CloudFormation

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 Infrastructure, for any resources (most of them 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
- I want an S3 bucket
- I want a load balancer (ELB) in front of these machines
- Then CloudFormation creates those for you, in the right order, with the exact configuration that you specify

Benefits of AWS CloudFormation (1/2)

- 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 templates at 5 PM and recreated at 8 AM, safely
- 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 and orchestration)
- Separation of concern: create many stacks for many apps, and many layers. Ex:
  - VPC stacks
  - Network stacks
  - App stacks

**How CloudFormation Works**

- Templates have to 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

**CloudFormation Building Blocks**

**Templates components (one course section for each):**

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:**

1. References
2. Functions

What are resources?

- Resources are the core of your CloudFormation template (MANDATORY)
- They represent the different AWS Components that will be created and configured
- Resources are declared and can reference 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

http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html

FAQ for resources

- Can I create a dynamic amount of resources?

ØNo, you can't. Everything in the CloudFormation template has to be declared. You can't perform code generation there

- Is every AWS Service supported?

ØAlmost. Only a select few niches are not there yet

ØYou can work around that using AWS Lambda Custom Resources

What are parameters?

- Parameters are a way to provide inputs to your AWS CloudFormation template
- They're important to know about if:
    - You want to reuse your templates across the company
    - 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:
  SecurityGroupDescription:
    Description: Security Group Description
    (Simple parameter)
    Type: String
```

Parameters Settings
Parameters can be controlled by all these settings:

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

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

```yaml
DbSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyVPC
```

Concept: Pseudo Parameters

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

| Reference Value | Example Return Value |
|---|---|
| AWS::AccountId | 1234567890 |
| AWS::NotificationARNs | [arn:aws:sns:us-east-1:123456789012:MyTopic] |
| AWS::NoValue | Does not return a value. |
| AWS::Region | us-east-2 |
| AWS::StackId | arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-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:
    "32": "ami-6411e20d"
    "64": "ami-7a11e213"
  us-west-1:
    "32": "ami-c9c7978c"
    "64": "ami-cfc7978a"
  eu-west-1:
    "32": "ami-37c2f643"
    "64": "ami-31c2f645"
```

**When would you use mappings vs parameters ?**

- 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)
- They allow safer control over the template.
- Use parameters when the values are really user specific

Fn::FindInMap

Accessing Mapping Values

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

```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      "32": "ami-6411e20d"
      "64": "ami-7a11e213"
    us-west-1:
      "32": "ami-c9c7978c"
      "64": "ami-cfc7978a"
    eu-west-1:
      "32": "ami-37c2f643"
      "64": "ami-31c2f645"
    ap-southeast-1:
      "32": "ami-66f28c34"
      "64": "ami-60f28c32"
    ap-northeast-1:
      "32": "ami-9c03a89d"
      "64": "ami-a003a8a1"
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]
      InstanceType: m1.small
```

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

```yaml
Outputs:
  StackSSHSecurityGroup:
    Description: The SSH Security Group for our Company
    Value: !Ref MyCompanyWideSSHSecurityGroup
    Export:
      Name: SSHSecurityGroup
```

Cross Stack Reference

- 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:
  MySecureInstance:
    Type: AWS::EC2::Instance
    Properties:
      AvailabilityZone: us-east-1a
      ImageId: ami-a4c7edb2
      InstanceType: t2.micro
      SecurityGroups:
        - !ImportValue SSHSecurityGroup
```

What are conditions used for?

- Conditions are used to control the creation of resources or outputs based on a condition.
- Conditions can be whatever you want them to be, but common ones are:
  - Environment (dev / test / prod)
  - AWS Region
  - Any parameter value
- Each condition can reference another condition, parameter value or mapping

How to define a condition?

- The logical ID is for you to choose. It's how you name condition
- The intrinsic function (logical) can be any of the following:
- Fn::And
- Fn::Equals
- Fn::If
- Fn::Not
- Fn::Or

```
Conditions:
  CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

Conditions can be applied to resources / outputs / etc

CloudFormation

Must Know Intrinsic Functions

- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc…)

Fn::GetAtt

- Attributes are attached to any resources you create
- To know the attributes of your resources, the best place to look at is the documentation.
- For example: the AZ of an EC2 machine!

```yaml
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: ami-1234567
      InstanceType: t2.micro
```

```yaml
  NewVolume:
    Type: "AWS::EC2::Volume"
    Condition: CreateProdResources
    Properties:
      Size: 100
      AvailabilityZone:
        !GetAtt EC2Instance.AvailabilityZone
```

Fn::Join

- Join values with a delimiter

```
!Join [ delimiter, [ comma-delimited list of values ] ]
```

- This creates "a:b:c"

```
!Join [ ":", [ a, b, c ] ]
```

Function Fn::Sub

- Fn::Sub, or !Sub as a shorthand, is used to substitute variables from a text. It's a very handy function that will allow you to fully customize your templates.
- For example, you can combine Fn::Sub with References or AWS Pseudo variables!
- String must contain ${VariableName} and will substitute them

```
!Sub
  - String
  - { Var1Name: Var1Value, Var2Name: Var2Value }
```

