What is AWS CloudFormation?

CloudFormation (CFN) is AWS's **Infrastructure as Code (IaC)** service.

Instead of:

- Clicking in AWS Console
- Manually creating resources
- Forgetting what you created or how it was wired

You **describe your infrastructure declaratively** in a file, and CloudFormation:

- Creates it
- Updates it
- Deletes it
- Rolls back if something fails
- Tracks state forever

Core Concept

Template → Stack → Resources

Term	Meaning
Template	A YAML/JSON file describing AWS resources
Stack	A running instance of that template
Resources	Actual AWS services created (EC2, S3, Lambda, etc.)

Template is immutable, Stack is mutable.

Why CloudFormation Exists

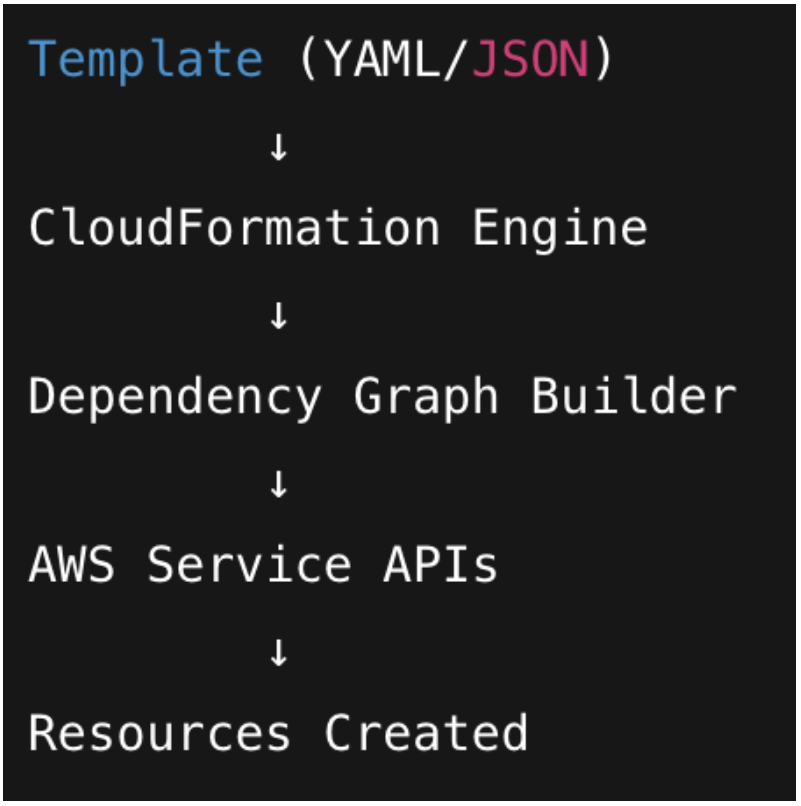
Without CloudFormation

- Manual mistakes
- No reproducibility
- No versioning
- Hard to delete everything cleanly
- No rollback

With CloudFormation

- Infrastructure is **versioned**
- Infrastructure is **repeatable**
- **Rollback on failure**
- **Idempotent** (safe to re-run)
- **Audit trail**
- **Drift detection**

CloudFormation Architecture



Important internal behavior:

- CloudFormation **calculates dependencies**
- Executes in **correct order**
- Uses **AWS service APIs under the hood**
- Maintains a **state file** (like Terraform)

