AWS CloudFormation

A Templating Language that **defines AWS resources** to be provisioned.
Automating the creation of resources via code.

***Infrastructure as Code**

CloudFormation - Introduction

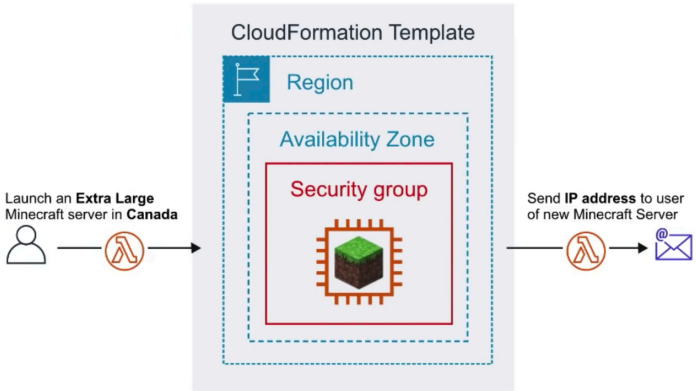
What is Infrastructure As Code? (IaC)

the process of managing and provisioning computer data centers (eg, AWS) through machine-readable definition files (eg, YAML, JSON files) rather than physical hardware configuration or interactive configuration tools. (stop doing manual configuration!)

Use Case

People pay a monthly subscription and we run a Minecraft server. They choose **where** they want and **what size** of server they want to run.

We can use their **inputs** and use an AWS Lambda to create a new CloudFormation stack. We have a lambda send them the email of their new Minecraft Server IP address and details.



CloudFormation – Template Formats

```json
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Description":"Launch an EC2 Instance running apache and expose default pag
  "Parameters":{
    "InstanceType":{
      "Description":"WebServer EC2 instance type",
      "Type":"String",
      "Default":"t2.micro",
      "AllowedValues":[
        "t2.nano",
        "t2.micro"
      ]
    }
  },
  "Resources":{
    "WebServer":{
      "Type":"AWS::EC2::Instance",
      "Properties":{
        "Tags":[
          {
            "Key":"Name",
            "Value":"Apache Default WebServer"
          }
        ],
        "InstanceType":{
          "Ref":"InstanceType"
        },
        "ImageId":"ami-0b898040803850657",
        "SecurityGroupIds":[
          {
            "Fn::GetAtt":[
              "SecurityGroup",
              "GroupId"
            ]
          }
        ],
        "UserData":{
          "Fn::Base64":{
            "Fn::Sub":"#!/usr/bin/env bash\nsu ec2-user\nsudo yum install htt
          }
        }
      }
    },
    "SecurityGroup":{
      "Type":"AWS::EC2::SecurityGroup",
      "Properties":{
        "GroupDescription":"Enable internet users to access.",
        "SecurityGroupIngress":[
          {
            "IpProtocol":"tcp",
            "FromPort":80,
            "ToPort":80,
            "CidrIp":"0.0.0.0/0"
          }
        ]
      }
    }
```

CloudFormation can be written in ✌️ two different formats:

👈 JSON

YAML 👉

```yaml
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Launch an EC2 Instance running apache and expose
  default page to the internet. Harcoded to work
  only with  US-EAST-1 (N. Virginia)
Parameters:
  InstanceType:
    Description: WebServer EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.nano
      - t2.micro
Resources:
  WebServer:
    Type: 'AWS::EC2::Instance'
    Properties:
      Tags:
        -
          Key: Name
          Value: Apache Default WebServer
      InstanceType: !Ref InstanceType
      # You shouldn't hardcode, just shown here for example
      ImageId: 'ami-0b898040803850657'
      SecurityGroupIds:
        - !GetAtt SecurityGroup.GroupId
      UserData:
        'Fn::Base64':
          !Sub |
          #!/usr/bin/env bash
          su ec2-user
          sudo yum install httpd -y
          sudo service httpd start
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable internet users to access.
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
Outputs:
  PublicIp:
    Value: !GetAtt WebServer.PublicIp