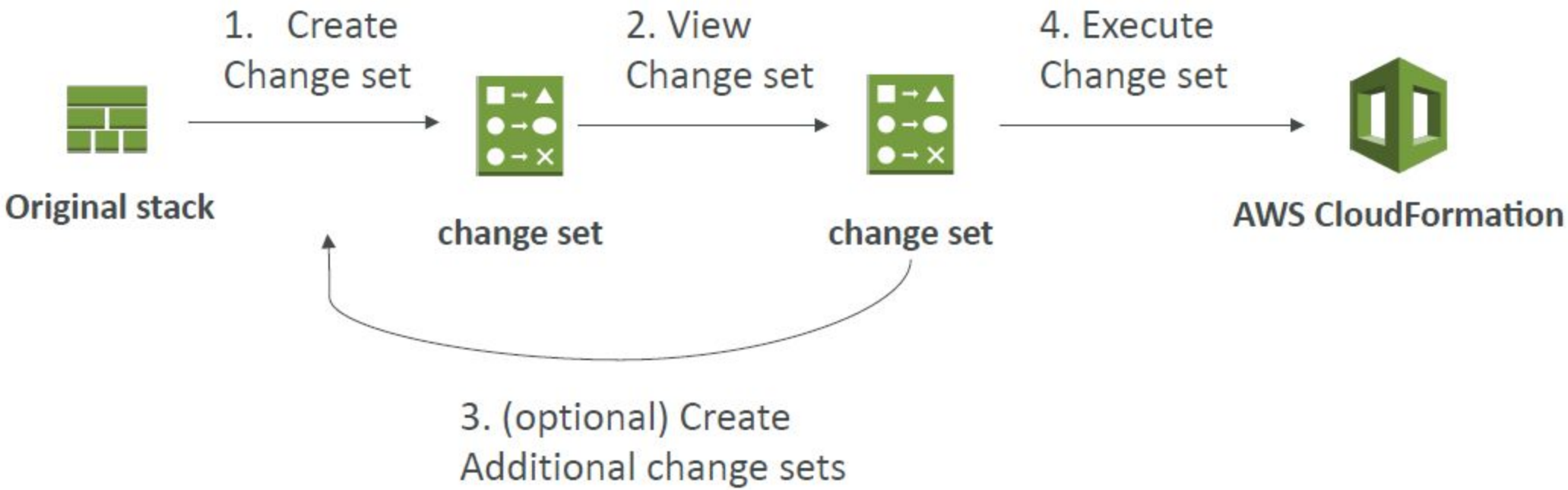
```
!Sub String
```

CloudFormation Rollbacks

- Stack Creation Fails:
  - Default: everything rolls back (gets deleted). We can look at the log
  - Option to disable rollback and troubleshoot what happened
- Stack Update Fails:
  - The stack automatically rolls back to the previous known working state
  - Ability to see in the log what happened and error messages

ChangeSets

- When you update a stack, you need to know what changes before it happens for greater confidence
- ChangeSets won't say if the update will be successful

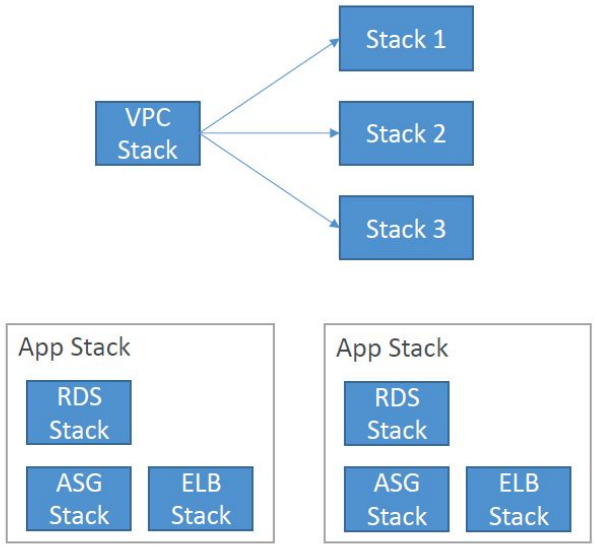


Nested stacks

- Nested stacks are stacks as part of other stacks
- They allow you to isolate repeated patterns / common components in separate stacks and call them from other stacks
- Example:
    - Load Balancer configuration that is re-used
    - Security Group that is re-used
- Nested stacks are considered best practice
- To update a nested stack, always update the parent (root stack)

CloudFormation – Cross vs Nested Stacks

- Cross Stacks
  - Helpful when stacks have different lifecycles
  - Use Outputs Export and Fn::ImportValue
  - When you need to pass export values to many stacks (VPC Id, etc…)
- Nested Stacks
  - Helpful when components must be re-used
  - Ex: re-use how to properly configure an Application Load Balancer
  - The nested stack only is important to the higher level stack (it's not shared)



CloudFormation - StackSets

- Create, update, or delete stacks across multiple accounts and regions with a single operation
- Administrator account to create StackSets
- Trusted accounts to create, update,delete stack instances from StackSets
- When you update a stack set, all associated stack instances are updated throughout all accounts and regions.