

TECHNOLOGY



Designing Applications and Architectures in AWS

(Aligned with AWS Solution Architect Associate Certification)

Monitoring and Automation



A Day in the Life of a Cloud Architect

You are a cloud architect in an organization and have been asked to enhance the stability and efficiency of various cloud-based applications.

A few servers recently crashed due to a high workload, forcing the company to upgrade its infrastructure with more robust servers.

This unexpected event took a significant amount of time and led to a substantial financial loss.



A Day in the Life of a Cloud Architect

Your responsibility is to prevent such failures in the future.

You've identified AWS services CloudWatch and CloudFormation as potential solutions. These tools can assist in monitoring resource performance and rapidly replicating the infrastructure.

To achieve these, you will learn a few concepts in this lesson that will help you find a solution for the given scenario.



Learning Objectives

By the end of this lesson, you will be able to:

- Monitor instances effectively by configuring and utilizing CloudWatch alarms
- Create CloudFormation templates to automate infrastructure provisioning and management
- Update resources within an existing infrastructure using CloudFormation, ensuring scalability and maintainability
- Build complete infrastructures efficiently using CloudFormation, streamlining the deployment process for AWS resources



TECHNOLOGY

CloudWatch

CloudWatch: Overview

CloudWatch is a service that monitors resources and applications run by users on AWS.



Amazon
CloudWatch

- It enables users to collect and access performance and operational data in the form of logs and metrics.
- CloudWatch provides comprehensive visibility into resource utilization, application performance, and operational health.

CloudWatch: Features



- It sets alarms and initiates automated actions.
- You can integrate CloudWatch with notification services.
- It monitors applications, infrastructure, and services.
- It helps in the collection and tracking of predefined and custom metrics.
- It creates dashboards to display metrics.

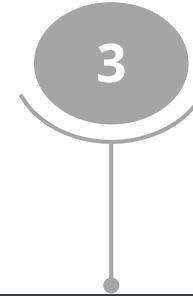
CloudWatch: Workflow

Here are the tasks to perform in the CloudWatch workflow:

Control: Collect metrics and logs from AWS services running on AWS and on-premises servers.



Act: Automate responses to operational changes using events and autoscaling with CloudWatch.



Compliance and Security: Integrate with IAM to track user activities on resources.



Monitor: Provide a uniform operational view through dashboards.



Analyze: Conduct real-time analysis using CloudWatch Metric Math.

CloudWatch: Metrics

CloudWatch Metrics offers performance metrics with two options: basic, which is free, and detailed, which is a paid service.



It maintains metric data for up to 15 months, enabling users to establish alarms based on specific metric values.



Users can also utilize CloudWatch Metrics for searching and displaying data on dashboards.



CloudWatch: Metrics



- A timestamp can be set up to two weeks in the past or two hours in the future.
- If the user does not provide a timestamp, CloudWatch will generate one based on the time it received the data point.

CloudWatch: Metrics



- Timestamps in CloudWatch consist of DateTime objects, including the full date, hours, minutes, and seconds.
- It's advisable to use Coordinated Universal Time (UTC) for time-related operations in CloudWatch.
- CloudWatch alarms evaluate metrics based on the current UTC.
- Using custom metrics with non-current UTC timestamps may lead to insufficient data states or delayed alarms.

CloudWatch: Metrics

CloudWatch retains metric data as follows:



- Data points with a period of fewer than 60 seconds are available for 3 hours.
- Data points with a period of 60 seconds are available for 15 days.
- Data points with a period of 300 seconds are available for 63 days.
- Data points with a period of 3600 seconds are available for 455 days (15 months).

CloudWatch: Metrics

CloudWatch aggregates data points initially published within a shorter period for long-term storage.



Example:

- If the user collects data at a one-minute interval, the data stays available for 15 days at a one-minute resolution.
- After 15 days, CloudWatch aggregates the data, making it retrievable at a five-minute resolution.
- After 63 days, CloudWatch further aggregates the data, making it available at a one-hour resolution.

CloudWatch: Metrics

Users employ an alarm to automatically initiate actions.



