



# Murex Cloud Dev Architecture In-Depth Analysis

# Agenda

- 01 introduction To CloudFormation
- 02 brief Introduction to necessary Services in AWS
- 03 mx-base-stack Analysis
- 04 mx-env-stack Analysis
- 05 rds-dbinfra Analysis
- 06 lambda Analysis
- 07 env-stack Analysis
- 08 necessary library functions
- 09 thank you

# welcome!



## Mission

We're going to analyze the aws murex dev architecture, along with all necessary aws services

It's a challenging and interesting journey, the sessions will in-depth

So follow along with me and try to practice.

# Introduction To CloudFormation

→ AWS CloudFormation: Managing your infrastructure as code

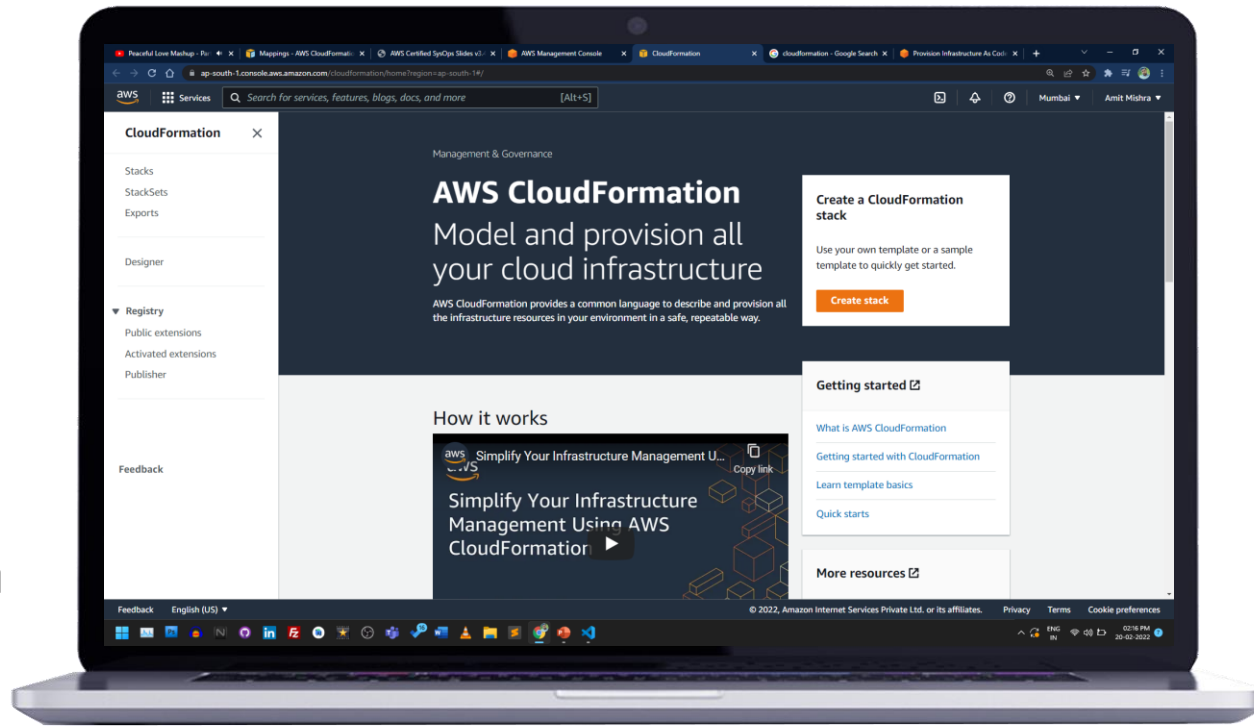
# Infrastructure as Code

- Currently, we have been doing a lot of manual work
- All this manual work will be very tough to reproduce:
  - In another region
  - In another AWS account
- Within the same region if everything was deleted
- Wouldn't it be great, if all our infrastructure was... code?
- That code would be deployed and create / update / delete our infrastructure