Templates

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

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

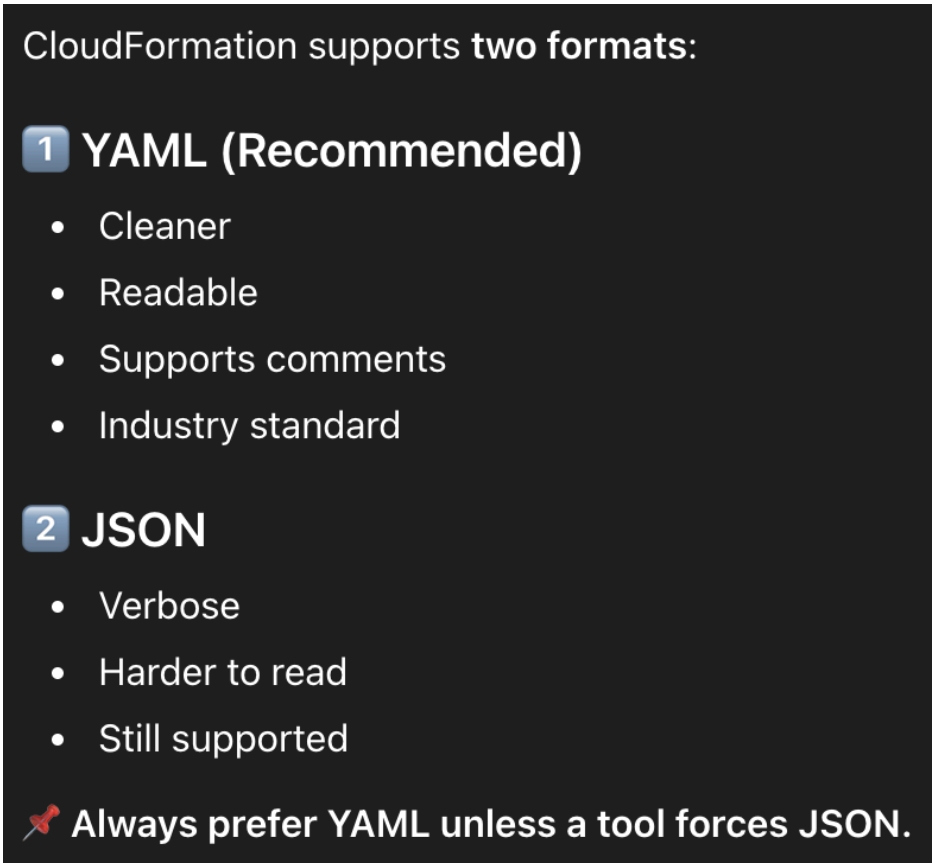
YAML

```yaml
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:
            VolumeType: io1
            Iops: 200
            DeleteOnTermination: false
            VolumeSize: 20
```

JSON

```json
{
    "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": {
                            "VolumeType": "io1",
                            "Iops": 200,
                            "DeleteOnTermination": false,
                            "VolumeSize": 20
                        }
                    }
                ]
            }
        }
    }
}
```

You can also specify multiple resources in a single template and configure these resources to work together. For example, you can modify the previous template to include an Elastic IP address (EIP) and associate it with the Amazon EC2 instance



What a Template IS

- Static file
- Versionable (Git)
- Reusable
- Environment-agnostic

Template Anatomy (VERY IMPORTANT)

A CloudFormation template has **sections**.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Description of what this template does

Parameters:
Mappings:
Conditions:
Resources:
Outputs:
Metadata:
Transform:
Rules:
```

Only **Resources** field is mandatory.

Breakdown of each fields:

AWSTemplateFormatVersion: '2010-09-09'

- Not updated often
- Safe to keep always

Description: Creates an EC2 instance with security group
- Human-readable documentation.

Parameters (Dynamic Input)

Allows user input at stack creation time.

```
Parameters:
  InstanceType:
    Type: String
    Default: t3.micro
    AllowedValues:
      - t2.micro
      - t3.micro
```

💡 **Avoid hardcoding values.**

Mappings (Static Lookup Tables)

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-0abcdef
```

Used when values depend on region or environment.

Conditions (Conditional Logic)

```
Conditions:
  IsProd: !Equals [ !Ref Env, "prod" ]
```

Used to:

- Create resources conditionally
- Apply properties conditionally

Resources (Heart of CloudFormation)

This is where actual AWS services are defined.

```
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
```

Each resource has:

- Logical ID
- Type
- Properties

Outputs (Expose Values)

```
Outputs:
  InstanceId:
    Value: !Ref MyEC2Instance
```

Used for:

- Cross-stack references
- CI/CD pipelines

- Debugging

---

*What does **declarative** mean in programming? (“WHAT ?”, not “How?”)*

- ***What, Not How**: You state the goal (e.g., "get all users in Berlin") rather than listing every loop and condition.*
- *Abstraction: Hides complex execution details, making code cleaner and more focused on the problem domain*

Intrinsic Functions (CloudFormation “Language”)

CloudFormation is declarative, but it has **functions**.

