

# **CHAPTER 01**

## **INTRODUCTION**

### **INTRODUCTION & OBJECTIVE**

AWS Cloud Formation simplifies provisioning and management on AWS. You can create templates for the service or application architectures you want and have AWS Cloud Formation use those templates for quick and reliable provisioning of the services or applications (called “stacks”).

You can also easily update or replicate the stacks as needed. AWS Cloud Formation templates are JSON (java script object Notation) script formatted text file. The collection of sample templates in AWS will help you get started with AWS Cloud Formation and quickly build your own templates. Using AWS Cloud Formation templates, it provides an easy way to quickly provision,configure deploy and update your applications.Quickly replicate infrastructure,easily control and track changes to your infrastructure.Gives developers and system administrators an easy way to create and manage a collection of related AWS resources.Understands dependencies and supports rollbacks and versioning.Allows for reusable component design strategies.Supports JSON and YAML formats.It automate, manage your applications,quickly replicate your infrastructure,easily control and track changes to your infrastructure and saves a lot of time.Just choose the resources and configurations you need.Customize your template as needed basis.

### **1.1 EXISTING SYSTEM**

In existing system,Building an application have to manage all the infrastructure resources that an application needs and hard to repeat if configuring everything manually like software configuration(PHP),serverscaling(APACHE),deployment and database setup(MYSQL) leads to heavy lifting. server admin or app developer has to login to cloud account and they have to provision server attach network, attach roles, storage, security groups this will consume hours to deploy single application.

## DRAWBACKS

- waste of time configuring everything(PHP,MYSQL,SSH...) manually ,so that your application development takes lot of time.
- complexity leads to get your application to your users slower.

## 1.2 PROPOSED SYSTEM

So will be going to use Cloud Formation templates to provision these resources within minutes and update the template as needed basis. so that you can focus on developing instead of doing all that heavy lifting.

Using AWS Cloud Formation templates,it provides an easy way to quickly provision, configure deploy and update your applications. It automate, manage your applications and saves a lot of time. without Cloud Formation template, you'd need to manually configure everything from web framework configuration ,installation scripts, initialization tasks or database setups for each instance. with cloud formation template you setup and configure whatever your application need at once and let cloud formation template automatically configure instance.

## ADAVANTAGES OF PROPOSED SYSTEM

- **Reliability:** The project performs intended function with required precision; hence this project is very reliable.Automating your infrastructure gets your application to your users faster,helps to manage complexity.
- **Feasibility:** The project maintenance is very easy and modifications can be made in the existing system in future.
- **Security:** provisions resources in secure manner and provides security to your applications from failure and downtime.
- Saves manual work and time.

- It lets you focus on building amazing applications and services for your users, without having to spend a lot of time manually configuring instances, software and databases.

### **1.3 SYSTEM SCOPE**

AWS CloudFormation is a service that gives developers and businesses an easy way to create a collection of related AWS resources and provision them in an orderly and predictable fashion. There is no additional charge for AWS CloudFormation. You pay for AWS resources (such as Amazon EC2 instances, Elastic Load Balancing load balancers, etc.) created using AWS CloudFormation in the same manner as if you created them manually. You only pay for what you use, as you use it; there are no minimum fees and no required upfront commitments.

- Security
- High- scalability
- Cost-effective
- Reliability
- Flexibility
- Easy Deployment

It lets you focus on building amazing applications and services for your users, without having to spend a lot of time manually configuring instances, software and databases. AWS CloudFormation provisions your resources in a safe, repeatable manner, allowing you to build and rebuild your infrastructure and applications using json script file formatted templates.

## CHAPTER 02

### LITERATURE SURVEY

AWS CloudFormation is a service that gives developers and businesses an easy way to create a collection of related AWS resources and provision them in an orderly and predictable fashion. Before, server admin or app developer has to login to cloud account and they have to provision server attach network, attach roles, storage, security groups this will consume hours to deploy single application and lot of wastage of time.

