
aws
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 resource (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 staged 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

Benefits of AWS CloudFormation (2/2)

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
- Don’t re-invent the wheel
  - Leverage existing templates on the web!
  - Leverage the documentation

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

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

AWS Resources Supported By CloudFormation

This section contains reference information for all AWS resource and property types that are supported by AWS CloudFormation.

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

EC2 Example Provisioning

- Example here (for an EC2 instance): Relevant docs

- http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/awsproperties-ec2-instance.html
- http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/awsproperties-ec2-security-group.html
- http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/awsproperties-ec2-eip.html

Understanding the CloudFormation template options

Let's learn about the parameters that are common to any CloudFormation template

1. Tags
2. Permissions
3. Notification Options
4. Timeouts
5. Rollback on Failure
6. Stack Policy

CloudFormation Building Blocks

Templates components (one course section for each):

1. Resources: your AWS resources declared in the template
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 (learning as we encounter them):

1. References
2. Functions

CLOUDFORMATION PARAMETERS

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.

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
Theory & Hands-On

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)

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

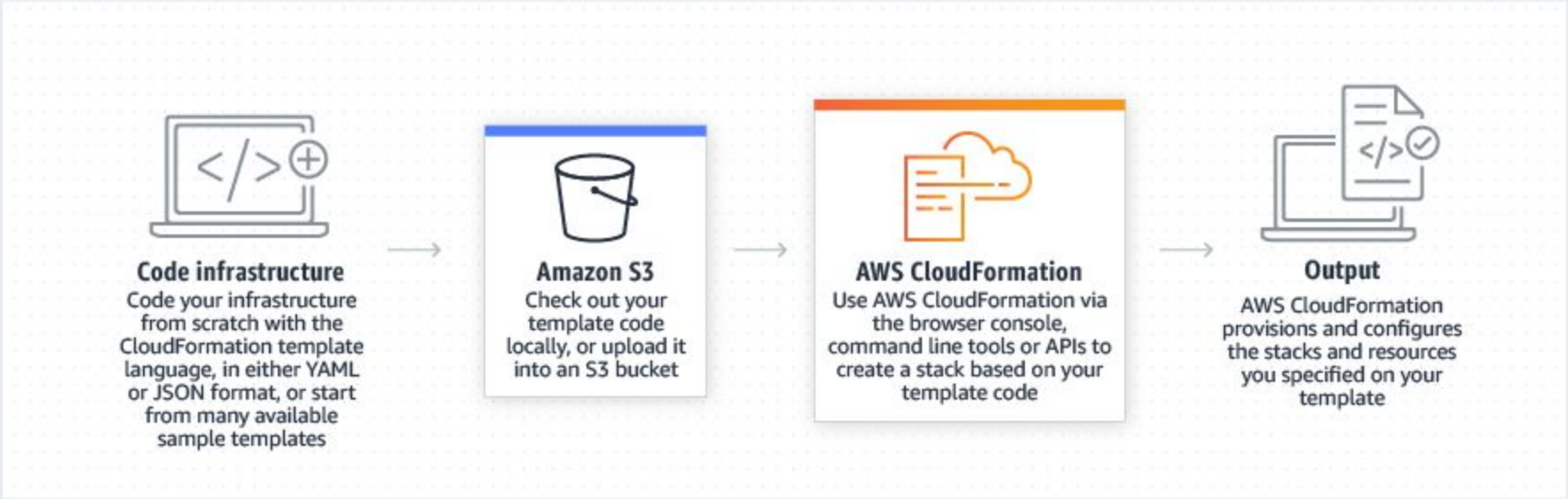
CLOUDFORMATION RESOURCES

What are resources?

- Resources are the core of your CloudFormation template.
- 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:

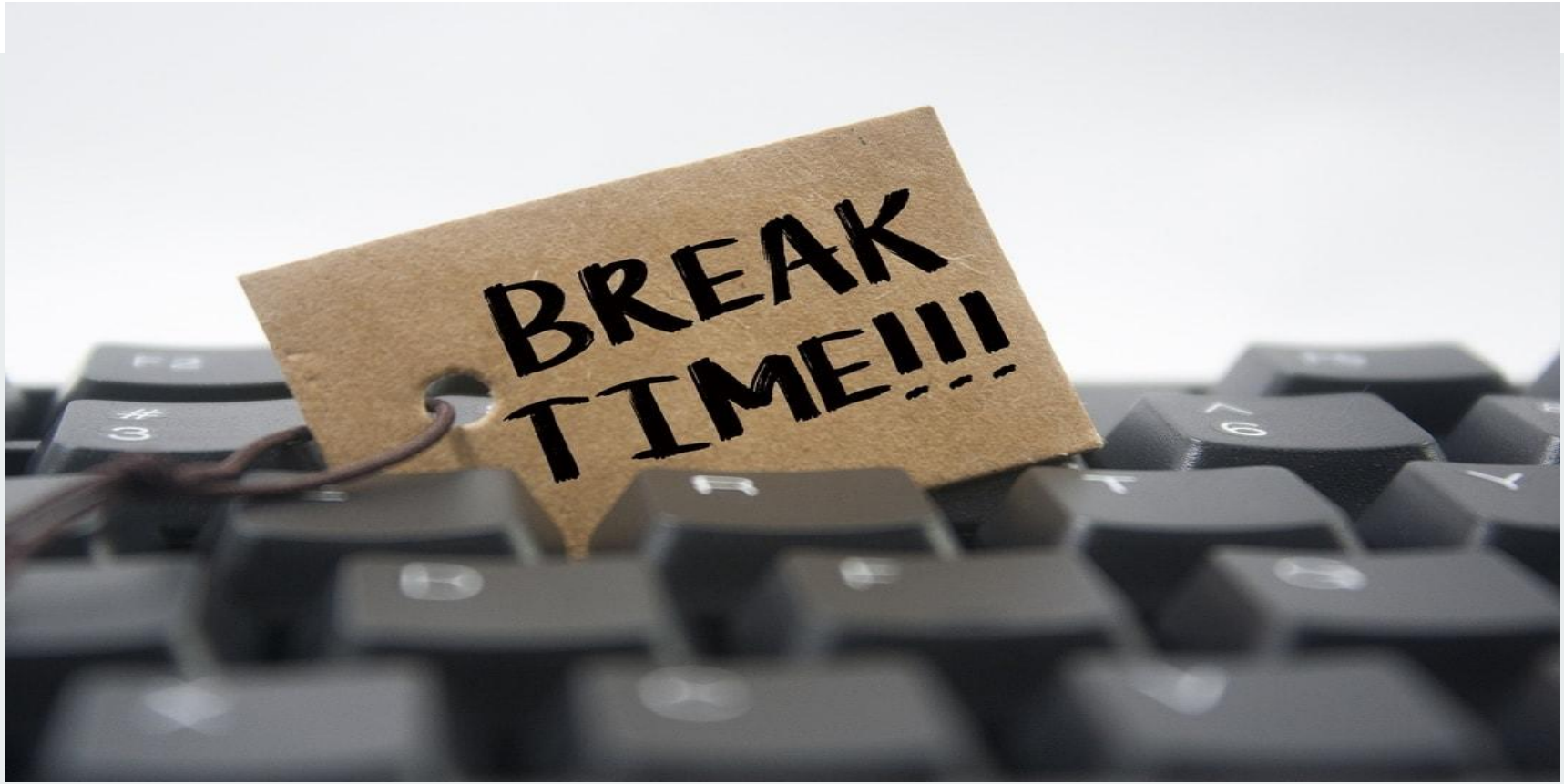
```
AWS::aws-product-name::data-type-name
```


Code infrastructure
Code your infrastructure from scratch with the CloudFormation template language, in either YAML or JSON format, or start from many available sample templates
Amazon S3
Check out your template code locally, or upload it into an S3 bucket
AWS CloudFormation
Use AWS CloudFormation via the browser console, command line tools or APIs to create a stack based on your template code
Output
AWS CloudFormation provisions and configures the stacks and resources you specified on your template

PART - 1

HANDS-ON - 1

- Provision Resources
- Use Parameters



Time: 1 Hour

PART 2

CLOUDFORMATION MAPPING

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"
```

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
```

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)
  - Etc…
- They allow safer control over the template.

- Use parameters when the values are really user specific

Hands-On

- Let's use this mapping and see how we can combine it with a parameter in order to deduce values

- We'll change the instance type of our AWS Instance based on the environment we're in
- We'll change the AMI of our AWS Instance based on the region we're in

PART - 2

HANDS-ON - 1

- Mapping (Fn::FindInMap)

CLOUDFORMATION METADATA

What is Metadata?

- You can use the *optional* metadata section to include arbitrary YAML that provide details about the template or resource.
- For example

```
Metadata:
  Instances:
    Description: "Information about the instances"
  Databases:
    Description: "Information about the databases"
```

Special metadata keys

- We actually have already encountered the Metadata section when dealing with the designer.
- There are 3 metadata keys that have special meaning:
  - AWS::CloudFormation::Designer : Describes how the resources are laid out in your template. This is automatically added by the AWS Designer.
  - AWS::CloudFormation::Interface : Define grouping and ordering of input parameters when they are displayed in the AWS Console.
  - AWS::CloudFormation::Init : Define configuration tasks for cfn-init. It's the most powerful usage of the metadata. See next section for more details

PART - 2

Hands-On - 2

- AWS::CLoudFormation::Designer

CLOUDFORMATION OUTPUTS

What are outputs?

- The Outputs section declares *optional* outputs values that we can import into other stacks!
- 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

Outputs Hands-On

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

```
Outputs:
  Logical ID:
    Description: Information about the value
    Value: Value to return
    Export:
      Name: Value to export
```

Cross Stack Reference Hands-On

- 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.

PART - 2

HANDS-ON - 3

- Outputs
- Cross Stack Reference

CLOUDFORMATION CONDITIONS

What are conditionals used for?

- Conditionals 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?

```
Conditions:
  Logical ID:
    Intrinsic function
```

- 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`

Conditional Hands On

- Let's analyze a CloudFormation template that optionally creates a volume and mount point only if "prod" is specified as a parameter

- It utilizes parameters, mappings, conditionals, outputs, so it's a great all round example!

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.