# What is CloudFormation



- CloudFormation is a declarative way of outlining your AWS
- Most of AWS services are supported.
- For example, within a CloudFormation template, you say:
  - I want a security group
  - I want two EC2 machines using this security group
  - I want two Elastic IPs for these EC2 machines
- Then CloudFormation creates those for you, in the right order, with the exact configuration that you specify



# Benefits of AWS CloudFormation

- Infrastructure as code
  - No resources are manually created, which is excellent for control
  - The code can be version controlled for example using git
  - Changes to the infrastructure are reviewed through code
- Cost
  - Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
  - You can estimate the costs of your resources using the CloudFormation template
  - Savings strategy: In Dev, you could automation deletion of stack at 5 PM and recreated at 8 AM next day, safely

# Benefits of AWS CloudFormation

- Productivity
  - Ability to destroy and re-create an infrastructure on the cloud on the fly
  - Automated generation of Diagram for your templates!
  - Declarative programming (no need to figure out ordering)
- Separation of concern: create many stacks for many apps, and many layers.
  - VPC stacks
  - Network stacks
  - App stacks
- Don't re-invent the wheel
  - Leverage existing templates on the web!
  - Leverage the documentation



# How CloudFormation Works

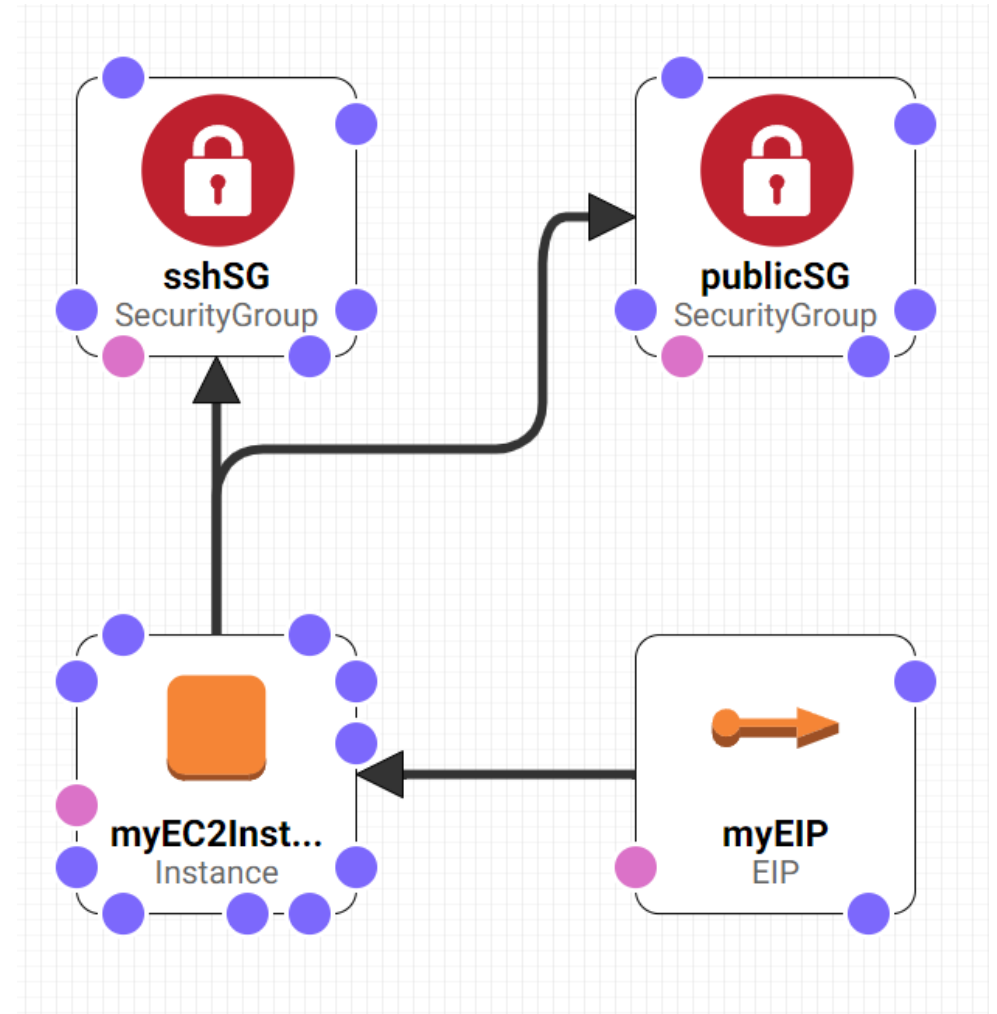
- Templates **must** be uploaded in **S3** and then referenced in CloudFormation
- To update a template, we can't edit previous ones. We have to reupload a new version of the template to AWS
- Stacks are identified by a name
- Deleting a stack deletes every single artifact that was created by CloudFormation.