AWS CloudFormation introduces two new concepts: The template, a JSON or YAML-format, text-based file that describes all the AWS resources you need to deploy to run your application and the stack, the set of AWS resources that are created and managed as a single unit when AWS CloudFormation instantiates a template

#### **Simplify Infrastructure Management**

For a scalable web application that also includes a back-end database, you might use an Auto Scaling group, an Elastic Load Balancing load balancer, and an Amazon Relational Database Service database instance. Normally, you might use each individual service to provision these resources. And after you create the resources, you would have to configure them to work together. All these tasks can add complexity and time before you even get your application up and running.

Instead, you can create or modify an existing AWS CloudFormation template. A template describes all of your resources and their properties. When you use that template to create an AWS CloudFormation stack, AWS CloudFormation provisions the Auto Scaling group, load balancer, and database for you. After the stack has been successfully created, your AWS resources are up and running. You can delete the stack just as easily, which deletes all the resources in the stack. By using AWS CloudFormation, you easily manage a collection of resources as a single unit.

## **Quickly Replicate Your Infrastructure**

If your application requires additional availability, you might replicate it in multiple regions so that if one region becomes unavailable, your users can still use your application in other regions. The challenge in replicating your application is that it also requires you to replicate your resources. Not only do you need to record all the resources that your application requires, but you must also provision and configure those resources in each region.

When you use AWS CloudFormation, you can reuse your template to set up your resources consistently and repeatedly. Just describe your resources once and then provision the same resources over and over in multiple regions.

## **Easily Control and Track Changes to Your Infrastructure**

In some cases, you might have underlying resources that you want to upgrade incrementally. For example, you might change to a higher performing instance type in your Auto Scaling launch configuration so that you can reduce the maximum number of instances in your Auto Scaling group. If problems occur after you complete the update, you might need to roll back your infrastructure to the original settings. To do this manually, you not only have to remember which resources were changed, you also have to know what the original settings were.

When you provision your infrastructure with AWS CloudFormation, the AWS CloudFormation template describes exactly what resources are provisioned and their settings. Because these templates are text files, you simply track differences in your templates to track changes to your infrastructure, similar to the way developers control revisions to source code. For example, you can use a version control system with your templates so that you know exactly what changes were made, who made them, and when. If at any point you need to reverse changes to your infrastructure, you can use a previous version of your template.

## **CHAPTER 03**

### **SOFTWARE REQUIREMENT ANALYSIS**

#### **3.1 HARDWARE REQUIREMENTS**

- RAM : 1GB Ram
- Hard Disk : 1CPU

#### **3.2 SOFTWARE REQUIREMENTS**

- Operating System: Windows 7 or later(32 or 64bit)
- AWS management console account
- Linux (ubuntu/putty),Apache,Mysql,PHP
- Json(javascript object notation)templates

#### **3.3 SYSTEM MODULES**

1.Create Stack:Specify required resources in template,stack details,Configure stack options.

2.Launch Stack:Copy ip address of running instance that helps to Install and deploy Wordpress on to a single Ec2 instance.PHP,Apache works on frontend and mysql database,linux on backend.

AWS CloudFormation templates are JSON or YAML-formatted text files that are comprised of five types of elements:

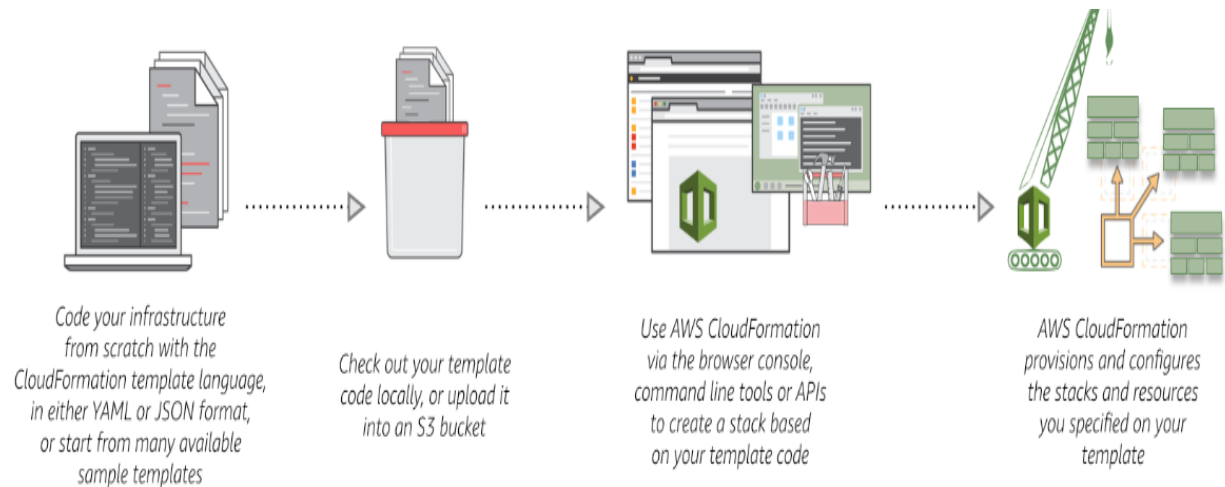
1. An optional list of template parameters (input values supplied at stack creation time)
2. An optional list of output values (e.g. the complete URL to a web application)
3. The list of AWS resources and their configuration values