- An alarm monitors a single metric over a specified time, taking actions based on the metric's value in comparison to a threshold.
- The alarm issues an alert to an Auto Scaling policy or an Amazon SNS subject as the corresponding action.¹

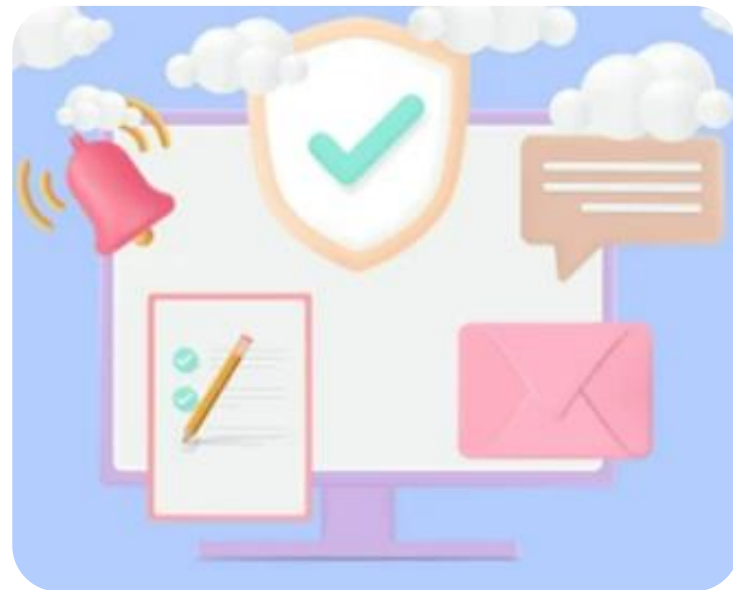
CloudWatch: Alarms



- The alarm invokes action only if the state change is sustained.
- The CloudWatch alarms do not invoke actions while in a particular state.
- A state change must be maintained for a specified number of periods before actions are invoked.

CloudWatch: Alarms

When setting up an alarm, a user must select a monitoring period equal to or greater than the metric's resolution.



High-resolution alarms incur higher charges than standard-resolution alarms.



CloudWatch: Alarms

An example:



- Amazon EC2's basic monitoring service provides metrics for an instance every five minutes.
- When setting an alarm on a basic monitoring metric, a user must select a specific duration.
- For high-resolution metrics, a user may choose a standard alarm with a period that is a multiple of 60 seconds.

CloudWatch: Logs

Users can utilize Amazon CloudWatch Logs to monitor, store, and access log files from sources such as Amazon EC2 instances, AWS CloudTrail, and Route 53.



CloudWatch: Logs

CloudWatch Logs enables users to centralize logs from all their systems, applications, and AWS services into a single, scalable service.



Once centralized, users can view these logs, search for specific error codes, filter by fields, or securely archive them for future analysis.



CloudWatch: Logs

The features of CloudWatch Logs include:



- **Alarms:** These notify API activities, aiding effective troubleshooting for the architect.
- **Retention Policies:** They help users to adjust the retention duration from one day to 10 years.
- **Route 53 DNS Queries Logging:** It allows the recording of DNS requests received by Route 53 resolver.

Automating EC2 Instance Shutdown with CloudWatch Alarm



Duration: 15 min

Problem Statement:

You have been assigned a task to create a CloudWatch alarm to automatically stop an instance based on CPU utilization.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a CloudWatch alarm



Evaluating CloudWatch Log Reports



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of evaluating CloudWatch Log reports by creating and graphing metrics from log data using CloudWatch Log Groups and Metric Filters.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Evaluate CloudWatch Log reports