```

CloudFormation – Template Anatomy

```yaml
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Launch an EC2 Instance running apache and expose
  default page to the internet. Harcoded to work
  only with  US-EAST-1 (N. Virginia)
Parameters:
  InstanceType:
    Description: WebServer EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.nano
      - t2.micro
Resources:
  WebServer:
    Type: 'AWS::EC2::Instance'
    Properties:
      Tags:
        -
          Key: Name
          Value: Apache Default WebServer
      InstanceType: !Ref InstanceType
      # You shouldn't hardcode, just shown here for example
      ImageId: 'ami-0b898040803850657'
      SecurityGroupIds:
        - !GetAtt SecurityGroup.GroupId
      UserData:
        'Fn::Base64':
          !Sub |
          #!/usr/bin/env bash
          su ec2-user
          sudo yum install httpd -y
          sudo service httpd start
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable internet users to access.
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
Outputs:
  PublicIp:
    Value: !GetAtt WebServer.PublicIp

```

Template Sections

MetaData	Additional information about the template
Description	A description of what this template is suppose to do
Parameters	Values to pass to your template at runtime
Mappings	A lookup table. Maps keys to values so you change your values to something else
Conditions	Whether resources are created or properties are assigned
Transform	Applies marcos (like applying a mod which change the anatomy to be custom)
Resources*	A resource you want to create eg. IAM Role, EC2 Instance, Lambda, RDS
Outputs	Values that returned eg. an ip-address of new server created.

CloudFormation Templates **requires** you to **at least list one resource**.

AWS Quick Starts

AWS Quick Starts are a collection of **pre-built CloudFormation templates**.

- Analytics
- Blockchain
- Business productivity
- Communications
- Contact center
- Containers & microservices
- Data lakes
- Databases
- DevOps
- Healthcare
- Infrastructure
- IoT
- Life sciences
- Machine learning & AI
- Media services
- Migration
- Networking & remote access
- SaaS
- Security, identity, compliance
- Serverless
- Storage
- Websites & web apps
- IBM
- Microsoft
- SAP

IOT | SERVERLESS

Quick Start

ONICA

AWS IoT Camera Connector

Built by Onica and AWS

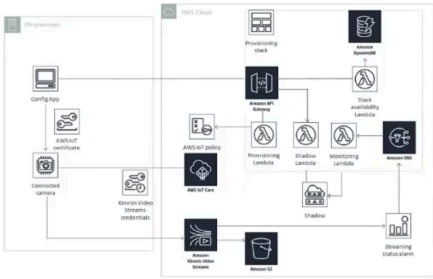
Builds a serverless architecture to connect and manage cameras through AWS IoT Core, and to stream camera

Time to deploy
5 min

What you'll build | **How to deploy** | **Cost and licenses**

Use this Quick Start to set up the following **serverless** architecture on AWS:

- An AWS IoT policy to associate with connected cameras.
- An AWS Identity and Access Management (IAM) role for connected cameras to stream to Kinesis Video Streams.
- An Amazon DynamoDB table to store provisioning keys. Once provisioning is complete, you should remove the keys.
- AWS Lambda functions to create a provisioning key and a role alias, verify the stack, and provision cameras.
- Amazon API Gateway to expose provisioning endpoints through HTTPS.
- Amazon CloudWatch alarms to expose camera streaming status through an Amazon Simple Notification Service (Amazon SNS) topic, and update the associated camera's IoT thing shadow.
- A separate Config App installable for provisioning cameras on the local network to stream to your AWS account.



Switch to full-screen view

CloudFormation *CheatSheet*

- When being asked to **automate** the provisioning of resources *think* CloudFormation
- When Infrastructure as Code (IaC) is mentioned *think* CloudFormation
- CloudFormation can be written in either JSON or YAML
- When CloudFormation encounters an error it will rollback with **ROLLBACK_IN_PROGRESS**
- CloudFormation templates larger than 51,200 bytes (0.05 MB) are too large to upload directly, and must be imported into CloudFormation via an S3 bucket.
- **NestedStacks** helps you break up your CloudFormation template into smaller reusable templates that can be composed into larger templates
- **At least one resource** under resources: must be defined for a CloudFormation template **to be valid**
- **MetaData** extra information about your template
- **Description** a description of what the template is suppose to do
- **Parameters** is how you get user inputs into templates
- **Transforms** Applies marcos (like applying a mod which change the anatomy to be custom)
- **Outputs** are values you can use to import into other stacks
- **Mappings** maps keys to values, just like a lookup table
- **Resources** defines the resources you want to provision, **at least one resource is required**
- **Conditions** are whether resources are created or properties are assigned