With parameters, you can customize aspects of your template at run time, when the stack is built. For example, the Amazon RDS database size, Amazon EC2 instance types, database and web server port numbers can be passed to AWS CloudFormation when a stack is created. Each parameter can have a default value and description and may be marked as “NoEcho” in order to hide the actual value you enter on the screen.

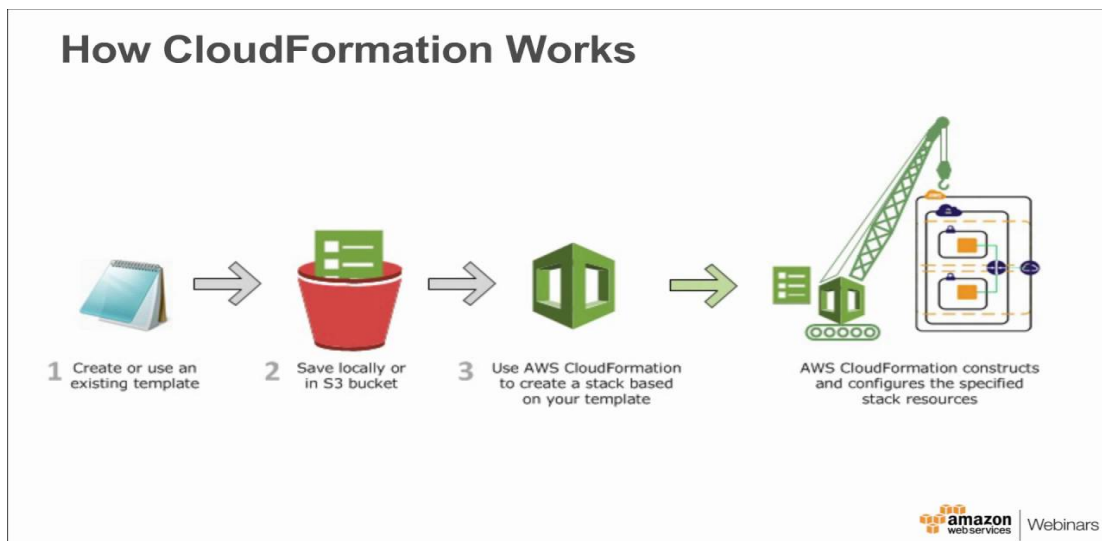
## CHAPTER 04

### SYSTEM DESIGN

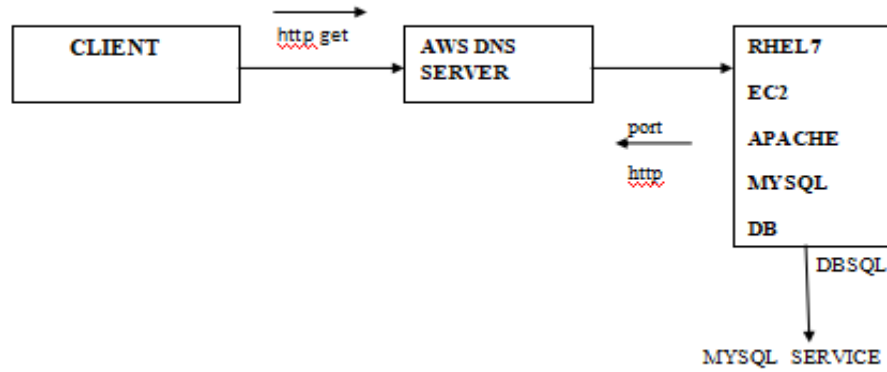
#### 4.1 SYSTEM ARCHITECTURE



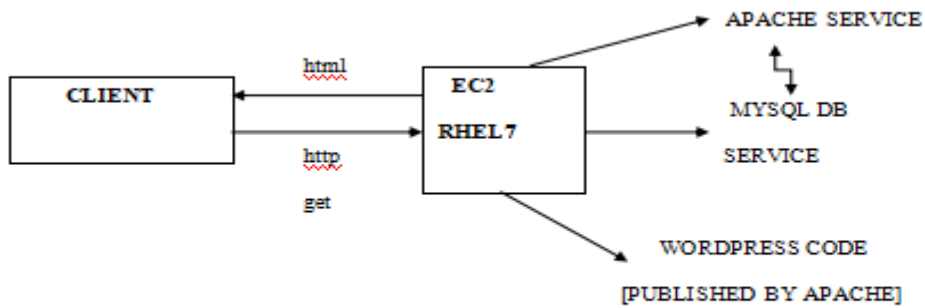
#### STEPS SHOWING HOW CLOUDFORMATION WORKS



## 4.2 DATA FLOW DIAGRAM



### Data Flow Diagram in Detail





## CHAPTER 05

### IMPLEMENTATION

#### 5.1 MODULE DESCRIPTION

A template is a JSON or YAML formatted text file that describes your AWS infrastructure.

#### STEP 1: CREATE A CLOUDFORMATION STACK

CloudFormer is itself an AWS CloudFormation stack, so the first step is to create and launch the stack from the AWS CloudFormation management console.

Firstly, select the AWS Region.

##### **To create a stack using the AWS CloudFormation Console**

- Log in to the AWS CloudFormation console and click **Create stack**.

##### **Step 1: Specify Template**

1. Select **template is ready**, and in the **specify template** section, select **upload a sample template** and then **Upload a template file**(Json/Yaml).

In the template text file, software configuration(PHP), server scaling(APACHE), deployment and database setup(MYSQL) everything is configured in the json format.