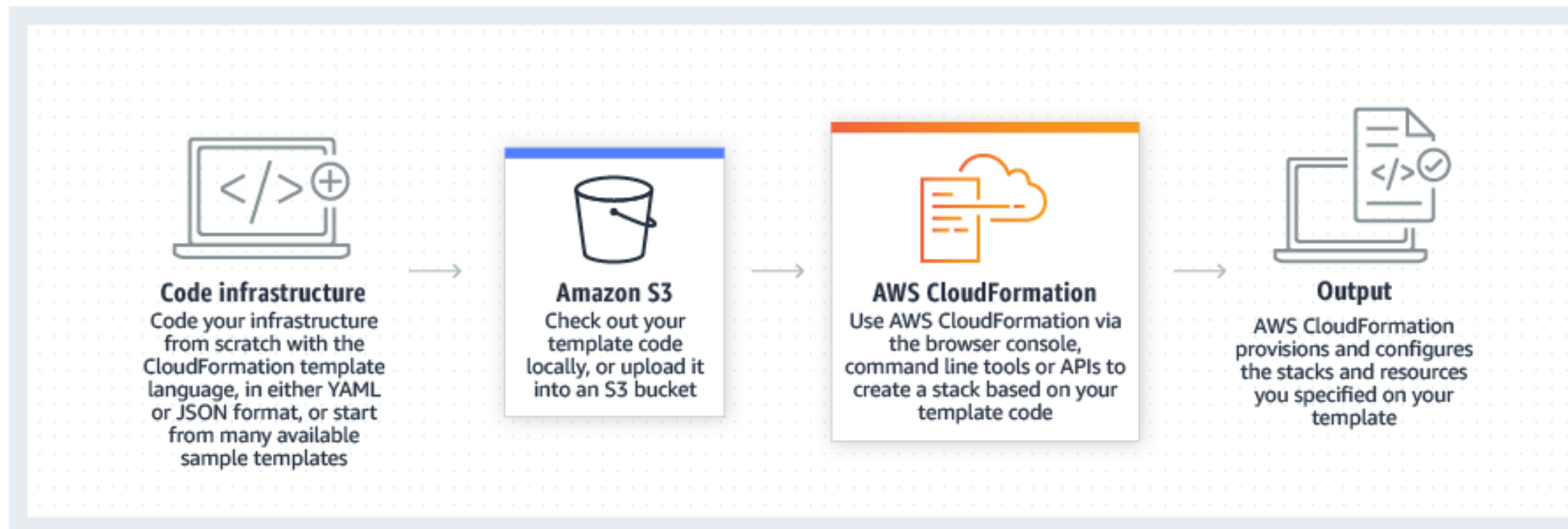
TECHNOLOGY

CloudFormation

Introduction to CloudFormation

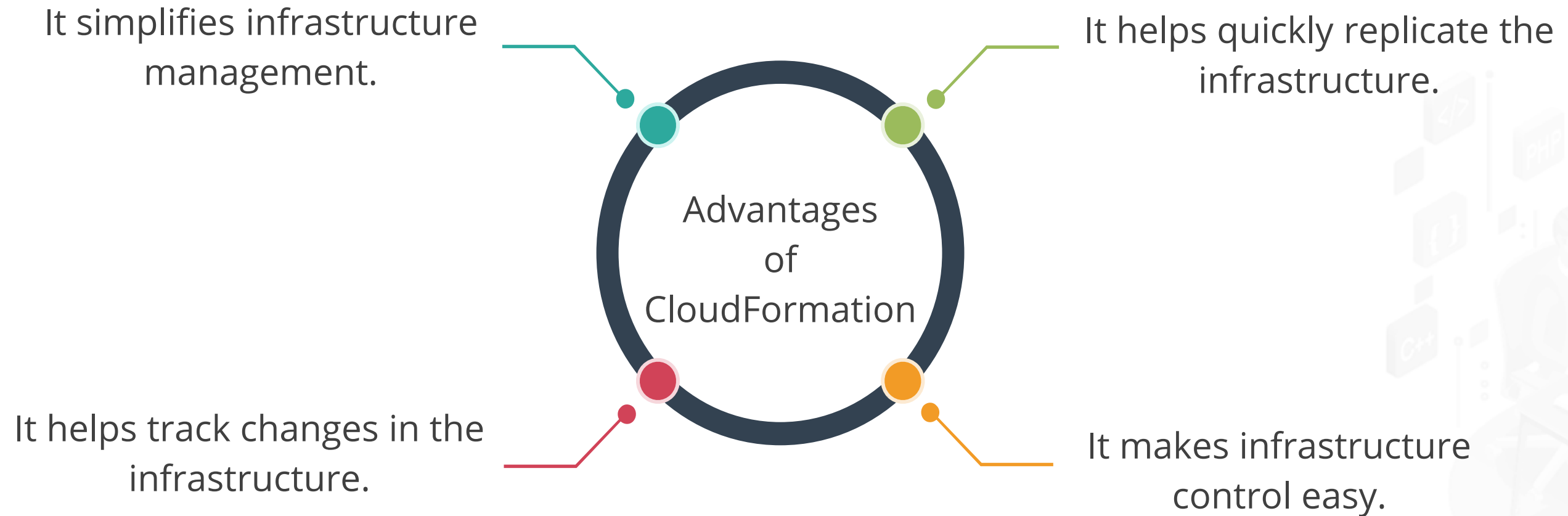


AWS CloudFormation is a service that helps an end-user automate AWS services by creating templates.



The CloudFormation interprets the template and makes the API calls to create the resources.

Introduction to CloudFormation



Introduction to CloudFormation

Templates and **Stacks** are important elements of CloudFormation.



A **template** is a JSON or YAML-formatted text file.



A **stack** is a collection of AWS resources that a user can manage as a single unit.

Types of Templates

Template in JSON format:

```
{
  "AWSTemplateFormatVersion" : "version date",
  "Description" : "JSON string",
  "Metadata" : {
    template metadata },
  "Parameters" : {
    set of parameters },
  "Rules" : {
    set of rules },
  "Mappings" : {
    set of mappings },
  "Conditions" : {
    set of conditions },
  "Transform" : {
    set of transforms},
  "Resources" : {
    set of resources },
  "Outputs" : {
    set of outputs }
}
```



Types of Templates