Function	Purpose
!Ref	Reference another resource
!GetAtt	Get attribute of resource
!Sub	String substitution
!Join	Join strings
!If	Conditional logic
!FindInMap	Lookup values
!ImportValue	Cross-stack reference

Example:

yaml

```yaml
SecurityGroupIds:
  - !Ref MySecurityGroup
```

What Services CloudFormation Supports

CloudFormation supports **almost all AWS services**, including:

- EC2, VPC, Subnets, Security Groups
- S3, DynamoDB, RDS
- Lambda, API Gateway
- IAM (roles, policies)
- CloudWatch
- SNS, SQS
- ECS, EKS
- Step Functions
- Amazon Connect
- AWS Bedrock (partial)

📌 If AWS adds a service → CloudFormation usually supports it shortly after.

CloudFormation Stack Lifecycle

Stack States

- `CREATE_IN_PROGRESS`
- `CREATE_COMPLETE`
- `UPDATE_IN_PROGRESS`
- `UPDATE_COMPLETE`
- `ROLLBACK_IN_PROGRESS`
- `DELETE_IN_PROGRESS`

Rollback

If **any resource fails**:

- Entire stack rolls back
- Partial resources deleted automatically

Stacks

When you use 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 CloudFormation template. Suppose you created a template that includes an Auto Scaling group, ELB 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 CloudFormation provisions all those resources for you.

What a Stack IS

- An **AWS-managed object**
- Created by applying a template
- Has **state**
- Has **lifecycle**
- Owns resources

Template vs Stack

Template

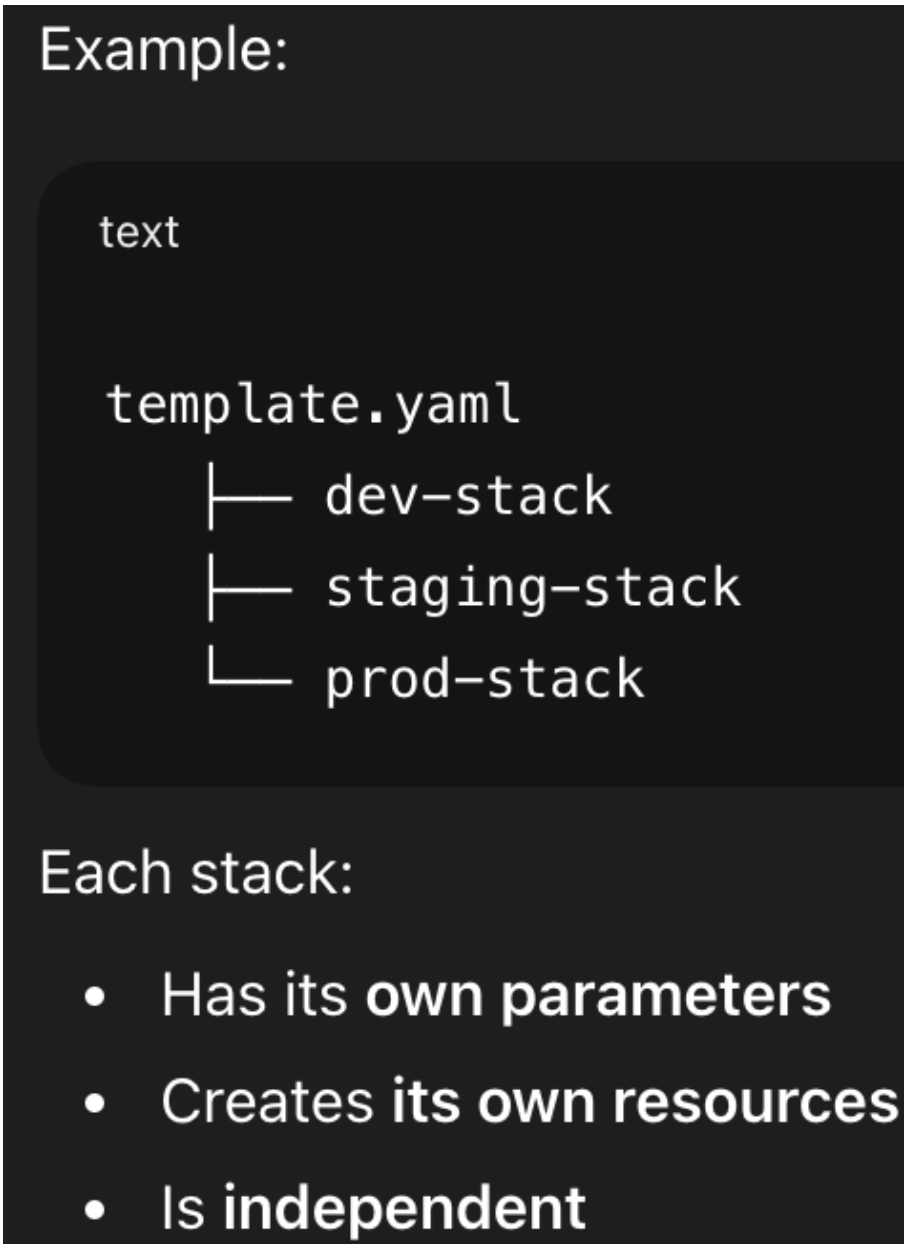
A **blueprint / recipe / desired-state description**
Just a **file** (YAML or JSON)
Describes *what should exist*