Syntax for the full function name:

```
Fn::GetAtt: [ logicalNameOfResource, attributeName ]
```

Syntax for the short form:

```
!GetAtt logicalNameOfResource.attributeName
```

PART - 2

HANDS-ON - 4

- CloudFormation Conditions
- Get::Att

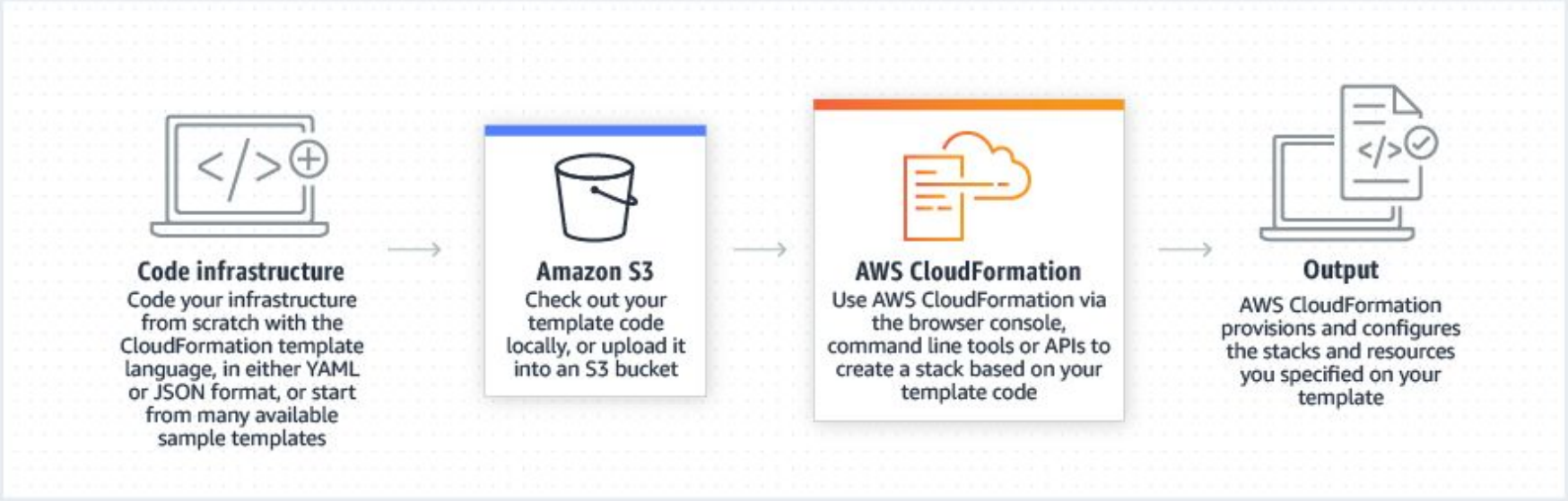
CLOUD FORMATION

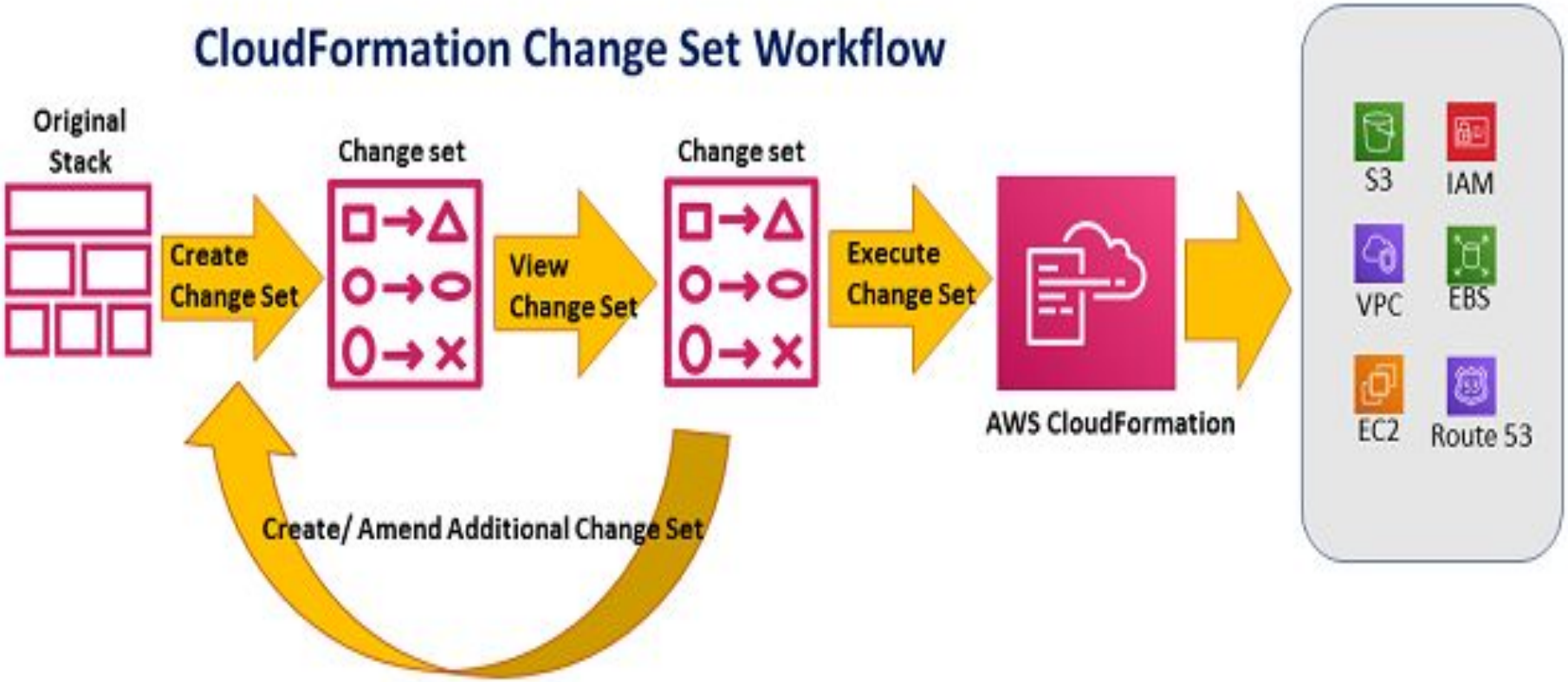
CHANGESETS

Change Sets

- Preview the changes, verify that they are in line with their expectations, and proceed with the update.
- You create a change set by submitting changes against the stack you want to update. CloudFormation compares the stack to the new template and/or parameter values and produces a change set that you can review and then choose to apply

THIS IS WHAT WE KNOW




CloudFormation Change Set Workflow
Original Stack
Create Change Set
Change set
View Change Set
Change set
Execute Change Set