Template in YAML format:

```
AWSTemplateFormatVersion: "version date"
Description:
  String
Metadata:
  template metadata
Parameters:
  set of parameters
Rules:
  set of rules
Mappings:
  set of mappings
Conditions:
  set of conditions
Transform:
  set of transforms
Resources:
  set of resources
Outputs:
  set of outputs
```



Template Structure: Overview

The sections of the template are as follows:



Data tables (optional):

This section includes static configuration values, such as AMI names.



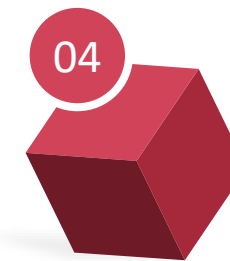
Parameters (optional):

This section includes input values that are provided while creating stacks.



Outputs (optional):

This section includes values that are returned whenever users view their stack's properties.



Resources (required):

This section includes the names and configurations of services to be added.

Template Structure: Overview

The sections of the template are as follows:



Conditions (optional)

This section includes actions to be taken based on conditional statements, such as equals.



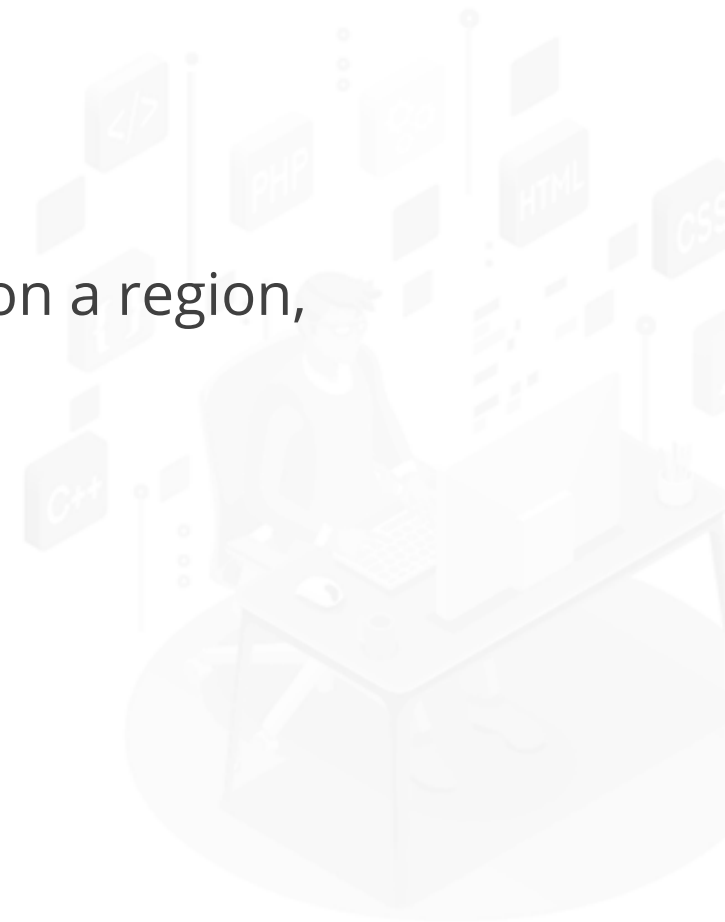
Mappings (optional)

