

Ch 1-1 CloudFormation Essentials

CloudFormation is essentially infrastructure as code

Code your infrastructure using json or yaml

Object oriented programming analogy: A template is like a class and a stack is like an object. Object oriented programming analogy. The stack is the living breathing instantiation of the template. The stack contains all the resources.

Since the template is comprised of code we can store the code in repositories (such as GitHub)

Therefore, we can version the code.

We can launch a basic sample stack.





Go to the following webpage which is a list of sample templates.

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html#w2ab1c27c58c13c17>

browse to the **Amazon EC2 instance in a security group** template

and hit the **Launch stack** button (see the screenshot below)

Amazon EC2

Template name	Description	View	View in designer	Launch
Amazon EC2 instance in a security group	Creates an Amazon EC2 instance in an Amazon EC2 security group.	View 	View in Designer 	 Launch 

Prepare Template: Template is ready ← leave as such

Template source: Amazon S3 URL ← leave as such (CloudFormation will create an S3 bucket to house our template)

Click **Next**

Stack Name: EC2SecurityGroupSample ← leave as such (was already prompted)

InstanceType: t2.micro ← select t2.micro instead of t2.small for this demonstration

KeyName: ← select a keypair. (If you do not have one for the region you are launching the instance in then you will need to create one (so leave this tab open). Open up a new browser tab. Go to the AWS console and select the region that you are going to be launching your stack into. Then click **services** and then click **EC2**. In the left margin click **Key Pairs**. Then click the **Create key pair** button.

Name: FirstKeyPair ← give it a name like so. Select the format (ppk or pem). Then click **Create key pair**. Now go ahead and return to the original tab we were working in.) Select **FirstKeyPair**

SSH Location: 0.0.0.0/0 ← leave it as such, even though it is not good security but it is a demonstration

Click **Next**

This prompts the configure stack options page

(We need permission to the resources defined in the stack so we need an IAM role for that)

However, if we do not define permissions then CloudFormation uses permissions based on your user credentials)

So we will not specify a role we will simply and we will not specify tags either. Simply click **Next**.

This will prompt the Review page

Just click the **Create stack** button at the bottom of the page

In a few seconds click the refresh icon in the Events section. There you will see that the security group is created and then the instance is created. In a minute click the refresh icon in the events section again and you will see that the stack creation has been completed. (If ever there were a problem with creation of the resources we would have a rollback and the resources would be destroyed and our stack rolls back as well. Ultimately, we would just delete the stack.)

Click **Services** and then Click **EC2**, then click **Running Instances**

(We can actually make changes to the instance in the EC2 console even though we created it in CloudFormation, although that is **NOT** a good practice to do so because then our stack and our template would no longer match. To alter the stack so that it does not match the template is called

“**CloudFormation DRIFT**”) So, if you create resources using CloudFormation, you should update those resources using CloudFormation!!!

Now click **Services** and click **CloudFormation**

✓ EC2SecurityGroupSample ← Select the Stack that we created called EC2SecurityGroupSample and then click **Delete** and then click **Delete stack** to confirm that you want to delete the stack

Ch 1-2 Introduction to JSON

JSON stands for JavaScript Object Notation.

It is a lightweight data interchange format. It is often used to transmit data between servers and web apps.

YAML is more compact but JSON is still used in a lot of place in AWS such as in IAM policies, and the in the CLI. So JSON is still important to understand.

JSON does not allow inline comments but YAML does allow inline comments. If you convert YAML to JSON then you are going to lose any inline comments that you made. Thus in a collaborative environment where comments are crucial, YAML is probably the best way to go.

Key Components of JSON (These are the building blocks):

Key/Value Pairs – Contain Data

Commas = Data is separated by commas

Curly Braces {} – Hold Objects

Square Brackets []-Hold Arrays

Key/Value Pair CloudFormation Examples:

“Type”:”AWS::EC2::SecurityGroup”

“Type”:”AWS::EC2::Instance”

CloudFormation Example of data separated by commas:

```
"IPProtocol": "TCP", "FromPort": "80", "ToPort": "80"
```

CloudFormation Object Example:

```
{  
  "Resources": {  
    "HelloBucket": {  
      "Type": "AWS::S3::Bucket"  
    }  
  }  
}
```

In the example Resource is an object as it is contained in curly brackets. Objects can contain other objects. HelloBucket is another object, and you can see it is enclosed in curly brackets as well.