AWS CloudFormation
S3
IAM
VPC
EBS
EC2
Route 53
Create/ Amend Additional Change Set

PART - 2

HANDS-ON - 5

- CloudFormation ChangeSets

CLOUD FORMATION

- NESTED STACKS
- DRIFTS

Nested Stacks Overview

- 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)
- Nested stacks can have nested stacks themselves!

Nested Stacks Update

- To update a nested stack…

- Ensure the updated nested stacks are uploaded onto S3 first
- Then re-upload your **root** stack

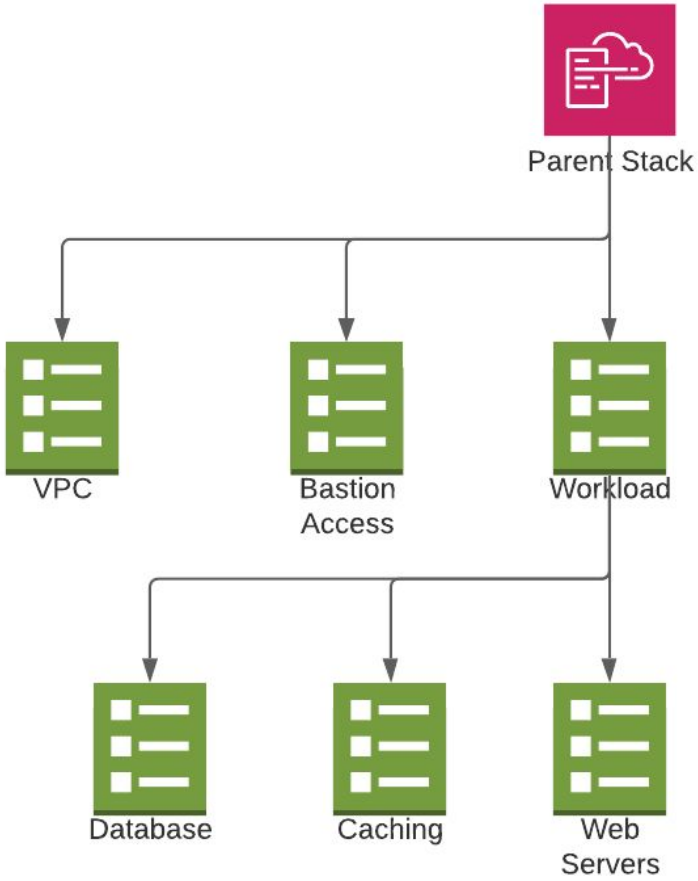
Stack Nesting

We have our application broken into pieces because we don't want to have a monolithic infrastructure, but we need a way to put the together

Nested Stacks!!

Parent template orchestrates the launching of all it's individual components.

This helps you get around stack limits like length...

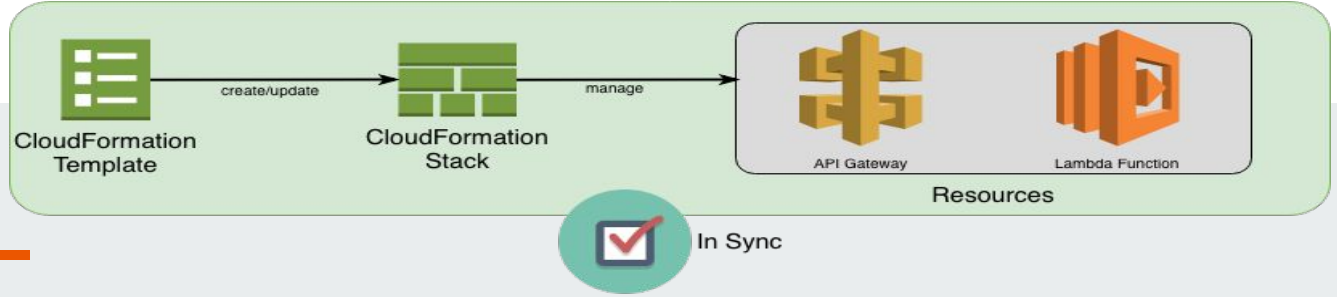


What's Drift?