This section has set values based on a region, such as AMIs.



Transform (optional)

This section includes snippets from outside the main template.



Template Structure: Overview

Template in JSON format

```
{ "AWSTemplateFormatVersion" : "version date",  
  
  "Description" : "JSON string",  
  "Metadata" : {  
    template metadata },  
  "Parameters" : {  
    set of parameters },  
  "Rules" : {  
    set of rules },  
  "Mappings" : {  
    set of mappings },  
  "Conditions" : {  
    set of conditions },  
  "Transform" : {  
    set of transforms },  
  "Resources" : {  
    set of resources },  
  "Outputs" : {  
    set of outputs  }}
```

The template contains the following sections:

- Template version
- Description
- Metadata
- Parameter
- Mapping
- Conditions
- Output
- Resources



Parameter Section: Example

Here is an example to show how to specify input parameters for a template:

```
"Parameters" : {  
  "InstanceTypeParameter": {  
    "Type" : "String",  
    "Default" : "t2.micro",  
    "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],  
    "Description" : "Enter t2.micro, m1.small, or m1.large. Default is  
t2.micro."  
  }  
}
```



Mapping Section: Example

Here is an example to show how to match a key to a corresponding set of named values:


```
"Mappings": {  
  "Mapping01" : {  
    "Key01" : {  
      "Name": "Value01"  
    }  
  
    "Key02": {  
      "Name" : "Value02"  
    }  
  
    "Key03": {  
      "Name" : "Value03"  
    }  
  }  
}
```





Condition Section

The properties of the condition section of the template are:



 The condition statement is optional. 



 It contains statements that define the circumstances under which the resources are created. 



Resources Section: Example

Here is an example to show how to declare the AWS resources that need to be included in the stack:

```
"Resources": {
  "SQSQ44CP0": {
    "Type": "AWS::SQS::Queue",
    "Properties": {},
    "Metadata": {
      "AWS::CloudFormation::Designer": {
        "id": "3f3e280c-2e3b-43d2-bdaa-9c9a529f6de8"
      }
    }
  },
  "S3B4XB7F": {
    "Type": "AWS::S3::Bucket",
    "Properties": {},
    "Metadata": {
      "AWS::CloudFormation::Designer": {
        "id": "e5cbfee9-294b-4688-afd2-43274d15a8bc"
      }
    }
  }
}
```

Creating an S3 Bucket Using CloudFormation



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of creating an S3 bucket stack using CloudFormation.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a template
2. Create an S3 bucket



Deleting an Existing Stack



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of deleting an existing stack, focusing on deleting an S3 bucket.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Locate an existing stack and delete it



Updating an Existing Stack



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of updating an existing stack, focusing on deleting an S3 bucket.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create an S3 Bucket using CloudFormation
2. Delete an S3 Bucket from the stack



Updating Input Parameters in an Existing Stack



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of updating input parameters in an existing CloudFormation stack.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Update the template and test the updates



Intrinsic Functions and Pseudo Parameters

Intrinsic Functions

Intrinsic functions are built-in functions that enable the users to assign values to properties that are available at runtime.

