**AWS CloudFormation**

Managing your Infrastructure as a code

**Why we need AWS CloudFormation?**

Creating and managaing multiple AWS resources in the portal is a big task, especially when you need to replace some services or setting up the duplicate environment So organization spends more time on building the infrasturceture and less focus on developing the application. Cloud formatiuon will ease that work.

**What is CloudFormation?**

AWS CloudFormation is a service that helps to set up a infrastructure as a code. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and AWS CloudFormation takes care of provisioning and configuring those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; AWS CloudFormation handles all of that

AWS CloudFormation is a service that provides you with a simple way to create and manage a collection of AWS resources by provisioning and updating them in an orderly and predictable way.

AWS cloudformation enables complete Infrastructiure or AWS resources through a text file.

**How CloudFormation works ?**

Provisioning

</>

**Output**

Use AWS CloudFormation via browser console, command line tools or APIs to create stack based on template code.

Stack

**AWS CloudFormation**

AWS CloudFormation provisions and configures the stack and resources you specified on you template.

Check out your template code locally or upload it into an s3 bucket

S3

**Amazon S3**

Code your infrastructure from scratch with the cloud formation template language in either in YAML or JSON or start from many available sample templates.

Template

**Code Infrastructure**

* Create a new template or use an existing CloudFormation template using the JSON or YAML format.
* Save your code template locally or in an S3 bucket.
* Use AWS CloudFormation to build a stack on your template.
* AWS CloudFormation constructs and configures the stack resources that you have specified in your template

**AWS CloudFormation Components**

**Templates Stack Change sets**

* **Templates**

It defines AWS resources required by user in an application. For example, in a template, you can describe an Amazon EC2 instance, such as the instance type, the AMI ID, block device mappings, and its Amazon EC2 key pair name.

n AWS CloudFormation template is a JSON or YAML formatted text file. You can save these files with any extension, such as .json, .yaml, .template, or .txt.

To create view and modify templates you can use AWS CloudFormation Desginer or any text editor tool.

**Templates has 9 objects:**

**FORMAT VERSION**

* Format version defines the capability of a template based on the version
* Example: in json & latest value of fomat version is 2010-09-09
* “AWSTemplateFormationVersion” : “2010-09-09”

**DESCRIPTION**

* Any comments about your template can be specified in the description.
* Example: “Description” : “<Give some info abt infra as a code>”

**PARAMETERS**:

* Templates can be customized using parameters. Each time you create or update your stack, parameters help you give your template custom values at runtime. Example: ssh key pair.
* Some inputs cannot be determined ahead of time.
* Dynamic Inputs for your template.
* Parameters are a way to provide inputs to your AWS cloudformation.
* Parameters can be controlled by string, Number, List, array ….etc

Example:

Parameters:

<KeyName>:

Description: The EC2 Key Pair to allow SSH access to the instance

Type: 'AWS::EC2::KeyPair::KeyName'

**<Keyname> name of the parameter. 🡺Sshsecuritygroup or sshkeypair anyname**

**How to reference a Parameter?**

* The **Fn:Ref function** can be leveraged to reference parameters and resourses as well.
* Parameter can be used anywhere in the template
* The shorthand for this yaml is **!Ref**
* **Example: VpcId: !Ref myvpc**

**RESOURCES [MANDATORY]**

The Resources section contains the definitions of the AWS resources you want to create with the template. Each resource is listed separately and specifies the properties that are necessary for creating that particular resource

* Fn::GetAtt function to get the bucket's DomainName attribute.

Resources:

HelloBucket:

Type: AWS::S3::Bucket

**MAPPINGS**

* Static/Fixed variable for your template.
* You use mappings to declare conditional values that are evaluated in a similar manner as a lookup table statement. The template uses mappings to select the correct Amazon machine image (AMI) for the region and the architecture type for the instance type.

**When you would you use mappings vs parameters?**

* Mapings are great when you know in advance all the values then can be taken and they can be deduced from variable such as
* Region Availabilty zone AWS A/C Env [Dev or prod]
* Use parameterswhen the values are really user specific.
* **Fn::FindInMap** [ Accessing mapping values]
* We use Fn::FindInMap to return a named value from a specific key.
* **Syntax: !FindInMap [ Mapname, Toplevelkey, Secondlevelkey ]**

**OUTPUTS**

* References to what has been created
* In a template, the output section describes the output values that you can import into other stacks or the values that are returned when you view your own stack properties.
* You cant’t delete a CloudFormation Stack if its output are being referenced by another stack
* For example, for an S3 bucket name, you can declare an output and use the “Description-stacks” command from the AWS CloudFormation service to make the bucket name easier to find.
* Export and Fn:: ImportValue
* Fn::Join function constructs the Target subproperty of the HealthCheck property for the ElasticLoadBalancer resource by concatenating the WebServerPort parameter with other literal strings to form the value needed. Fn::Join function is also useful for declaring output values for the stack.