Stack

A **live, running instance** of that template
Exists **inside AWS**
Owns **real AWS resources**

One Template → Many Stacks (VERY IMPORTANT)

You can reuse the same template to create **multiple stacks**.



Practical example

Using **the same CloudFormation template**, we will create:

Stack Name	Difference
`dev-stack`	Smaller EC2, dev tag, dev S3 bucket
`prod-stack`	Bigger EC2, prod tag, prod S3 bucket

Both stacks:

- Run **independently**
- Do **not affect each other**
- Use **same template**
- Have **different parameters**

1. CloudFormation Template (Single File)

File name: `example.yaml`

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Simple EC2 + S3 example with parameters

Parameters:
  EnvironmentName:
    Type: String
    AllowedValues:
      - dev
      - prod
    Description: Environment name (dev or prod)

  InstanceType:
    Type: String
    Description: EC2 instance type

  KeyName:
    Type: AWS::EC2::KeyPair::KeyName
    Description: Existing EC2 KeyPair for SSH

Resources:
  AppBucket:
```

```
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub my-${EnvironmentName}-bucket-${AWS::AccountId}

  AppSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow SSH access
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0

  AppEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref InstanceType
      KeyName: !Ref KeyName
      ImageId: ami-0a0f1259dd1c90938   # Amazon Linux (example)
      SecurityGroups:
        - !Ref AppSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName}-ec2-instance
        - Key: Environment
          Value: !Ref EnvironmentName

Outputs:
  EC2InstanceId:
    Description: EC2 Instance ID
    Value: !Ref AppEC2Instance

  S3BucketName:
    Description: S3 Bucket Name
    Value: !Ref AppBucket
```

2. Prerequisites

Before running stacks:

1. AWS CLI configured

```
aws configure
```

2. EC2 KeyPair exists

Check in EC2 → Key Pairs

`My-keypair    (e.g. Dax-kp)`

3. Create FIRST Stack (Dev)

Can be done in two ways:

1. CLI Based

```
aws cloudformation create-stack \
  --stack-name dev-stack \
  --template-body file://example.yaml \
  --parameters \
    ParameterKey=EnvironmentName,ParameterValue=dev \
    ParameterKey=InstanceType,ParameterValue=t2.micro \
    ParameterKey=KeyName,ParameterValue=my-keypair
```

```
daxrajsinh@Daxrajsinhs-MacBook-Air Cloudformation-practice_and_examples % aws cloudformation create-stack \
  --stack-name dev-stack \
  --template-body file://example.yaml \
  --parameters \
    ParameterKey=EnvironmentName,ParameterValue=dev \
    ParameterKey=InstanceType,ParameterValue=t2.micro \
    ParameterKey=KeyName,ParameterValue=Dax-kp
```

Note that we need to provide values in **Key - Value** based (You can see in above pic)

We'll get something like:

```
{
  "StackId": ".../stack/dev-stack/...",
  "OperationId": "..."
}
```

✅ Means: **CloudFormation accepted your request**

❌ Does **NOT** mean: EC2 is running successfully

CloudFormation works like this:

```sql
CREATE_STACK request accepted
        ↓
Stack state = CREATE_IN_PROGRESS
        ↓
Resources are attempted one by one
```

If **any resource fails**, EC2 will **never appear**.

ALWAYS Check Stack Events (Most Important Debug Step)

Run this **immediately**:

***aws cloudformation describe-stack-events ***
--stack-name dev-stack

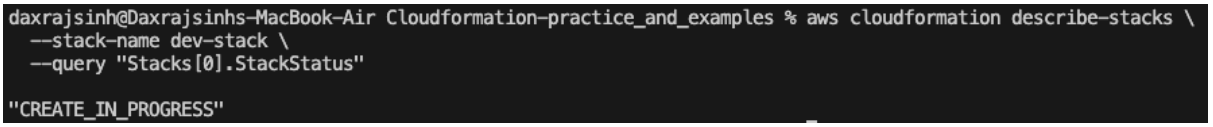
We can something like (Cropping for security purpose)

Confirm Stack Status (Quick Check)

```
aws cloudformation describe-stacks \
  --stack-name dev-stack \
  --query "Stacks[0].StackStatus"
```

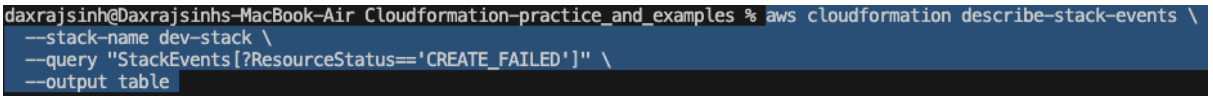
If you see:

- ROLLBACK_COMPLETE → resources failed and were deleted
- CREATE_IN_PROGRESS → still working
- CREATE_FAILED → failed early



To check the REASON why Stack failed:

```
aws cloudformation describe-stack-events \
  --stack-name dev-stack \
  --query "StackEvents[?ResourceStatus=='CREATE_FAILED']" \
  --output table
```



DELETING A STACK (If broken)

```
aws cloudformation delete-stack --stack-name dev-stack
```

Summary:

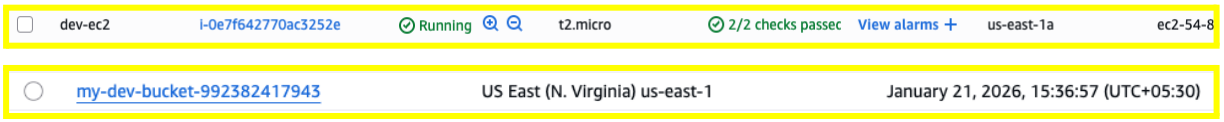
To deploy:

```bash
aws cloudformation delete-stack --stack-name dev-stack
aws cloudformation wait stack-delete-complete --stack-name dev-stack

aws cloudformation create-stack \
  --stack-name dev-stack \
  --template-body file://template.yaml \
  --parameters \
      ParameterKey=EnvironmentName,ParameterValue=dev \
      ParameterKey=InstanceType,ParameterValue=t2.micro \
      ParameterKey=KeyName,ParameterValue=your-key-name-here
```

You can see Cloudformation is created!

```
daxrajsinh@Daxrajsinhs-MacBook-Air Cloudformation-practice_and_examples % aws cloudformation describe-stacks \
  --stack-name dev-stack \
  --query "Stacks[0].StackStatus"

"CREATE_COMPLETE"
```

Resources created (EC2 and S3)



NOTE *(Good practice):*

Deleting Stack will delete all the resources initiated by it (including VPC, subnets, ec2, s3, etc)

2. UI Based (on AWS Console)

We just need to select Create Stack

Same steps just uploading a .***yaml*** file there.