Examples of intrinsic function:

- Fn::Base64
- Fn::Cidr
- Fn::Join
- Fn::Select
- Fn::Split
- Fn::Transform
- Ref



Intrinsic Functions

The applications of intrinsic function are as follows:

They can be used to create stack resources conditionally.



Examples:
Fn::If, **Fn::Equals**, and **Fn::Not**

They are used only in specific sections of the template.



Examples: **resource properties**,
outputs, **metadata attributes**, and
update policy attributes

They are used in templates to assign values to properties that are not available until runtime.

Intrinsic Functions

An example to show how to declare resources using **Ref** intrinsic function:

```
"My EIP" : {  
  "Type" : "AWS::EC2::EIP",  
  "Properties" : {  
    "InstanceId" : {"Ref" : "MyEC2Instance"}  
  }  
}
```



Pseudo Parameters

Pseudo parameters are those parameters that are predefined by AWS CloudFormation.

01

They are not declared in the template.

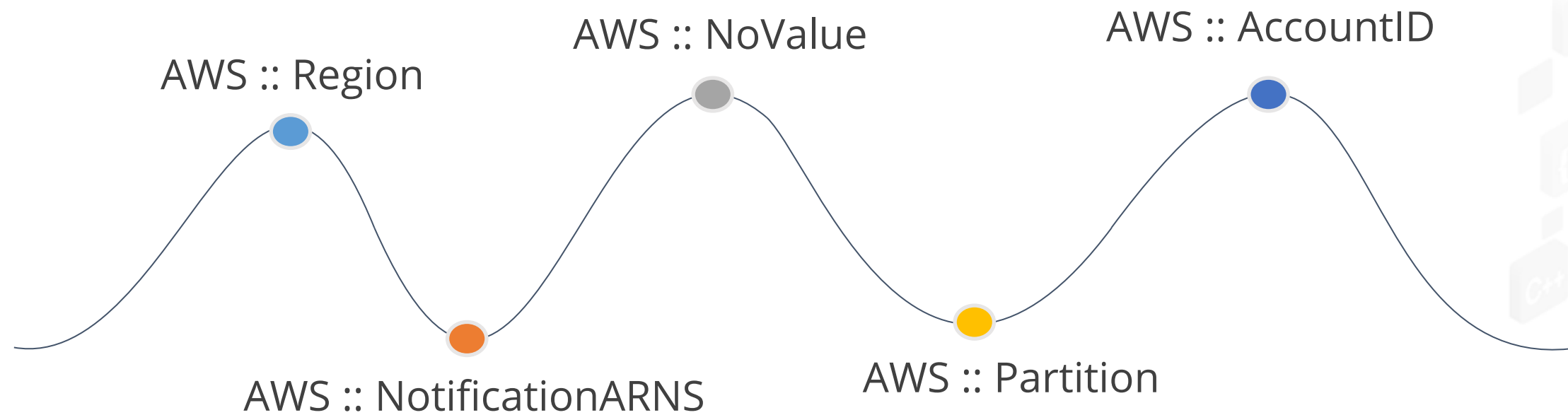
02

They can be referenced by using **Ref** intrinsic function.



Pseudo Parameters

The list of pseudo parameters are as follows:



Pseudo Parameters

An example to show how to assign the value of **AWS::Region** to the output value:

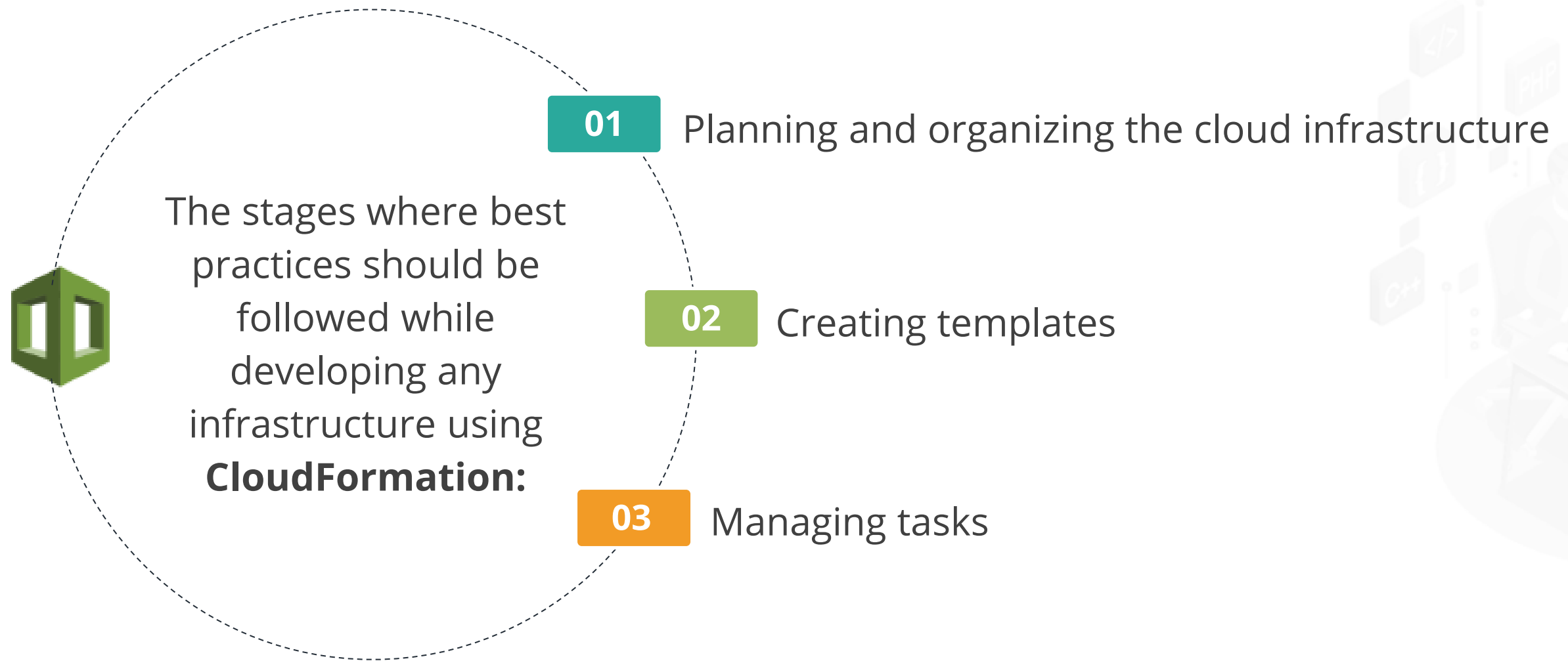
```
"Outputs" : {  
    "MyStacksRegion" : { "Values" { "Ref" :  
    "AWS::Region", } }  
}
```



CloudFormation: Best Practices

AWS CloudFormation: Best Practices

The best practices are recommendations that can help in the effective utilization of **AWS CloudFormation** throughout its entire workflow.



AWS CloudFormation: Best Practices

The list of practices to be followed while planning and organizing the cloud infrastructure are as follows:

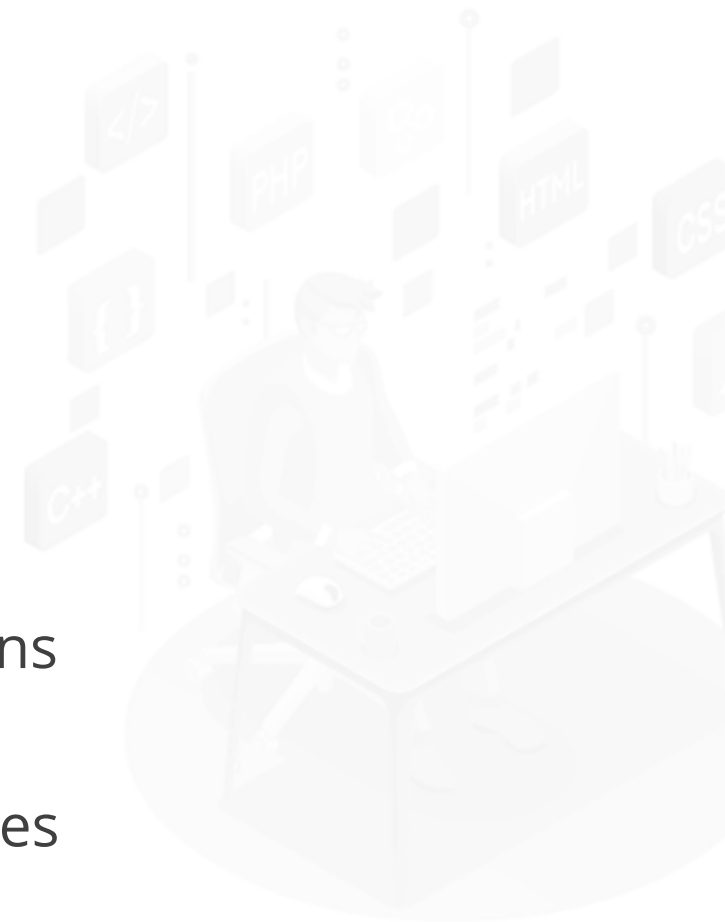
- Organize the stacks by life cycle and ownership
- Use cross-stack references to export shared resources
- Use IAM to control access
- Reuse templates to replicate stacks in multiple environments
- Verify quotas for all resource types
- Use modules to reuse resource configurations



AWS CloudFormation: Best Practices

The list of practices to be followed while creating templates are as follows:

- Do not use credentials in a template
- Use AWS-specific parameter types
- Use parameter constraints
- Use **AWS::CloudFormation::Init** to deploy software applications on Amazon EC2 instances
- Use code reviews and revision controls to manage the templates
- Update the Amazon EC2 instances regularly



AWS CloudFormation: Best Practices

The list of practices to be followed while creating templates are as follows:

- Manage all stack resources through AWS CloudFormation
- Create change sets before updating the stacks
- Use stack policies
- Use AWS CloudTrail to log AWS CloudFormation calls
- Use code reviews and revision controls to manage the templates
- Update the Amazon EC2 instances regularly



Adding Deletion Policy to Protect Resources



Duration:10 min

Problem Statement:

You have been assigned a task to demonstrate the process of adding a deletion policy to protect specific AWS S3 bucket resources in a CloudFormation stack.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Edit deletion policy for specific S3 bucket
2. Delete the stack



Setting up an Auto Scaling Group with a Launch Template



Duration:15 min

Problem Statement:

You have been assigned a task to set up an auto-scaling group using a launch template in AWS.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a Launch template
2. Create an Auto Scaling group



Creating a Resource Group for Organizing AWS Resources



Duration:10 min

Problem Statement:

You have been assigned a task to create a resource group, which is a valuable tool for organizing and managing resources efficiently within your AWS environment.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Launch multiple EC2 instances
2. Create a Resource group



Key Takeaways

- CloudWatch is a monitoring and observational service used to monitor resources and applications that a user runs on AWS.
- AWS CloudFormation is a service that helps to model and set up AWS resources in an efficient manner.
- Templates and stacks are the major elements of CloudFormation.
- Intrinsic functions are used to assign values to properties that are only available at runtime.



Creating an Alarm Using CloudWatch

Duration: 30 Mins



Project agenda: To create an alarm using CloudWatch that will allow you to watch CloudWatch metrics (CPU utilization) with a given threshold and receive notifications when the metrics fall outside the threshold levels that you configure.

Description: Launch 3 virtual machine instances (Linux), perform tasks on these VMS of your choice, and set up a dashboard with metrics showing CPU utilization of all 3 VMS.

Perform the following:

1. Launch Linux VMs
2. Connect SSH to VMs
3. Perform Linux-related tasks on the VM
4. Configure the CloudWatch services
5. Create metrics for CPU utilization for all VMs
6. Create an alarm and send a notification through SNS

TECHNOLOGY

Thank You