**OPTIONAL RESOURCE**

**CONDITIONS**

* Conditions are used to control the creation of resources or outputs based on a condition [Env(dev,prod,qa) or aws regionor based on Any parameter value]
* In a template, conditions define whether certain resources are created or when resource properties are assigned to a value during stack creation or updating. Conditions can be used when you want to reuse the templates by creating resources in different contexts. You can use intrinsic functions to define conditions
* Conditions:

**CreateProdResources: !Equals [ !Ref EnvType, prod]**

Its true only if envtype is equal to prod

**Intrinsic function**

* Condition Functions **[Fn::And, Fn::Equals, Fn:: If, Fn:: Not, Fn:: Or]**
* **Ref**

Fn::Ref function can be leveraged to reference

Parameters ==> Returns vaule of the parameter

Resources ==> Returns the phy ID of underlying resources [ex: EC2 ID]

Shorhand ofr YAML is !Ref

* **Fn:: GetAtt**

Attributes are attached to any resources you create

Example: AZ, Private and public IP of an EC2 machine

**!GetAtt Ec2Instance.AvailabilityZone gives AZ related to Ec2instance id**

* **Fn:: FindInMap**

We use Fn:: FindInMap values are exported on other templates

For this we use the Fn:: ImportValue function

* **Fn:: ImportValue**
* **Fn:: Join**

We can join values with a delimiter

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

!Join [ “:”, [a,b,c] ]

This created “a:b:c”

* **Fn:: Sub**

Use to substitute variable from a text.

String must contain ${variable name} and will substitute them

!sub

* String
* {var1name: var1value, var2Name:var2value }

!sub string

**METADATA**

* Provides details abt the resources

Syntax : !FindInMap [ mapname, topelevelkey, seconlevelkey]

**TRANSFORM**

* Transform builds a simple declarative language for AWS CloudFormation and enables reuse of template components. Here, you can declare a single transform or multiple transforms within a template.

**STACK**

When you use AWS CloudFormation, you manage related resources as a single unit called a stack. You create, update, and delete a collection of resources by creating, updating, and deleting stacks. All the resources in a stack are defined by the stack's AWS CloudFormation template. Suppose you created a template that includes an Auto Scaling group, Elastic Load Balancing load balancer, and an Amazon Relational Database Service (Amazon RDS) database instance. To create those resources, you create a stack by submitting the template that you created, and AWS CloudFormation provisions all those resources for you. You can work with stacks by using the AWS CloudFormation [console](https://console.aws.amazon.com/cloudformation/), [API](https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/), or [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/cloudformation).

## **CloudFformation Change sets**

* When you update a stack, you need to know what changes before it happens for greater confidence.
* Change sets wont say if the update will be successfull.

Eexecute change set

View change set

Create change set

Orginal stack change set change set AWS CLoudformation

* If you need to make changes to the running resources in a stack, you update the stack. Before making changes to your resources, you can generate a change set, which is a summary of your proposed changes. Change sets allow you to see how your changes might impact your running resources, especially for critical resources, before implementing them.
* For example, if you change the name of an Amazon RDS database instance, AWS CloudFormation will create a new database and delete the old one. You will lose the data in the old database unless you've already backed it up. If you generate a change set, you will see that your change will cause your database to be replaced, and you will be able to plan accordingly before you update your stack

**CloudFormation UserData**

We can have user data at EC2 instance launch through console

We can also include it in CloudFormation

We have to pass entire script through the function **Fn::Base64**

User data script log : **/var/log/cloud-init-output.log**

We install our webserver with user data

UserData:

Fn::Base64: |

#! /bin/bash –xe

Yum update –y

Yum install –y httpd

Systemctl start httpd

Systemctl enable httpd

Echo “hello world from user data“ > /var/www/html/index.html

**cfn-init**

AWS::CloudFormation::Init must be in the Metadata of a resource

With the cfn-init script we make complex EC2 configurations readable.