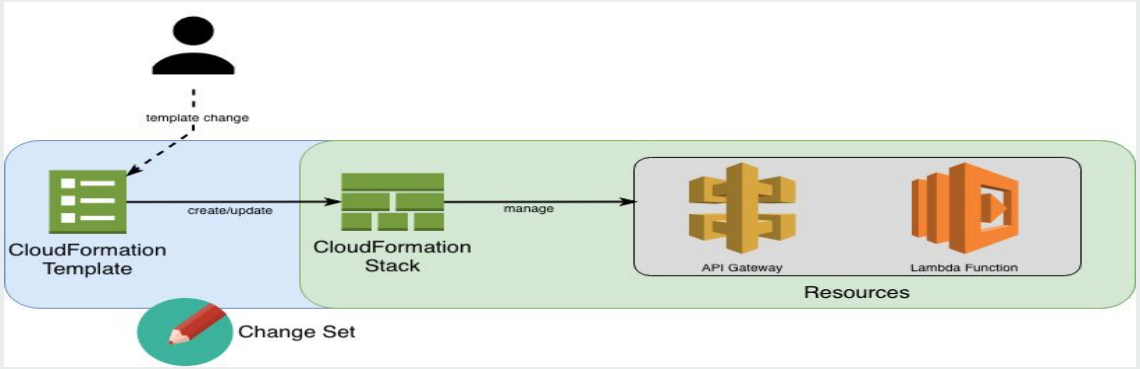
- CloudFormation allows you to create infrastructure
- But it doesn't protect you against manual configuration changes
- How do we know if our resources have **drifted**?

- We can use CloudFormation drift!

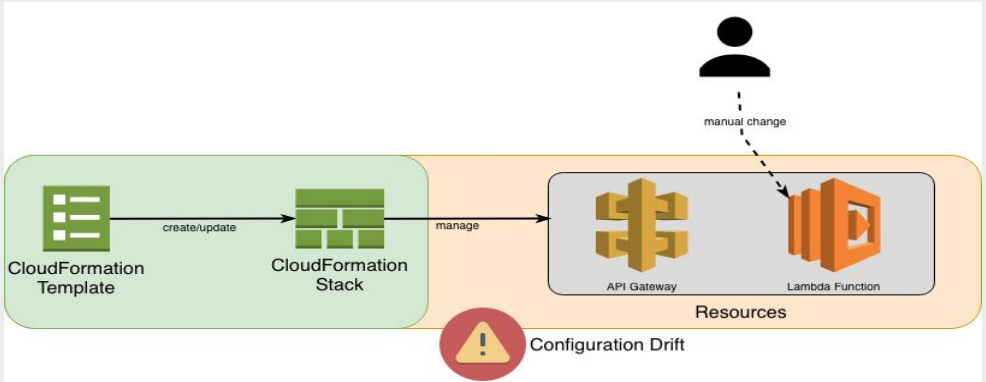
1


create/update
manage
CloudFormation Template
CloudFormation Stack
API Gateway
Lambda Function
Resources
In Sync

2


template change
create/update
manage
CloudFormation Template
CloudFormation Stack
API Gateway
Lambda Function
Resources
Change Set

3


manual change
create/update
manage
CloudFormation Template
CloudFormation Stack
API Gateway
Lambda Function
Resources
Configuration Drift

