AWS CloudFormation - Best Practices

# AWS CloudFormation Best Practices

* AWS CloudFormation Best Practices are based on real-world experience from current AWS CloudFormation customers
* AWS CloudFormation Best Practices help provide guidelines on
  + how to plan and organize stacks,
  + create templates that describe resources and the software applications that run on them,
  + and manage stacks and their resources

#### **Required Mainly for Developer, SysOps Associate & DevOps Professional Exam**

## Planning and Organizing

### Organize Your Stacks By Lifecycle and Ownership

* Use the lifecycle and ownership of the AWS resources to help you decide what resources should go in each stack.
* By grouping resources with common lifecycles and ownership, owners can make changes to their set of resources by using their own process and schedule without affecting other resources.
* For e.g. Consider an Application using Web and Database instances. Both the Web and Database have a different lifecycle and usually the ownership lies with different teams. Maintaining both in a single stack would need communication and co-ordination between different teams introducing complexity. It would be best to have different stacks owned by the respective teams, so that they can update their resources without impacting each others’s stack.

### Use Cross-Stack References to Export Shared Resources

* With multiple stacks, there is usually a need to refer values and resources across stacks.
* Use cross-stack references to export resources from a stack so that other stacks can use them
* Stacks can use the exported resources by calling them using the Fn::ImportValue function.
* For e.g. Web stack would always need resources from the Network stack like VPC, Subnets etc.

### Use IAM to Control Access

* Use IAM to control access to
  + what AWS CloudFormation actions users can perform, such as viewing stack templates, creating stacks, or deleting stacks
  + what actions CloudFormation can perform on resources on their behalf
* Remember, having access to CloudFormation does not provide user with access to AWS resources. That needs to be provided separately.
* To separate permissions between a user and the AWS CloudFormation service, use a service role. AWS CloudFormation uses the service role’s policy to make calls instead of the user’s policy.

### Verify Quotas for All Resource Types

* Ensure that stack can create all the required resources without hitting the AWS account limits.

### Reuse Templates to Replicate Stacks in Multiple Environments

* Reuse templates to replicate infrastructure in multiple environments
* Use parameters, mappings, and conditions sections to customize and make templates reusable
* for e.g. creating the same stack in development, staging and production environment with different instance types, instance counts etc.

### Use Nested Stacks to Reuse Common Template Patterns

* Nested stacks are stacks that create other stacks.
* Nested stacks separate out the common patterns and components to create dedicated templates for them, preventing copy pasting across stacks.
* for e.g. a standard load balancer configuration can be created as nested stack and just used by other stacks

## ****Creating templates****

### Do Not Embed Credentials in Your Templates

* Use input parameters to pass in sensitive information such as DB password whenever you create or update a stack.
* Use the NoEcho property to obfuscate the parameter value.

### Use AWS-Specific Parameter Types

* For existing AWS-specific values, such as existing Virtual Private Cloud IDs or an EC2 key pair name, use AWS-specific parameter types
* AWS CloudFormation can quickly validate values for AWS-specific parameter types before creating your stack.

### Use Parameter Constraints

* Use Parameter constraints to describe allowed input values so that CloudFormation catches any invalid values before creating a stack.
* For e.g. constraints for database user name with min and max length

### Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances

* Use AWS::CloudFormation::Init resource and the cfn-init helper script to install and configure software applications on EC2 instances

### Validate Templates Before Using Them

* Validate templates before creating or updating a stack
* Validating a template helps catch syntax and some semantic errors, such as circular dependencies, before AWS CloudFormation creates any resources.
* During validation, AWS CloudFormation first checks if the template is valid JSON or a valid YAML. If both checks fail, AWS CloudFormation returns a template validation error.

## ****Managing stacks****

### Manage All Stack Resources Through AWS CloudFormation

* After launching the stack, any further updates should be done through CloudFormation only.
* Doing changes outside the stack can create a mismatch between the stack’s template and the current state of the stack resources, which can cause errors if you update or delete the stack.

### Create Change Sets Before Updating Your Stacks

* Change sets provides a preview of how the proposed changes to a stack might impact the running resources before you implement them
* CloudFormation doesn’t make any changes to the stack until you execute the change set, allowing you to decide whether to proceed with the proposed changes or create another change set.

### Use Stack Policies

* Stack policies help protect critical stack resources from unintentional updates that could cause resources to be interrupted or even replaced
* During a stack update, you must explicitly specify the protected resources that you want to update; otherwise, no changes are made to protected resources

### Use AWS CloudTrail to Log AWS CloudFormation Calls

* AWS CloudTrail tracks anyone making AWS CloudFormation API calls in the AWS account.
* API calls are logged whenever anyone uses the AWS CloudFormation API, the AWS CloudFormation console, a back-end console, or AWS CloudFormation AWS CLI commands.
* Enable logging and specify an Amazon S3 bucket to store the logs.

### Use Code Reviews and Revision Controls to Manage Your Templates

* Using code reviews and revision controls help track changes between different versions of your templates and changes to stack resources
* Maintaining history can help revert the stack to a certain version of the template.

## AWS Certification Exam Practice Questions

1. A company has deployed their application using CloudFormation. They want to update their stack. However, they want to understand how the changes will affect running resources before implementing the updated. How can the company achieve the same?
   1. Use CloudFormation Validate Stack feature
   2. Use CloudFormation Dry Run feature
   3. Use CloudFormation Stage feature
   4. **Use CloudFormation Change Sets feature**
2. You have multiple similar three-tier applications and have decided to use CloudFormation to maintain version control and achieve automation. How can you best use CloudFormation to keep everything agile and maintain multiple environments while keeping cost down?
   1. Create multiple templates in one CloudFormation stack.
   2. Combine all resources into one template for version control and automation.
   3. Use CloudFormation custom resources to handle dependencies between stacks
   4. **Create separate templates based on functionality, create nested stacks with CloudFormation.**
3. You are working as an AWS DevOps admins for your company. You are in-charge of building the infrastructure for the company’s development teams using CloudFormation. The template will include building the VPC and networking components, installing a LAMP stack and securing the created resources. As per the AWS best practices what is the best way to design this template?
   1. Create a single CloudFormation template to create all the resources since it would be easier from the maintenance perspective.
   2. Create multiple CloudFormation templates based on the number of VPC’s in the environment.
   3. Create multiple CloudFormation templates based on the number of development groups in the environment.
   4. **Create multiple CloudFormation templates for each set of logical resources, one for networking, and the other for LAMP stack creation.**