**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-signal.html) helper script or [**SignalResource**](https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/API_SignalResource.html) API.
* AWS CloudFormation publishes valid signals to the stack events so that you track the number of signals sent.

The following CloudFormation resources support creation policies:

* [**AWS::AutoScaling::AutoScalingGroup**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-group.html)
* [**AWS::EC2::Instance**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html)
* [**AWS::CloudFormation::WaitCondition**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-waitcondition.html)

DeletionPolicy attribute:

* With the DeletionPolicy attribute you can preserve or (in some cases) backup a resource when its stack is deleted.
* You specify a DeletionPolicy attribute for each resource that you want to control.
* If a resource has no DeletionPolicy attribute, AWS CloudFormation deletes the resource by default.

DependsOn attribute:

* With the DependsOn attribute you can specify that the creation of a specific resource follows another.
* When you add a DependsOn attribute to a resource, that resource is created only after the creation of the resource specified in the DependsOn attribute.

WaitCondition:

* Note: For Amazon EC2 and Auto Scaling resources, AWS recommends that you use a CreationPolicy attribute instead of wait conditions.
* You can use a wait condition for situations like the following:
* To coordinate stack resource creation with configuration actions that are external to the stack creation.
* To track the status of a configuration process.

UpdatePolicy Attribute (WaitOnResourceSignals)

Use the UpdatePolicy attribute to specify how AWS CloudFormation handles updates to the following resources:

* [**AWS::AutoScaling::AutoScalingGroup**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-group.html),
* [**AWS::ElastiCache::ReplicationGroup**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticache-replicationgroup)
* [**AWS::Elasticsearch::Domain**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticsearch-domain.html)
* [**AWS::Lambda::Alias**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html)

UpdateReplacePolicy attribute:

* Use the UpdateReplacePolicy attribute to retain or (in some cases) backup the existing physical instance of a resource when it is replaced during a stack update operation.

## **Rollbacks and Creation Failures**

[**Stack creation failures**](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/troubleshooting.html):

* By default everything will be deleted.
* You can optionally disable rollback (good for troubleshooting failures).

Stack update failures:

* The stack will automatically roll back to the previous known working state.
* The logs can assist with understanding what issue occurred.

## **Monitoring and Reporting**

You can monitor the progress of a stack update by viewing the stack’s events. The console’s **Events** tab displays each major step in the creation and update of the stack sorted by the time of each event with latest events on top.

For resources created by CloudFormation, use AWS monitoring and reporting tools applicable to the service.

## **Authorization and Access Control**

You can use IAM with AWS CloudFormation to control what users can do with AWS CloudFormation, such as whether they can view stack templates, create stacks, or delete stacks.

In addition to AWS CloudFormation actions, you can manage what AWS services and resources are available to each user.

That way, you can control which resources users can access when they use AWS CloudFormation.

For example, you can specify which users can create Amazon EC2 instances, terminate database instances, or update VPCs. Those same permissions are applied anytime they use AWS CloudFormation to do those actions.

## **Charges**

There is no additional charge for AWS CloudFormation.

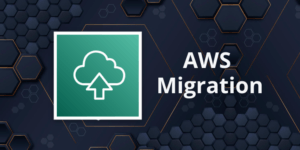
You pay for AWS resources (such as Amazon EC2 instances, Elastic Load Balancing load balancers, etc.) created using AWS CloudFormation in the same manner as if you created them manually.

You only pay for what you use, as you use it; there are no minimum fees and no required upfront commitments.

## **Snippets**

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/CHAP_TemplateQuickRef.html>

### **Related posts:**

* ****
* ****
* ****