PART - 2

HANDS-ON - 6

- Nested Stacks
- CloudFormation Drift

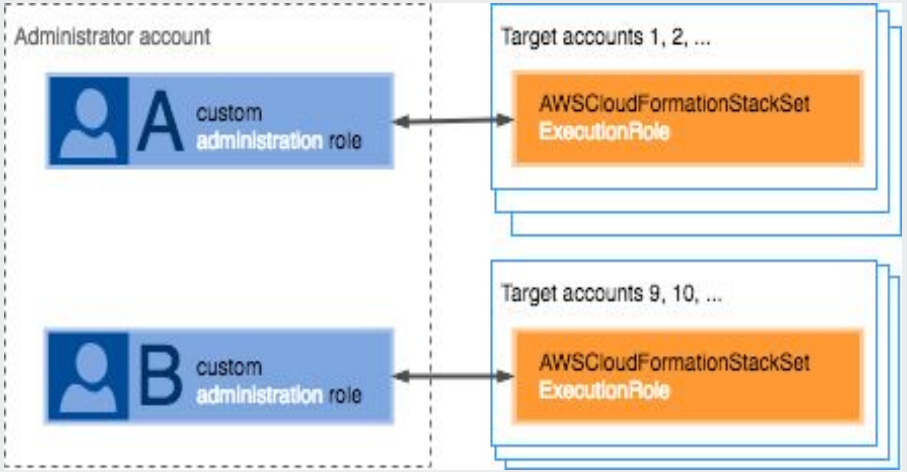
CLOUD FORMATION

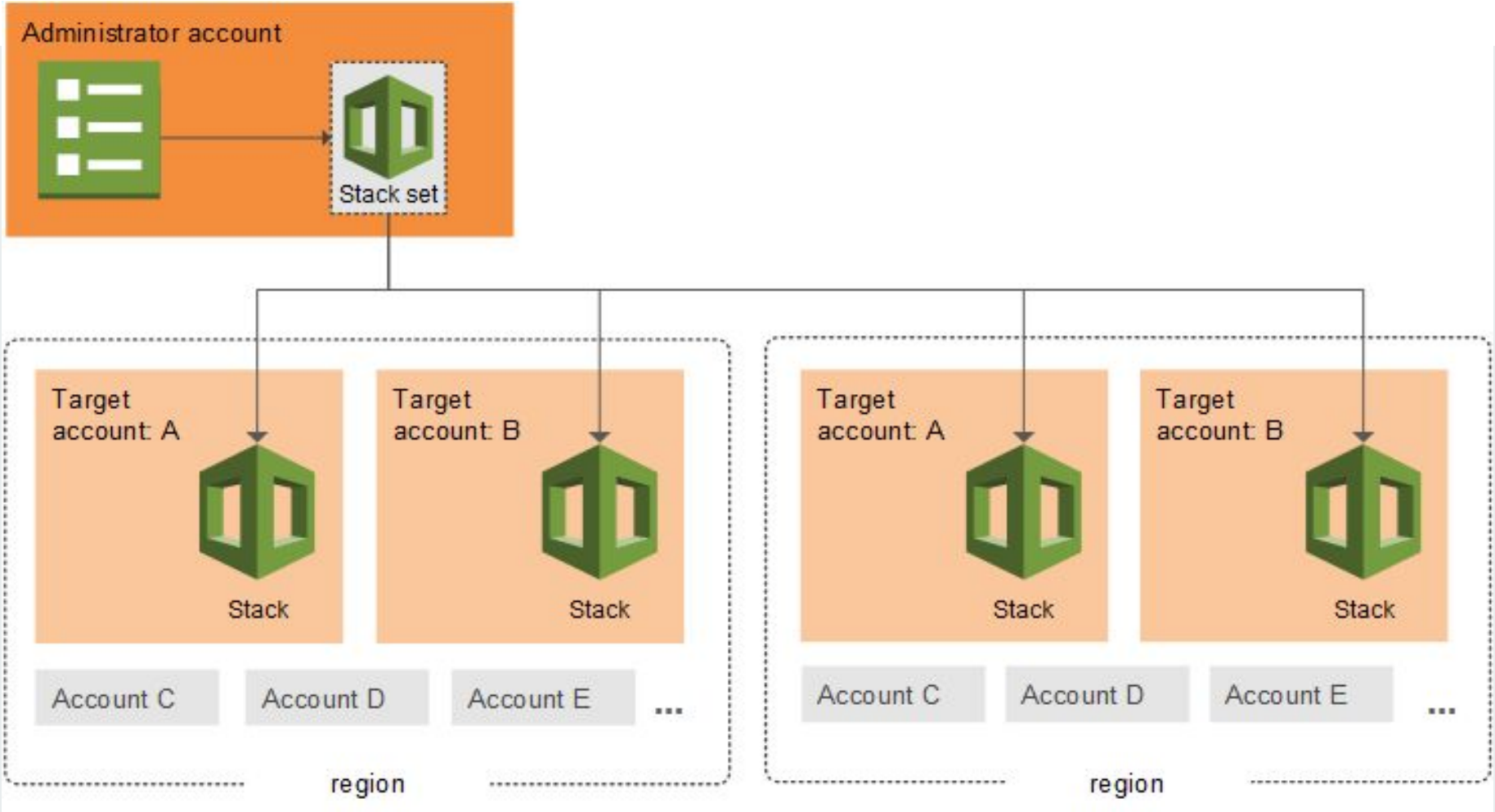
STACKSETS

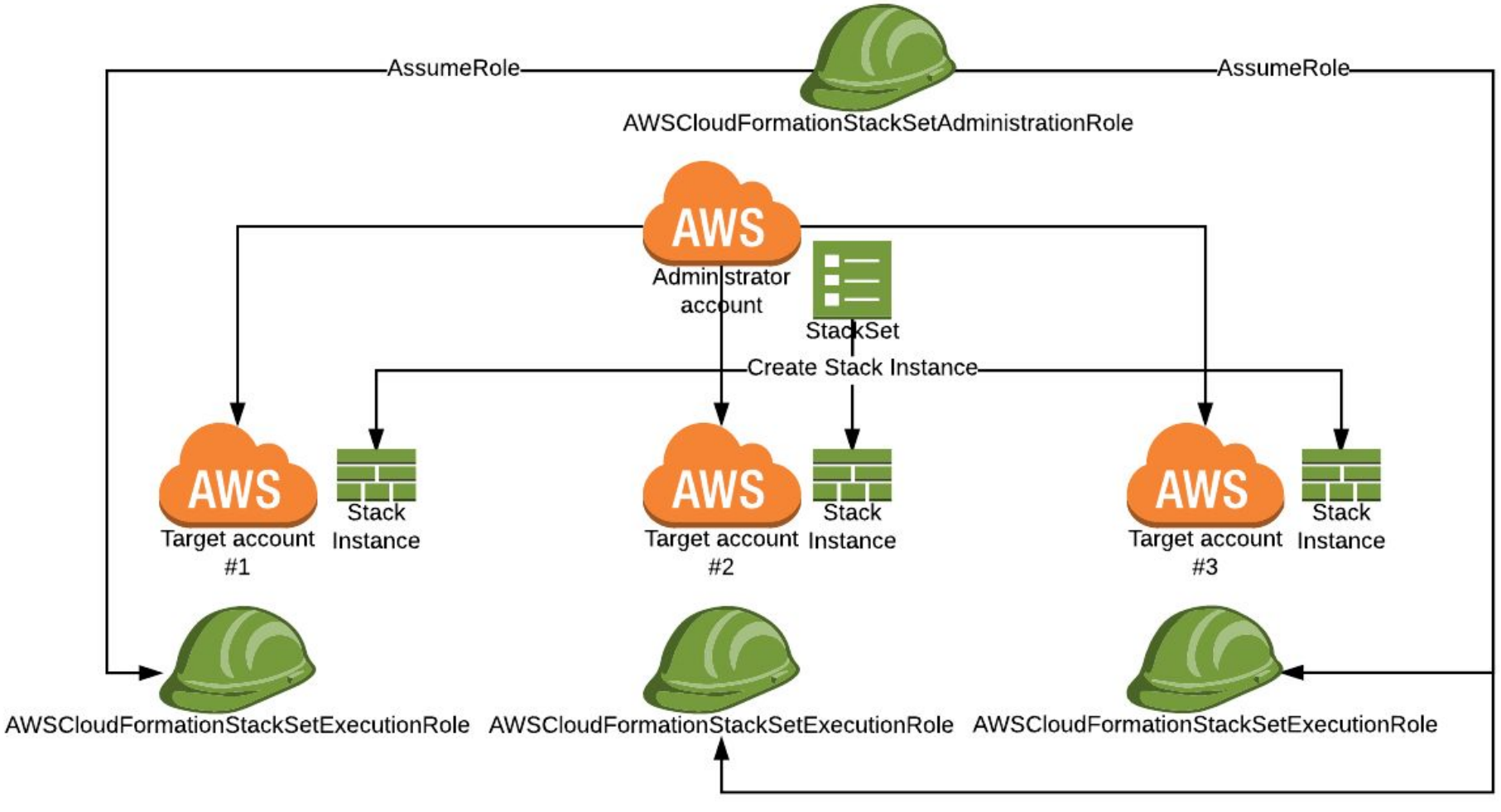
What is CloudFormation StackSets ?

AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation.

Using an Administrator account, you define and manage an AWS CloudFormation template, and use the template as the basis for provisioning stacks into selected target accounts across specified regions




Administrator account
Stack set
Target account: A
Stack
Target account: B
Stack
Account C
Account D
Account E
...
region
Target account: A
Stack
Target account: B
Stack
Account C
Account D
Account E
...
region


AssumeRole
AWSCloudFormationStackSetAdministrationRole
AssumeRole
AWS
Administrator account
StackSet
Create Stack Instance
AWS
Target account #1
Stack Instance
AWS
Target account #2
Stack Instance
AWS
Target account #3
Stack Instance
AWSCloudFormationStackSetExecutionRole
AWSCloudFormationStackSetExecutionRole
AWSCloudFormationStackSetExecutionRole

PART - 3

HANDS-ON - 2, AWS CLI

- CloudFormation Stacksets
- Dynamic Reference - Accessing Secretes from CloudFormation
- Dynamic Reference - Accessing SSM Parameter Store from CloudFormation

ADVANCE CLOUD FORMATION

- USER DATA
- HELPER SCRIPTS
- CFT + AWS CLI

CFN-Init and EC2-user data. What!?

- Many of the CloudFormation templates will be about provisioning computing resources in your AWS Cloud.
- These resources can be either
  - EC2 instances
  - AutoScaling Groups.
- Usually, you want the instances to be self configured so that they can perform the job they are supposed to perform.
- You can fully automate your EC2 fleet state with CloudFormation Init.
- First, let's deal with a simple manual example.