## CREATE STACK

The screenshot shows the 'Create stack' wizard in the AWS Management Console. On the left, a sidebar lists four steps: Step 1 (Specify template), Step 2 (Specify stack details), Step 3 (Configure stack options), and Step 4 (Review). Step 1 is currently active. The main content area is titled 'Create stack' and contains two sections. The first section, 'Prerequisite - Prepare template', explains that every stack is based on a template (JSON or YAML) and provides three radio button options: 'Template is ready' (selected), 'Use a sample template', and 'Create template in Designer'. The second section, 'Specify template', explains that a template is a JSON or YAML file describing stack resources and properties. It has a 'Template source' section with two radio button options: 'Amazon S3 URL' and 'Upload a template file' (selected). Below this is an 'Upload a template file' section with a 'Choose file' button and a text input field containing 'minipro\_json%2520file'. A note below the input field states 'JSON or YAML formatted file'.

Step 1  
Specify template

Step 2  
Specify stack details

Step 3  
Configure stack options

Step 4  
Review

### Create stack

#### Prerequisite - Prepare template

Prepare template  
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Template is ready ☐ Use a sample template ☐ Create template in Designer

#### Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source  
Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL ☒ Upload a template file

Upload a template file

Choose file

JSON or YAML formatted file

1. Click **Next** to specify the stack name and input parameters.

### Step 2. Specify Stack Details

1. Specify a stack name in the **Name** field.
2. In the **Parameters** section, type a DBusername and DBpassword , Wordpress database admin account password, Wordpress DBname , DBroot password, Specify SSH location, keyname.
3. You can't leave the password blank, and you can't use special characters in the password (such as ; & ! " £ \$ % ^ ( ) / ).

