CloudFormation

If you are studying for AWS Developer Associate Exam, this guide will help you with quick revision before the exam. it can use as study notes for your preparation.

Dashboard | Other Certification Notes

CloudFormation

- 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 want to:
  - 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 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 staged with an identifier so you can easily see how much a stack costs you
  - We 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:
  - PC 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 re-upload 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 Stacks and StackSets

- A stack is a collection of AWS resources that we can manage as a single unit
- All the resources in a stack are defined by the stack's AWS CloudFormation template
- A stack, for instance, can include all the resources required to run a web application, such as a web server, a database, and networking rules
- If we no longer require that web application, we can simply delete the stack, and all of its related resources are deleted
- CloudFormation StackSets allow us to roll out our application into multiple regions and accounts. It is commonly used together with AWS Organizations

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

CloudFormation 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`
- Resource documentation:
  - All the resources can be found here: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
  - Example here (for an EC2 instance): http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html

Analysis of CloudFormation Templates

- Relevant documentation can be found here:
  - http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html
  - http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group.html
  - http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-eip.html

FAQ for resources

- Can I create a dynamic amount 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

CloudFormation 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

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

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

CloudFormation 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
- We use `Fn::FindInMap` to return a named value from a specific key

When would you use Mapping 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

CloudFormation 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 examples:
    - Creating a SSH Security Group as part of one template
    - Create an output that references that security group

Cross Stack Reference

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

CloudFormation Conditions

- 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

Defining Conditions

- 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 can be applied to resources / outputs / etc

CloudFormation Transforms

- Transforms can specify one or more macros that CloudFormation can use to process our template
- Macros are executed in the order they are specified
- CloudFormation supports some macros predefined by AWS

CloudFormation Intrinsic Functions

- Refs
  - The `Fn::Ref` function can be leveraged to reference
    - Parameters => returns the value of the parameter
    - Resources => returns the physical ID of the underlying resource (ex: EC2 ID)
  - The shorthand for this in YAML is `!Ref`
- `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
- `Fn::FindInMap`
  - We use `Fn::FindInMap` to return a named value from a specific key
  - `!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]`
- `Fn::ImportValue`
  - Import values that are exported in other templates
    - Use the `Fn::ImportValue` function
- `Fn::Join`
  - Join values with a delimiter
- `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
- Condition Functions (`Fn::If`, `Fn::Not`, `Fn::Equals`, etc...)
  - 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`

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