Ec2 Instances will querry the CloudFormation service to get the init data

Logs go to /var/log/cfn-init.log

UserData:

Fn::Base64: |

#!/bain/bash –xe

# get the latest version of cloudformation package

yum update –y aws-cfn-bootstrap

#start cfn-init

/opt/aws/bin/cfn-init –s ${AWS::StackId} –r Myinstance –region ${AWS::Region} ||

Error\_exit ‘failed to run cfn-init’

Here Myinstance refers to Metadata which tells what packages has to install

Metadata:

Comment: Install a simple apache

AWS::CloudFormation::Init:

Config:

Packages:

Yum:

Httpd:[]

Files:

“/var/www/html/index.html”:

Content: |

<h1> hello work welcome <h1>

<p> this was created using cfn-init <p>

Mode: ‘000644’

Services:

**cfn-signal & wait conditions**

* We still don’t know how to tell Cloudformation that EC2 instance got properly configured after cfn-init
* For this we can use the cfn-signal script
* We run cfn-signal right after cfn-init
* Tell cloudformation service to keep on going or fail
* We need to define Waitcondtion
* Block the template until it receives a signal from cfn-signal
* We attach a creationPolicy [also works on EC2, ASG] it dictates how many signal you want to recive and how long has top wait

UserData:

Right after cfn-int

#start cfn-signal to the wait condition

/opt/aws/bin/cfn-signal –e $? –stack ${AWS::StackId} –resource SampleWaitCondition --region ${AWS::region}

MetaData:

SampleWaitCondition:

CreationPolicy:

ResourceSignal:

Timeout: PT2M -----> wait for 2min

Count: 1 -------> wait for 1 instance

Type: AWS::CloudFormation::WaitCondition

**CloudFormation cfn-signal failure troubleshooting**

* Wait condition didtn received the required number of signals from an Amazon EC2 instance.
* Ensure that the AMI your are using has the AWS CloudFormation helper scripts installed. If the AMI doesn’t include the helper script you can also download them to your instance
* Verify that the cfn-init & cfn-signal command was successfully run on the instance. You can view the logs such as /var/log/cloud-init.log or /var/log/cfn-init.log to help you to debug the instance launch
* You can retrieve the logs by logging in to your instance but you must disable rollback on failure or else AWS CloudFormation deletes the instances after your stack failure.
* Verify that the instance has a connection to the internet. If the instance is in vpc, the instance should be able to connect to the internet through a NAT device, if its in private subnet or through an internet gateway if it’s a public gateway.

**CloudFormation Rollbacks**

**Rollbacks on failure**

**Stack CreationFails: [CreateStack API]**

* Default: everything rolls back (gets deleted). We can look at the log OnFailure=ROOLBACK
* Troubleshoot: Option to disable rollback and manually troubleshoot Onfailure=DO\_NOTHING
* Delete: get rid of the stack entirely do no keep anything OnFailure=DELETE

**Stack Update Fails: [Update stack API]**

The stack automatically rolls back to the previous known working state.

**CloudFormation Nested Stack**

* Nested stack are stacks as part of other stack
* Nested stack are considered best practice.
* They allow you to isolate repeated patterns / common common components in separate stacks and call them from other stacks.
* To update a nested stack, always update the parent [root stack]

**Example:**

* Load balancer configuration that is reused
* Security group that is reused

**CloudFormation Deletion Policy**

**Retaing data on deletes**

* You can put a Deletion policy on any resource to control what happen wheb the cloudformation template is deleted.
* DeletionPolicy=Retain
  + Specify on resources to preserve / backup in case of CloudFormation deletes.
  + To keep a resource, specify Retain [works for any resource / nested stack]
* DeletionPolicy=Snapshot
  + EBS Volume, ElasticCache Cluster, ElasticCache Replicatio group
  + RDS, DBInstance, RDS DBCluster, Redshift Cluster

DeletePolicy=Delete [Default]

Note : for AWS ::RDS::DBCluster resources the default policy is snapshot

Note: to delete an S3 bucket you need to first empty the bucket of its content

**CloudFormation Termination Protection**

* To prevent accidental deletes of cloudformation templates use termination protection**.**

**CloudFormation Parameters from ssm [Systems Manager]**

* It is used to store centralized Parameter value in ssm, so that we can fetch these parameters value in stack.
* **Service🡪SSM🡪ParameterStore---create parameter—give ant name [**/EC2InstanceType ] and pass AMI ID of required EC2 instance and create another parameter and mention type of the ec2 [t2.micro] in value.
* Parameters:
* InstanceType:
* Type: ‘AWS::SSM::Parameter::Value<string>
* Default: /EC2InstanceType ------------------> name of the parameter available in ssm

**CloudFormation Public Parameters from ssm**

aws ssm get-parameters-by-path –-path /aws/service/ami-amazon-linux-latest –query ‘parameter[].Name’

**CloudFormation Depends on**

* Resources:
  + Ec2Iinstance:
  + Type: AWS::EC2::Instance
  + ………………
  + …………….
  + DependsOn: MyDB
* This make sure that ec2 is created only when DB iscreated.
* Basically it creates some relationship

**CloudFormation Stack Policy**