# Deploying CloudFormation templates

- Manual way:
  - Editing templates in the CloudFormation Designer
  - Using the console to input parameters, etc
- Automated way:
  - Editing templates in a YAML file
  - Using the AWS CLI (Command Line Interface) to deploy the templates
  - Recommended way when you fully want to automate your flow

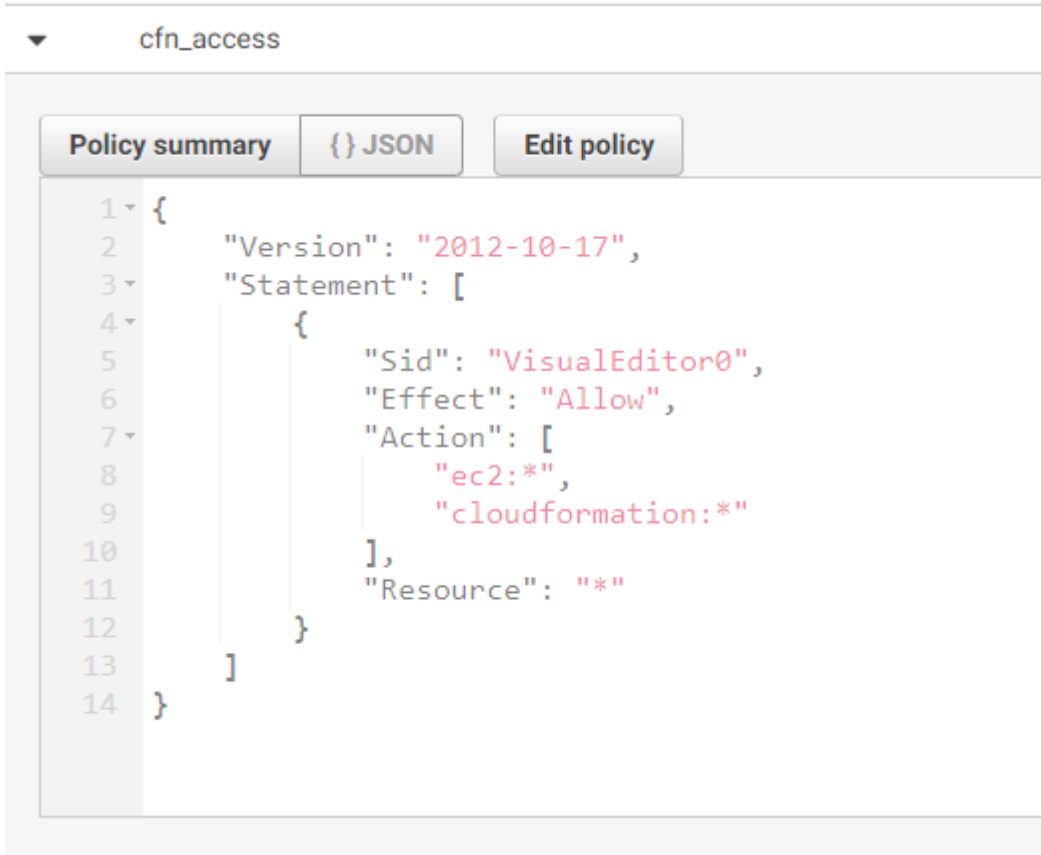
# Introductory Example

- We're going to create a simple EC2 instance.
- Then we're going to create to add an Elastic IP to it
- And we're going to add two security groups to it
- We'll see how in no-time, we are able to get started with CloudFormation



# Question Time???

There is an iam user named mishra12 having following iam policy attached to this user profile



```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "VisualEditor0",
6       "Effect": "Allow",
7       "Action": [
8         "ec2:*",
9         "cloudformation:*"
10      ],
11       "Resource": "*"
12     }
13   ]
14 }
```

The question is that whether he will be able to launch a t2.micro ec2 instance using cloudformation?