The problems with EC2 user data

- Now, what if we want to have a very large instance configuration?
- What if we want to evolve the state of the EC2 instance without terminating it and creating a new one?
- How do we make EC2 user-data more readable?
- How do we know or signal that our EC2 user-data script actually completed successfully?

Enter CloudFormation helper scripts

- We have 4 python scripts, that come directly on Amazon Linux AMI, or can be installed using yum on non Amazon Linux
  - cfn-init: Used to retrieve and interpret the resource metadata, installing packages, creating files and starting services.
  - cfn-signal: A simple wrapper to signal an AWS CloudFormation CreationPolicy or WaitCondition, enabling you to synchronize other resources in the stack with the application being ready.
  - cfn-get-metadata: A wrapper script making it easy to retrieve either all metadata defined for a resource or path to a specific key or subtree of the resource metadata.
  - cfn-hup: A daemon to check for updates to metadata and execute custom hooks when the changes are detected.
- Usual flow: cfn-init, then cfn-signal, then optionally cfn-hup

***Helper Scripts are* NOT** used extensively in the Industry TODAY. Most Especially in Enterprise Environments and the reason is because;

- Spinning up infrastructure and installing softwares all have seperate lifecycles
- It makes CloudFormation hard to read
- There are Better alternatives available
  - Ansible
  - System Manager (SSM)
  - Puppet
  - Build a Golden AMI

Using the AWS CLI to deploy CloudFormation

- We can use the AWS CLI to create, update or delete CloudFormation templates.
- It is very convenient when you start automating your deployments
- It's also great for managing parameters in a file

- Let's get hands-on!