CloudFormation Array Example:

```
["SubnetConfig", "VPC", "CIDR"]
```

By the way, arrays can be values of an object property

Such as:

```
"cars": [ "Ford", "BMW", "Fiat" ]
```

Let's look at some JSON that is in AWS.

Go to the AWS console. Click **Services** and click **IAM**

Click **Policies** in the left margin

Click on the **AdministratorAccess** policy which is near the top of the list.

We see the following JSON code:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}

```

←Here is a key value pair

Now let's look at another example, so click **Policies**

Then click **AmazonEC2FullAccess** which is further down the list

We see the following JSON code:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ec2:*",
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "elasticloadbalancing:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "cloudwatch:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "autoscaling:*",
      "Resource": "*"
    },
  ],
}

```

```

{
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:AWSServiceName": [
        "autoscaling.amazonaws.com",
        "ec2scheduled.amazonaws.com",
        "elasticloadbalancing.amazonaws.com",
        "spot.amazonaws.com",
        "spotfleet.amazonaws.com",
        "transitgateway.amazonaws.com"
      ]
    }
  }
}

```

←Condition is a key that contains an object, which contains an array

The policy as a whole is an object and as you can see it is enclosed in curly braces. And as you can see the policy is simply comprised of the key components mentioned above.

JSON recap:

- A json string contains an array of values, or an object (an array of key/value pairs).
- An array is surrounded by [] brackets, and contains a comma separated list of values.
- An Object is surrounded by {}, and contains a comma-separated list of key/value pairs.
- A key/value pair consists of a field name, followed by :, followed by the field value.
- A value in an array or object can be:
 - a number (int or float)
 - a string (in " ")
 - a Boolean (true or false)
 - another array surrounded by []
 - another object surrounded by {}
 - the Null value

Ch 1-3 Introduction to YAML

YAML templates are cleaner than JSON templates. The larger a JSON template is, the smaller the YAML version of that exact same template is in comparison.

YAML is a data serialization language designed to be readable by humans.

- YAML is a superset of JSON. This means you can parse JSON with a YAML parser.

- YAML Ain't Markup Language – It is Designed for data!!!

- Indentation conveys structure in YAML

- Do not use tabs! Use spaces.

- Whitespaces and lines have meaning in YAML, they are delimiters.

- Pros: Much more compact than JSON. And can use inline comments. Ansible Playbooks, Docker Compose, and Kubernetes use YAML. Support for complex data types.

- Cons: Not many. It may not appeal to the JavaScript crowd.

Here are the components of YAML:

Key/Value Pairs – Contain Data

Objects

YAML Arrays (can have 2 Formats)

Key/Value Pair CloudFormation Examples

Type: AWS::EC2::SecurityGroup

Type: AWS::EC2::Instance

Object Example

A list with 2 elements each

Department:

- finance : 12
IT : 8
- pre-sales : 1
marketing : 8

Array Examples

Inline Array

Instance:

[t2.micro, t2.small, t2.large]

Indented Array

Instance

- t2.micro
- t2.micro
- t2.large

YAML important points:

Indentation is a key part of YAML and represents relationships between layers of data.

Colons – key/value pairs are separated by colons.

Dashes – To represent lists of items, a single dash followed by a space is used.

Key/Value Pair – Name : John

There are two extensions you can use for a yaml file.

.yaml or alternatively you can use .yml

Below is an example of YAML code:

key/value pair examples

name : Mustang

maker : Ford

year : 2012

milage : 40,000

issues:

- hood dent
- fender rust
- brakes worn
- front tires tread wear

#objects

specs:

hp : 480

weight : 4 tons

type : 4 door

roof : hard top

#lists containing dictionaries

cars:

- Mustang:
 - year : 2012
 - make : 4 door

model : GT
- Corvette:
year : 1972
make : 2 door
model : Stringray

Let's analyze the code

issues:

- hood dent
- fender rust
- brakes worn
- front tires tread wear

← this is the notation for describing an array.
This one is an indented array but keep in mind
that alternatively we can create an inline array.
The inline array is more JSON-esque.

#objects

specs:

hp : 480
weight : 4 tons
type : 4 door
roof : hard top

← this is a dictionary object example

By the way here is a yaml validator website called YAML LINT which can check if your yaml is valid.

<http://www.yamllint.com/>

#lists containing dictionaries

cars:

- Mustang:
 - year : 2012
 - make : 4 door
 - model : GT
- Corvette:
 - year : 1972
 - make : 2 door
 - model : Stingray

←this is a list containing dictionaries. Here we have a list of cars. And then elements of that list are the Mustang and the Corvette. With further indentation we have elements of our list object which are all dictionary objects.