## SPECIFY STACK DETAILS

Step 2

Specify stack details

Step 3

Configure stack options

Step 4

Review

### Stack name

Stack name

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

### Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

**DBName**  
The WordPress database name

**DBPassword**  
The WordPress database admin account password

**DBRootPassword**  
MySQL root password

**DBUser**  
The WordPress database admin account username

**InstanceType**  
WebServer EC2 instance type

**KeyName**  
Name of an existing EC2 KeyPair to enable SSH access to the instances

**SSHLocation**  
The IP address range that can be used to SSH to the EC2 instances

Cancel

Previous

Next

4.Click Next.

1

### Step 3. Configure Stack Options

#### Advanced options

You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

► **Stack policy**

Defines the resources that you want to protect from unintentional updates during a stack update.

► **Rollback configuration**

Specify alarms for CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm threshold, CloudFormation rolls it back. [Learn more.](#)

► **Notification options**

► **Stack creation options**

Cancel

Previous

Next

Click **next**.

## Step 4. Review

Step 1  
Specify template

Step 2  
Specify stack details

Step 3  
Configure stack options

Step 4  
**Review**

### Review wpsite

Step 1: Specify template Edit

Template

Template URL  
https://s3.ap-south-1.amazonaws.com/cf-templates-ujnnsagv98vc-ap-south-1/2019181co5-minipro\_json%2520file

Stack description  
AWS CloudFormation Sample Template WordPress\_Single\_Instance: WordPress is web software you can use to create a beautiful website or blog. This template installs WordPress with a local MySQL database for storage. It demonstrates using the AWS CloudFormation bootstrap scripts to deploy WordPress. **\*\*WARNING\*\*** This template creates an Amazon EC2 instance. You will be billed for the AWS resources used if you create a stack from this template.

Estimate cost not available

Step 2: Specify stack details Edit

Parameters (7)

Key	Value
DBName	wordpressdb
DBPassword	*****
DBRootPassword	*****
DBUser	*****
InstanceType	t2.small
KeyName	AWScloud
SSHLocation	0.0.0.0/0

Step 3: Configure stack options

Tags (0)

## Review of Stack

**Notification options**

No notification options  
There are no notification options defined

**Stack creation options**

Rollback on failure  
Enabled

Timeout  
-

Termination protection  
Disabled

► Quick-create link

CancelPreviousCreate change setCreate stack

After you finish reviewing the stack information, click **Create** to start creating the Cloud Formation stack.

It goes through the normal stack creation process, which can take a few minutes.

### **STEP 2: LAUNCH THE CLOUD FORMATION STACK**

After the Cloud Formation stack's status is **CREATE\_COMPLETE**, you can launch the stack.

## STACK CREATION COMPLETE

### Stacks (1)

Create stack
Refresh

Active

View nested
1

wpsite
2019-06-30 10:49:09 UTC+0530
CREATE\_COMPLETE

2019-06-30 10:49:09 UTC+0530
CREATE\_COMPLETE

### Events

Timestamp	Logical ID	Status	Status reason
2019-06-30 10:50:57 UTC+0530	wpsite	CREATE_COMPLETE	-
2019-06-30 10:50:56 UTC+0530	WebServer	CREATE_COMPLETE	-
2019-06-30 10:50:55 UTC+0530	WebServer	CREATE_IN_PROGRESS	Received SUCCESS signal with Uniqueld i-09a1a408ca55d...
2019-06-30 10:49:22 UTC+0530	WebServer	CREATE_IN_PROGRESS	Resource creation Initiated
2019-06-30 10:49:21 UTC+0530	WebServer	CREATE_IN_PROGRESS	-
2019-06-30 10:49:18 UTC+0530	WebServerSecurityGroup	CREATE_COMPLETE	-
2019-06-30 10:49:17 UTC+0530	WebServerSecurityGroup	CREATE_IN_PROGRESS	Resource creation Initiated
2019-06-30 10:49:13 UTC+0530	WebServerSecurityGroup	CREATE_IN_PROGRESS	-
2019-06-30 10:49:09 UTC+0530	wpsite	CREATE_IN_PROGRESS	User Initiated

- click on Ec2 running instances.
- Installs and deploys Wordpress on to a single Ec2 instance.