PART - 3

HANDS-ON - 1

- CloudFormation User Data
- Deploy CloudFormation Templates using the CLI

ADVANCE CLOUD FORMATION

INFRASTRUCTURE PROTECTION

Protect Your CloudFormation Templates

- Deletion Policy (DeletionPolicy: “Retain”)
- IAM
- Termination Protection
- Stack Policy

Create Infrastructure Patterns for Your Organization Using the AWS CDK

Amazon Web Services

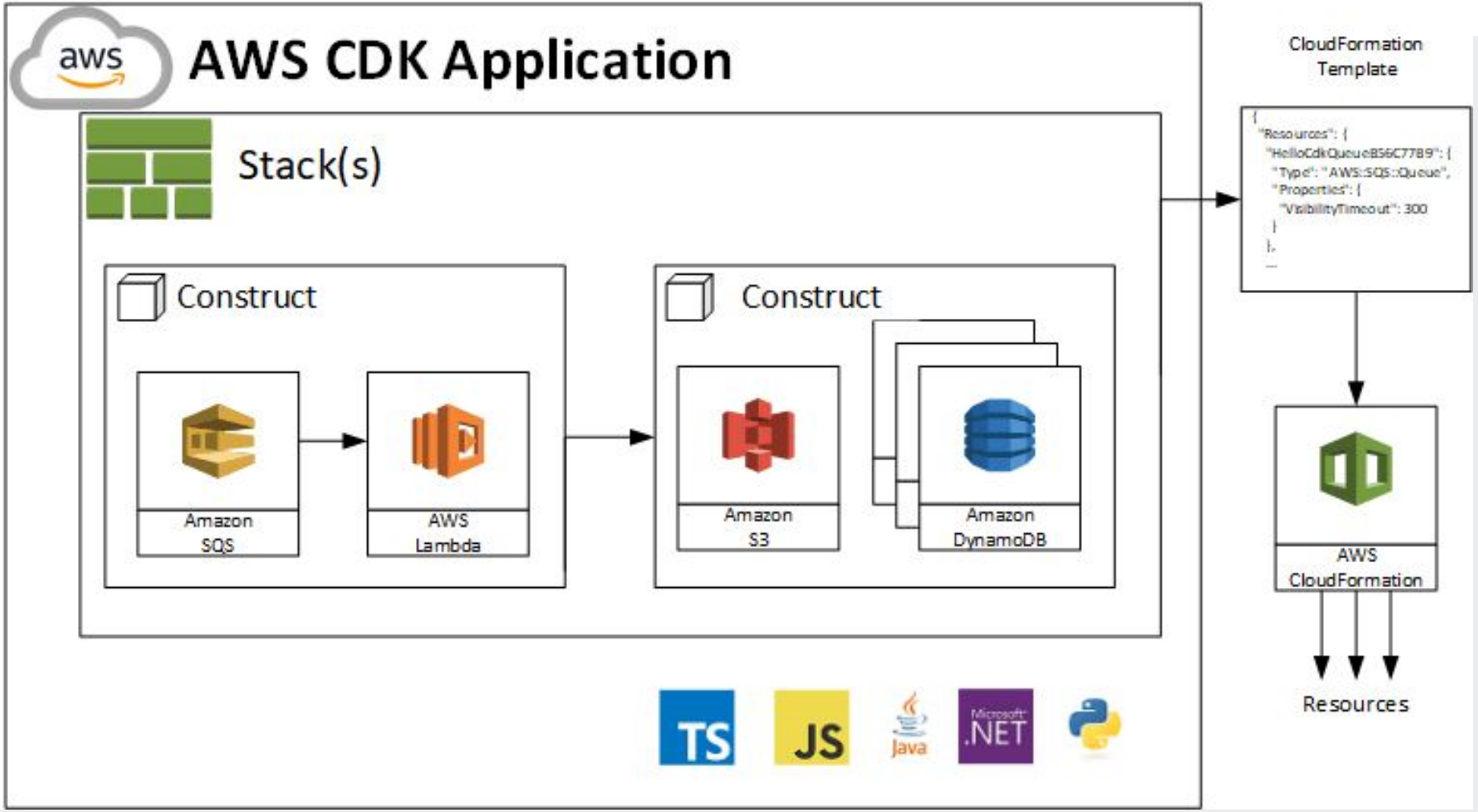
What's AWS CDK

The AWS Cloud Development Kit (AWS CDK) is an open source software development framework to define your cloud application resources using familiar programming languages.

Supported Languages: Python, JavaScript, TypeScript, Java and C#

WHY AWS CDK!!

- Easier Cloud Onboarding
- Customizable and Shareable
- Faster Development Process
- No Context Switching


AWS CDK Application
Stack(s)
Construct
Amazon SQS
AWS Lambda
Construct
Amazon S3
Amazon DynamoDB
TS
JS
Java
.NET
CloudFormation Template
{
"Resources": {
"HelloCdkQueueB56C7789": {
"Type": "AWS::SQS::Queue",
"Properties": {
"VisibilityTimeout": 300
}
},
...
AWS CloudFormation
Resources

THANK YOU!!