So when do we use a list?

issues:

- hood dent
- fender rust
- brakes worn
- front tires tread wear

Issues is one item. Issues for the use vehicle include hood dent, fender rust, brakes worn, front tires tread wear. All of which are individual issues that either exist or they do not exist...so they are NOT variable.

Where as for a dictionary (see below) the year could be variable. The model could be variable.

cars:

- Mustang:
 - year : 2012
 - make : 4 door
 - model : GT

Dictionaries are unordered. We can change the order of the dictionary and it would not matter as it is still the same information.

However, with our array list hood dent, fender rust, brakes worn, and front tires tread wear, if we change the order then we are changing the array. So if we had a second array with the same things but in a different order, those two arrays are NOT equal.

Below is a YAML template for creating an EC2 instance with a Security Group using CloudFormation.

AWSTemplateFormatVersion: 2010-09-09

Description: >-

AWS CloudFormation Sample Template EC2InstanceWithSecurityGroupSample: Create an Amazon EC2 instance running the Amazon Linux AMI. The AMI is chosen based on the region in which the stack is run. This example creates an EC2 security group for the instance to give you SSH access. ****WARNING**** This template creates an Amazon EC2 instance. You will be billed for the AWS resources used if you create a stack from this template.

Parameters:

KeyName:

Description: Name of an existing EC2 KeyPair to enable SSH access to the instance

Type: 'AWS::EC2::KeyPair::KeyName' **#here is a key value pair example**

ConstraintDescription: must be the name of an existing EC2 KeyPair.

InstanceType:

Description: WebServer EC2 instance type

Type: String

Default: t2.small

AllowedValues: **#here is an example of an array**

- t1.micro
- t2.nano
- t2.micro
- t2.small
- t2.medium
- t2.large
- m1.small
- m1.medium

- m1.large
- m1.xlarge
- m2.xlarge
- m2.2xlarge
- m2.4xlarge
- m3.medium
- m3.large
- m3.xlarge
- m3.2xlarge
- m4.large
- m4.xlarge
- m4.2xlarge
- m4.4xlarge
- m4.10xlarge
- c1.medium
- c1.xlarge
- c3.large
- c3.xlarge
- c3.2xlarge
- c3.4xlarge
- c3.8xlarge
- c4.large
- c4.xlarge
- c4.2xlarge
- c4.4xlarge
- c4.8xlarge
- g2.2xlarge
- g2.8xlarge
- r3.large
- r3.xlarge
- r3.2xlarge
- r3.4xlarge
- r3.8xlarge
- i2.xlarge
- i2.2xlarge
- i2.4xlarge
- i2.8xlarge
- d2.xlarge
- d2.2xlarge
- d2.4xlarge
- d2.8xlarge
- hi1.4xlarge

- hs1.8xlarge
- cr1.8xlarge
- cc2.8xlarge
- cg1.4xlarge

ConstraintDescription: must be a valid EC2 instance type.

SSHLocation:

Description: The IP address range that can be used to SSH to the EC2 instances

Type: String

MinLength: '9'

MaxLength: '18'

Default: 0.0.0.0/0

AllowedPattern: '(\d{1,3})\.\d{1,3})\.\d{1,3})\.\d{1,3})/(\d{1,2})'

ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

Mappings:

AWSInstanceType2Arch:

t1.micro:

Arch: HVM64

t2.nano:

Arch: HVM64

t2.micro:

Arch: HVM64

t2.small:

Arch: HVM64

t2.medium:

Arch: HVM64

t2.large:

Arch: HVM64

m1.small:

Arch: HVM64

m1.medium:

Arch: HVM64

m1.large:

Arch: HVM64

m1.xlarge:

Arch: HVM64

m2.xlarge:

Arch: HVM64

m2.2xlarge:

Arch: HVM64

m2.4xlarge:

Arch: HVM64

m3.medium:

Arch: HVM64
m3.large:
Arch: HVM64
m3.xlarge:
Arch: HVM64
m3.2xlarge:
Arch: HVM64
m4.large:
Arch: HVM64
m4.xlarge:
Arch: HVM64
m4.2xlarge:
Arch: HVM64
m4.4xlarge:
Arch: HVM64
m4.10xlarge:
Arch: HVM64
c1.medium:
Arch: HVM64
c1.xlarge:
Arch: HVM64
c3.large:
Arch: HVM64
c3.xlarge:
Arch: HVM64
c3.2xlarge:
Arch: HVM64
c3.4xlarge:
Arch: HVM64
c3.8xlarge:
Arch: HVM64
c4.large:
Arch: HVM64
c4.xlarge:
Arch: HVM64
c4.2xlarge:
Arch: HVM64
c4.4xlarge:
Arch: HVM64
c4.8xlarge:
Arch: HVM64
g2.2xlarge:

Arch: HVMG2
g2.8xlarge:
Arch: HVMG2
r3.large:
Arch: HVM64
r3.xlarge:
Arch: HVM64
r3.2xlarge:
Arch: HVM64
r3.4xlarge:
Arch: HVM64
r3.8xlarge:
Arch: HVM64
i2.xlarge:
Arch: HVM64
i2.2xlarge:
Arch: HVM64
i2.4xlarge:
Arch: HVM64
i2.8xlarge:
Arch: HVM64
d2.xlarge:
Arch: HVM64
d2.2xlarge:
Arch: HVM64
d2.4xlarge:
Arch: HVM64
d2.8xlarge:
Arch: HVM64
hi1.4xlarge:
Arch: HVM64
hs1.8xlarge:
Arch: HVM64
cr1.8xlarge:
Arch: HVM64
cc2.8xlarge:
Arch: HVM64
AWSInstanceType2NATArch:
t1.micro:
Arch: NATHVM64
t2.nano:
Arch: NATHVM64

t2.micro:
Arch: NATHVM64
t2.small:
Arch: NATHVM64
t2.medium:
Arch: NATHVM64
t2.large:
Arch: NATHVM64
m1.small:
Arch: NATHVM64
m1.medium:
Arch: NATHVM64
m1.large:
Arch: NATHVM64
m1.xlarge:
Arch: NATHVM64
m2.xlarge:
Arch: NATHVM64
m2.2xlarge:
Arch: NATHVM64
m2.4xlarge:
Arch: NATHVM64
m3.medium:
Arch: NATHVM64
m3.large:
Arch: NATHVM64
m3.xlarge:
Arch: NATHVM64
m3.2xlarge:
Arch: NATHVM64
m4.large:
Arch: NATHVM64
m4.xlarge:
Arch: NATHVM64
m4.2xlarge:
Arch: NATHVM64
m4.4xlarge:
Arch: NATHVM64
m4.10xlarge:
Arch: NATHVM64
c1.medium:
Arch: NATHVM64

c1.xlarge:
Arch: NATHVM64
c3.large:
Arch: NATHVM64
c3.xlarge:
Arch: NATHVM64
c3.2xlarge:
Arch: NATHVM64
c3.4xlarge:
Arch: NATHVM64
c3.8xlarge:
Arch: NATHVM64
c4.large:
Arch: NATHVM64
c4.xlarge:
Arch: NATHVM64
c4.2xlarge:
Arch: NATHVM64
c4.4xlarge:
Arch: NATHVM64
c4.8xlarge:
Arch: NATHVM64
g2.2xlarge:
Arch: NATHVMG2
g2.8xlarge:
Arch: NATHVMG2
r3.large:
Arch: NATHVM64
r3.xlarge:
Arch: NATHVM64
r3.2xlarge:
Arch: NATHVM64
r3.4xlarge:
Arch: NATHVM64
r3.8xlarge:
Arch: NATHVM64
i2.xlarge:
Arch: NATHVM64
i2.2xlarge:
Arch: NATHVM64
i2.4xlarge:
Arch: NATHVM64

i2.8xlarge:

Arch: NATHVM64

d2.xlarge:

Arch: NATHVM64

d2.2xlarge:

Arch: NATHVM64

d2.4xlarge:

Arch: NATHVM64

d2.8xlarge:

Arch: NATHVM64

hi1.4xlarge:

Arch: NATHVM64

hs1.8xlarge:

Arch: NATHVM64

cr1.8xlarge:

Arch: NATHVM64

cc2.8xlarge:

Arch: NATHVM64

AWSRegionArch2AMI:

us-east-1:

HVM64: ami-0080e4c5bc078760e

HVMG2: ami-0aeb704d503081ea6

us-west-2:

HVM64: ami-01e24be29428c15b2

HVMG2: ami-0fe84a5b4563d8f27

us-west-1:

HVM64: ami-0ec6517f6edbf8044

HVMG2: ami-0a7fc72dc0e51aa77

eu-west-1:

HVM64: ami-08935252a36e25f85

HVMG2: ami-0d5299b1c6112c3c7

eu-west-2:

HVM64: ami-01419b804382064e4

HVMG2: NOT_SUPPORTED

eu-west-3:

HVM64: ami-0dd7e7ed60da8fb83

HVMG2: NOT_SUPPORTED

eu-central-1:

HVM64: ami-0cfbf4f6db41068ac

HVMG2: ami-0aa1822e3eb913a11

eu-north-1:

HVM64: ami-86fe70f8

HVMG2: ami-32d55b4c
ap-northeast-1:
HVM64: ami-00a5245b4816c38e6
HVMG2: ami-09d0e0e099ecabba2
ap-northeast-2:
HVM64: ami-00dc207f8ba6dc919
HVMG2: NOT_SUPPORTED
ap-northeast-3:
HVM64: ami-0b65f69a5c11f3522
HVMG2: NOT_SUPPORTED
ap-southeast-1:
HVM64: ami-05b3bcf7f311194b3
HVMG2: ami-0e46ce0d6a87dc979
ap-southeast-2:
HVM64: ami-02fd0b06f06d93dfc
HVMG2: ami-0c0ab057a101d8ff2
ap-south-1:
HVM64: ami-0ad42f4f66f6c1cc9
HVMG2: ami-0244c1d42815af84a
us-east-2:
HVM64: ami-0cd3dfa4e37921605
HVMG2: NOT_SUPPORTED
ca-central-1:
HVM64: ami-07423fb63ea0a0930
HVMG2: NOT_SUPPORTED
sa-east-1:
HVM64: ami-05145e0b28ad8e0b2
HVMG2: NOT_SUPPORTED
cn-north-1:
HVM64: ami-053617c9d818c1189
HVMG2: NOT_SUPPORTED
cn-northwest-1:
HVM64: ami-0f7937761741dc640
HVMG2: NOT_SUPPORTED

Resources:

EC2Instance:

Type: 'AWS::EC2::Instance'

Properties:

InstanceType: !Ref InstanceType

SecurityGroups:

- !Ref InstanceSecurityGroup

KeyName: !Ref KeyName

ImageId: !FindInMap
- AWSRegionArch2AMI
- !Ref 'AWS::Region'
- !FindInMap
- AWSInstanceType2Arch
- !Ref InstanceType
- Arch

InstanceSecurityGroup:
Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: Enable SSH access via port 22

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: '22'

ToPort: '22'

CidrIp: !Ref SSHLocation

Outputs:

InstanceId:

Description: InstanceId of the newly created EC2 instance

Value: !Ref EC2Instance

AZ:

Description: Availability Zone of the newly created EC2 instance

Value: !GetAtt

- EC2Instance

- AvailabilityZone

PublicDNS:

Description: Public DNSName of the newly created EC2 instance

Value: !GetAtt

- EC2Instance

- PublicDnsName

PublicIP:

Description: Public IP address of the newly created EC2 instance

Value: !GetAtt

- EC2Instance

- PublicIp

Ch 1-4 CloudFormation and IAM Part 1

We use IAM to control what users can do with CloudFormation. Grant least privilege as necessary to prevent abuse or mistakes.

We can create specific policies for the CloudFormation permissions we want to grant to specific users/groups. Examples:

- *Create a policy for users you only want to be able to view a stack.

- *Create a policy for users to create, update, and delete stacks.

- *Users who can create/delete stacks, must also have appropriate permissions to the resources in the stack (EC2, VPC, RDS, etc.)

The anatomy of a policy is as follows

Version – Version of the policy language. Use the latest 2012-10-17 version

Statement – Container for the following elements. You can include more than one statement in a policy

SID (Optional) – Optional ID to differentiate between statements.

Effect – Allow or Deny Access

Principal – Indicate the account, user, role, or federated user to which you would like to allow or deny access

Action – List of actions that the policy allows or denies

Resource – List resources to which the actions apply

Conditional (Optional) – Specify the circumstances under which the policy grants permission.

