Cloud Formation

Kiran

May 13, 2021

Contents

1 AWS Cloud Formation

1.1 Infrastructure as Code(IAC):

Infrastructure as Code is the process of provisioning, managing and decommissioning infrastructure with configuration files (code) rather than interactive tools.

1.2 Introduction to Cloud Formation:

AWS CloudFormation is a service that helps provision your Amazon Web Services resources.

You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and 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; CloudFormation handles that.

1.3 Advantages of Cloud Formation

1. Simplify Infrastructure Management

2. Reusability: Replicate your infrastructure in mutliple regions or in case of DR. Cloud Formation helps you in describing your resources once and then provision the same resources over and over in multiple regions or even accounts.

3. Track Changes to Infrastructure: In case where you have to update your infrastructure incrementally, like updating your Auto scaling group, CloudFormation helps in keeping track of the changes and revert back if needed.

1.4 Important Concepts:

1. Templates:

A 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. CloudFormation uses these templates as blueprints for building your AWS resources. 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. Whenever you create a stack, you also specify a template that CloudFormation uses to create whatever you described in the template.

Example:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "A sample template",
  "Resources" : {
    "MyEC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
"ImageId" : "ami-0ff8a91507f77f867",
"InstanceType" : "t2.micro",
"KeyName" : "testkey",
"BlockDeviceMappings" : [
  {
    "DeviceName" : "/dev/sdm",
    "Ebs" : {
      "VolumeSize" : 20
    }
  }
```

```
]
      }
    }
  }

}
```

If you created a stack with the above template, CloudFormation provisions an instance with an `ami-0ff8a91507f77f867` AMI ID, `t2.micro` instance type, testkey key pair name, and an Amazon EBS volume.

The same can be written in Yaml format

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0b32c7d613c3681b8
      InstanceType: t2.micro
      KeyName: cloudops
      BlockDeviceMappings:
- DeviceName: "/dev/sdm"
  Ebs:
    DeleteOnTermination: True
    VolumeSize: 20
      Tags:
- Key: Name
  Value: CFTest
- Key: Environment
  Value: Testing
```

Note on YAML: YAML (Yaml ain't a Markup Language) is a data serialization like JSON. It's easier to write and read than json. There are no braces and commas. The spaces are important in dealing in Yaml. The Properties of an item are written with an indentation of 2 spaces.

Read More about YAML here

Anatomy of a Template

```
{
  "AWSTemplateFormatVersion" : "version date",
```

```
  "Description" : "JSON string",

  "Metadata" : {
    template metadata
  },

  "Parameters" : {
    set of parameters
  },

  "Rules" : {
    set of rules
  },

  "Mappings" : {
    set of mappings
  },

  "Conditions" : {
    set of conditions
  },

  "Transform" : {
    set of transforms
  },

  "Resources" : {
    set of resources
  },

  "Outputs" : {
    set of outputs
  }
}
```

Let's take a look at each of components

Format Version (optional): The AWS CloudFormation template version that the template conforms to. The template format version isn't the same as the API or WSDL version. The template format version can change

independently of theAPI and WSDL versions.

_ Description (optional)_: A text string that describes the template. This section must always follow the template format version section.

Metadata (optional): Objects that provide additional information about the template.

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.

Rules (optional): Validates a parameter or a combination of parameters passed to a template during a stack creation or stack update.

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): Conditions that control whether certain resources are created or whether certain resource properties are assigned a value during stack creation or update. For example, you could conditionally create a resource that depends on whether the stack is for a production or test environment.

Transform (optional): For serverless applications (also referred to as Lambda-based applications), specifies the version of the AWS Serverless Application Model (AWS SAM) to use. When you specify a transform, you can use AWS SAM syntax to declare resources in your template. The model defines the syntax that you can use and how it is processed. You can also use AWS::Include transforms to work with template snippets that are stored separately from the main AWS CloudFormation template. You can store your snippet files in an Amazon S3 bucket and then reuse the functions across multiple templates.

Resources (required): Specifies the stack resources and their properties, such as an Amazon Elastic Compute Cloud instance or an Amazon Simple Storage Service bucket. You can refer to resources in the Resources and Outputs sections of the template.

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.

Nested Stacks

Nestede

Stack sets:

A stack set lets you create stacks in AWS accounts across regions by using a single CloudFormation template. All the resources included in each stack are defined by the stack set's CloudFormation template. As you create the stack set, you specify the template to use, in addition to any parameters and capabilities that template requires.

After you've defined a stack set, you can create, update, or delete stacks in the target accounts and Regions you specify. When you create, update, or delete stacks, you can also specify operation preferences, such as the order of regions in which you want the operation to be performed, the failure tolerance beyond which stack operations stop, and the number of accounts in which operations are performed on stacks concurrently.

A stack set is a regional resource. If you create a stack set in one Region, you can't see it or change it in other Regions.