Explain your answer with valid reason.

# YAML Crash Course

```
1 ---
2 Parameters:
3   SecurityGroupDescription:
4     Description: Security Group Description
5     Type: String
6
7 Resources:
8   MyInstance:
9     Type: AWS::EC2::Instance
10    Properties:
11      AvailabilityZone: us-east-1a
12      ImageId: ami-009d6802948d06e52
13      InstanceType: t2.micro
14      SecurityGroups:
15        - !Ref SSHSecurityGroup
16        - !Ref ServerSecurityGroup
17
18 # an elastic IP for our instance
19 MyEIP:
20   Type: AWS::EC2::EIP
21   Properties:
22     InstanceId: !Ref MyInstance
23
```

- YAML and JSON are the languages you can use for CloudFormation.
- JSON is horrible for CF
- YAML is great in so many ways
- Let's learn a bit about it!
- Yaml Syntax:

```
<key>: <value>
```

# YAML Crash Course

- Comments

```
# comments Syntax example in YAML file  
or  
#### comments example
```

- Scalars

```
integer: 25  
hex: 0x12d4 #evaluates to 4820  
octal: 023332 #evaluates to 9946  
float: 25.0  
exponent: 12.3015e+05 #evaluates to 1230150.0  
boolean: Yes  
string: "25"  
infinity: .inf # evaluates to infinity  
neginf: -.Inf #evaluates to negative infinity  
not: .NAN #Not a Number
```

# YAML Crash Course

- Strings

```
str: Hello World
data: |
  These
  Newlines
  Are broken up
data: >
  This text is
  wrapped and is a
  single paragraph
```

- Array

```
shopping:
- milk
- eggs
- juice
```

# YAML Crash Course

- Dictionaries

```
Employees:  
- mishra12:  
  name: Amit Mishra  
  job: Support Executive  
  team: MES
```



# CloudFormation Building Blocks

Templates components (IMPORTANT):

1. **Resources:** your AWS resources declared in the template (MANDATORY)
2. **Parameters:** the dynamic inputs for your template
3. **Mappings:** the static variables for your template
4. **Outputs:** References to what has been created
5. **Conditionals:** List of conditions to perform resource creation
6. **Metadata**

Templates helpers:

- References
- Functions

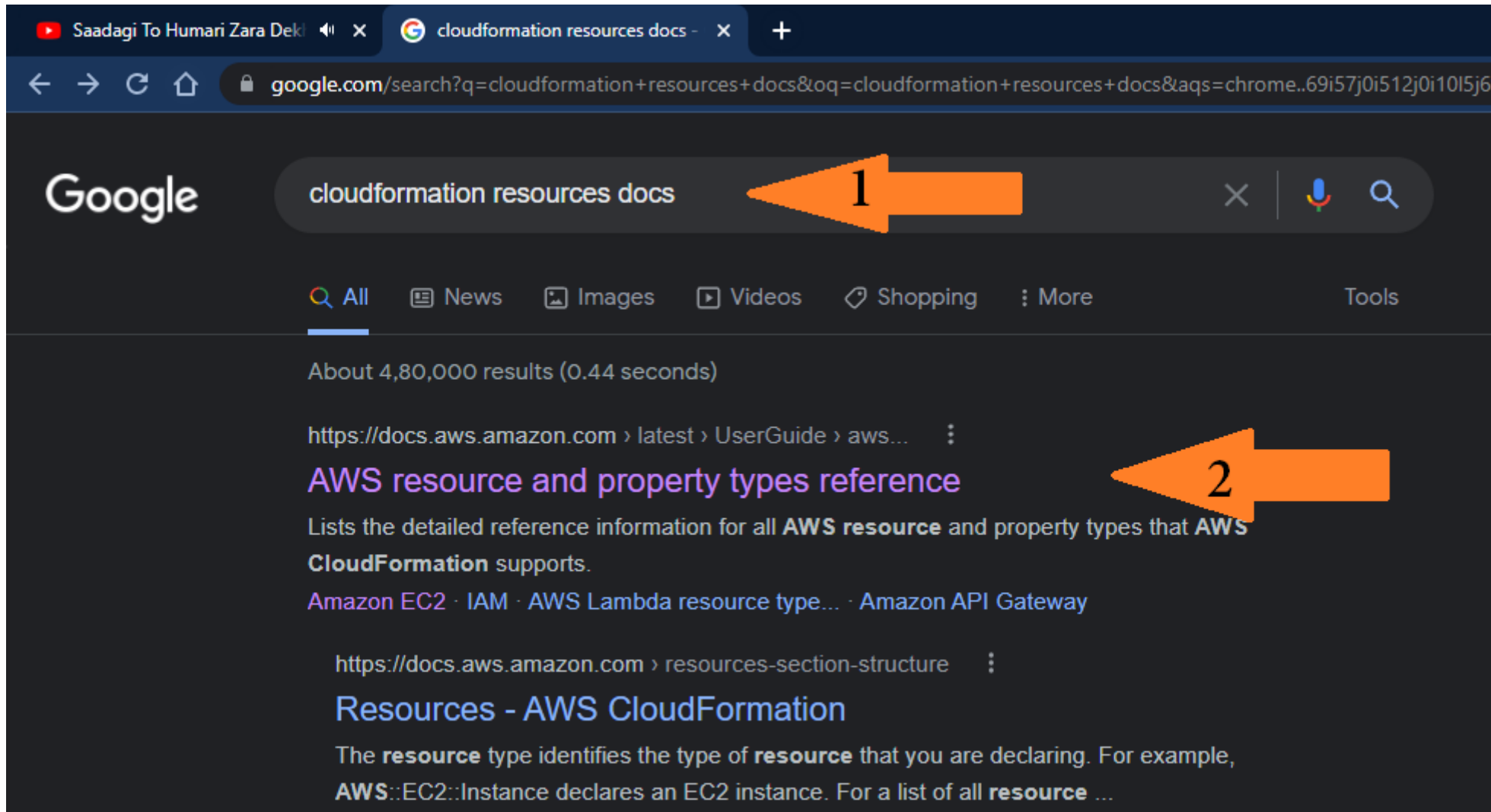
# What are resources?

- Resources are the core of your CloudFormation template (MANDATORY)
- Represent AWS Services that can be created and configured
- Resources are declared and can be connected to each other
- AWS figures out creation, updates and deletes of resources for us
- There are over 224 types of resources (!)
- Resource types identifiers are of the form:

`AWS::aws-product-name::data-type-name`

- `AWS::EC2::Instance`
- `AWS::EC2::SecurityGroup`
- `AWS::EC2::EIP`

# How do I find resources documentation?



# FAQ for resources

- Can I create a dynamic number of resources?

No, you can't. Everything in the CloudFormation template has to be declared.

You can't perform code generation there

- Is every AWS Service supported?

Almost. Only a select few niches are not there yet

You can work around that using AWS Lambda Custom Resources

# What are parameters

- Parameters are a way to provide inputs to AWS CloudFormation template
- They're important to know about if:
  - Some inputs can not be determined ahead of time
- Parameters are extremely powerful, controlled, and can prevent errors from happening in your templates thanks to types.
- When should you use a parameter?
  - Ask yourself this:
    - Is this CloudFormation resource configuration likely to change in the future?
    - If so, make it a parameter.
  - You won't have to re-upload a template to change its content

# Parameters Settings

Parameters can be controlled by all these settings:

- Type:
  - String
  - Number
  - Comma Delimited List
  - List<Type>
  - AWS Parameter (to help catch invalid values – match against existing values in the AWS Account)
- Description
- Min/MaxLength
- Min/MaxValue
- Defaults
- AllowedValues (array)
- AllowedPattern (regex)
- NoEcho (Boolean)

```
Parameters:
  InstanceTypeParameter:
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - m1.small
      - m1.large
    Description: Choose EC2 Instance Type.
```

# How to Reference a Parameter

- The Fn::Ref function can be leveraged to reference parameters
- Parameters can be used anywhere in a template.
- The shorthand for this in YAML is !Ref
- The function can also reference other elements within the template

```
---
Parameters:
  myEC2InstanceType:
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - m1.small
      - m1.large
    Description: Choose EC2 Instance Type.

Resources:
  myEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref myEC2InstanceType
      ImageId: "ami-0c6615d1e95c98aca"
      AvailabilityZone: "ap-south-1a"
```

# Concept: Pseudo Parameters

- AWS offers us pseudo parameters in any CloudFormation template.
- These can be used at any time and are enabled by default

Pseudo Parameter	Result
AWS::AccountId	1234567890
AWS::NotificationARNs	[arn:aws:sns:us-east1:123456789012:MyTopic]
AWS::NoValue	Does not return a value
AWS::Region	us-east-2
AWS::StackId	arn:aws:cloudformation:us-east1:123456789012:stack/MyStack/1c2fa62 0-982a-11e3-aff7-50e2416294e0
AWS::StackName	MyStack



# What are mappings?

- Mappings are fixed variables within your CloudFormation Template.
- They're very handy to differentiate between different environments (dev vs prod), regions (AWS regions), AMI types, etc
- All the values are hardcoded within the template
- Example:

```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
```

```
RegionMap:
  us-east-1:
    HVM64: ami-0ff8a91507f77f867
    HVMG2: ami-0a584ac55a7631c0c
  us-west-1:
    HVM64: ami-0bdb828fd58c52235
    HVMG2: ami-066ee5fd4a9ef77f1
  eu-west-1:
    HVM64: ami-047bb4163c506cd98
    HVMG2: ami-0a7c483d527806435
```

# Mappings vs Parameters: when to use

- Mappings are great when you know in advance all the values that can be taken and that they can be deduced from variables such as
  - Region
  - Availability Zone
  - AWS Account
  - Environment (dev vs prod)
  - Etc...
- They allow safer control over the template.
- Use parameters when the values are really user specific
- i.e Mapping are static and Parameters are Dynamic variables

# Accessing Mapping Values

- We use `Fn::FindInMap` to return a named value from a specific key
- `!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]`

```
Mappings:
  RegionMap:
    us-east-1:
      HVM64: ami-0ff8a91507f77f867
      HVMG2: ami-0a584ac55a7631c0c
    us-west-1:
      HVM64: ami-0bdb828fd58c52235
      HVMG2: ami-066ee5fd4a9ef77f1
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]
      InstanceType: t2.micro
```

# What are outputs?

- The Outputs section declares optional outputs values that we can import into other stacks (if you export them first)!
- You can also view the outputs in the AWS Console or in using the AWS CLI
- They're very useful for example if you define a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
- It's the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack
- You can't delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack

# Outputs Example

- Creating a SSH Security Group as part of one template
- We create an output that references that security group

```
---
Resources:
  SGPing:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: SG to test ping
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
Outputs:
  StcakSSHSG:
    Description: The Instance ID
    Value: !Ref SGPing
    Export:
      Name: myInstanceSSHSG
```

# Cross Stack Reference : Importing Outputs

- We then create a second template that leverages that security group
- For this, we use the `Fn::ImportValue` function
- You can't delete the underlying stack until all the references are deleted too.

```
---
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId: 'ami-0c6615d1e95c98aca'
      SecurityGroups:
        - !ImportValue myInstanceSSHSG
```

**more slides coming soon...**



**thanks!**

A large orange arrow that starts under the 't' and ends under the 's', curving upwards to form a smile.



# You can find me on



/mishracodes



## Amit Kumar Mishra

Murex Technical Analyst

and

System Engineer at Tata Consultancy Services