So here again is the administrator access policy in IAM:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

Below is the policy called AWSCloudFormationReadOnlyAccess:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:Describe*",
        "cloudformation:EstimateTemplateCost",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:ValidateTemplate",
        "cloudformation:Detect*"
      ],
      "Resource": "*"
    }
  ]
}
```

So any resource within our stack is readable by anybody that is given this policy
Drift is when your stack is altered outside of CloudFormation

Here is the CloudFormationFullAccess policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

We can create a CloudFormation group and place our CloudFormation users in that group and then attach a single policy that will apply to that whole group.

In Summary:

Use IAM to control what users can do with CloudFormation such as view stack templates or create, update, and delete stacks.

When building stacks, you introduce a lot of different resources (VPCs, IGW, EC2, Subnet, Security Groups) You need to control which services/resources the user can access with IAM.

Specify who can terminate DBs, EC2 instances, or update other resources through IAM.

CloudFormation should be governed by IAM, and not be an open gate for users to do as they wish.

Practice least privilege.

When you create a user or group in IAM, you can attach a policy to that user or group which specifies the permissions you want to grant for CloudFormation.

Here are some IAM policy examples:

EXAMPLE 1)

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "cloudformation:DescribeStacks",
      "cloudformation:DescribeStackEvents",
      "cloudformation:DescribeStackResource",
      "cloudformation:DescribeStackResources"
    ],
    "Resource": "*"
  }]
}
```

EXAMPLE 2)

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "sqs:*",
      "cloudformation:CreateStack",
      "cloudformation:DescribeStacks",

```



```

        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResources",
        "cloudformation:GetTemplate",
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "*"
}

```

Example 3)

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Action": [
            "cloudformation:DeleteStack",
            "cloudformation:UpdateStack"
        ],
        "Resource": "arn:aws:cloudformation:us-east-1:123456789012:stack/MyProductionStack/*"
    }]
}

```

This policy denies update stack and delete stack

And we want this to only apply to our production stack hence we specify that in the resource section

Ch 1-5 CloudFormation and IAM Part 2

Iam users that are using CloudFormation in the AWS console require additional permissions that are not required for users that are using CloudFormation from the Command Line Interface or CloudFormation APIs. (The reason is that when creating a template in CloudFormation using the AWS console, an S3 bucket is automatically created to store the template)

One additional permission is the ability to upload a template to an S3 bucket.

So IAM users who use the CloudFormation console require additional permissions Uploading templates to an S3 Bucket:

CloudFormation : CreateUploadBucket is only available from the console and requires:

S3 : PutObject

S3 : List Bucket

S3 : GetObject

S3 : CreateBucket

So our users that are using CloudFormation from the AWS management console will need these extra permissions.

A service role is an IAM role that allows CloudFormation to make calls to resources in a stack on your behalf. So you can specify a service role that allows CloudFormation to create, update, or delete your stack resources.

So for example, if you have admin access but want to limit what can be done in CloudFormation then you would use a service role.

In the AWS Management console click **Services** and then click **IAM**
Then click **Roles** in the left margin. This is where you can create roles that pertain to CloudFormation usage.

When you create a stack in CloudFormation there is a section called Permissions where you can choose a role. If you do not choose a role, CloudFormation uses permissions based on your user credentials.

Even if you yourself have Admin Access you can add a role to your stack that dictates what you can and cannot do in CloudFormation. So when you are creating your stacks you need to make sure your users have adequate permissions to CloudFormation and also the potential resources that can be created with CloudFormation.

Use a CloudFormation Service Role to make calls to resources in a Stack on your behalf. Use a Service Role to explicitly specify actions that CloudFormation can perform which might not always be the same actions you or others can perform.

Ch 1-6 Resource Types

★★★Here is a link to the AWS Resource and Property Types Reference page
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

AWS CloudFormation is infrastructure as code

We specify our resources, such as VPC, IGW, EC2, RDS, etc. in our Template

But CloudFormation does not support every resource in AWS. New services are often not immediately supported in CloudFormation.

How do we know which resources CloudFormation supports? Go to the link here:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

What if a resource is not currently supported by CloudFormation?

The answer is CloudFormation Custom Resources.

Custom resources enable you to write custom provisioning logic in templates that CloudFormation runs anytime you create, update, or delete stacks. Example: You can include resources not available as CF resource types by using Custom Resources. That way you can manage all related resources in a single stack. (To create custom resources will require coding ability but it is possible) By the way, usually when talking about custom resources we are referring to Lambda Functions.

Go to the link:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-group.html>

By following the link above we can see how to provision an EC2 Auto Scaling group in our CloudFormation template. It gives us the JSON code as well as the YAML code that we would need to do so.

Follow this link to see how we can provision an EC2 instance in our CloudFormation template:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html>