## Resources

You are using the following Amazon EC2 resources in the Asia Pacific (Mumbai) region:

1 Running Instances	0 Elastic IPs
0 Dedicated Hosts	0 Snapshots
1 Volumes	0 Load Balancers
1 Key Pairs	2 Security Groups
0 Placement Groups	

## Launch Instance

Launch Instance

Connect

Actions

Filter by tags and attributes or search by keyword

1 to 1 of 1

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4
		i-09a1a408ca55dd2bf	t2.small	ap-south-1a	running	2/2 checks ...	None	ec2-13-127-38-140.ap-...	13.1.

Instance ID

i-09a1a408ca55dd2bf

Instance state

running

Instance type

t2.small

Elastic IPs

Availability zone

ap-south-1a

Public DNS (IPv4)

ec2-13-127-38-140.ap-south-1.compute.amazonaws.com

IPv4 Public IP

13.127.38.140

IPv6 IPs

-

Private DNS

ip-172-31-32-116.ap-south-1.compute.internal

Private IPs


172.31.32.116

- copy the ip address and open <http://ipaddress/wordpress> via the browser console.



## Wordpress Installation

13.127.38.140/wordpress/wp-admin/install.php



### Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

### Information needed

Please provide the following information. Don't worry, you can always change these settings later.

<b>Site Title</b>	<input type="text" value="AMR CEMENTS"/>
<b>Username</b>	<input type="text" value="RISHIKA"/> <small>Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.</small>
<b>Password</b>	<input type="password" value="SHKVD3\$s3u*#mz6uKE"/> <input type="button" value="Hide"/> <div>Strong</div> <p><b>Important:</b> You will need this password to log in. Please store it in a secure location.</p>
<b>Your Email</b>	<input type="text" value="rishikaredd@gmail.com"/> <small>Double-check your email address before continuing.</small>

- Click on install, you will be directed to your wordpress login site.

## 5.2 SOFTWARE ENVIRONMENT

Creation of stack using json templates,launching stack,provisioning of resources is done in aws management console by creating AWS account.AWS provides EC2 instances in which Redhat operating system is used.wordpress sample code is used to deploy on this operating system.

## 5.3 SOURCE CODE

```
{  
  
  "AWSTemplateFormatVersion" : "2010-09-09",  
  
  "Description": "AWS CloudFormation Sample Template WordPress_Single_Instance:  
WordPress is web software you can use to create a beautiful website or blog. This template  
installs WordPress with a local MySQL database for storage. It demonstrates using the  
AWS CloudFormation bootstrap scripts to deploy WordPress. **WARNING** This  
template creates an Amazon EC2 instance. You will be billed for the AWS resources used  
if you create a stack from this template.",  
  
  "Parameters" :  
  
    {  
  
      "KeyName": {  
  
        "Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",  
  
        "Type": "AWS::EC2::KeyPair::KeyName",  
  
        "ConstraintDescription" : "must be the name of an existing EC2 KeyPair."  
  
      },  
  
      "InstanceType" : {  
  
        "Description" : "WebServer EC2 instance type",  
  
        "Type" : "String",  
  
        "Default" : "t2.small",  
  
        "AllowedValues" : [ "t1.micro", "t2.nano", "t2.micro", "t2.small", "t2.medium", "t2.large",  
"m1.small", "m1.medium", "m1.large", "m1.xlarge", "m2.xlarge", "m2.2xlarge",  
"m2.4xlarge", "m3.medium", "m3.large", "m3.xlarge", "m3.2xlarge", "m4.large",  
"m4.xlarge", "m4.2xlarge", "m4.4xlarge", "m4.10xlarge", "c1.medium", "c1.xlarge", ]  
      }  
    }  
}
```

```

"ConstraintDescription" : "must be a valid EC2 instance type."

},

"SSHLocation": {

"Description": "The IP address range that can be used to SSH to the EC2 instances",

"Type": "String",

"MinLength": "9",

"MaxLength": "18",

"Default": "0.0.0.0/0",

"AllowedPattern": "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",

"ConstraintDescription": "must be a valid IP CIDR range of the form x.x.x.x/x."

},

"DBName" : {

"Default": "wordpressdb",

"Description" : "The WordPress database name",

"Type": "String",

"MinLength": "1",

"MaxLength": "64",

"AllowedPattern" : "[a-zA-Z][a-zA-Z0-9]*",

"ConstraintDescription" : "must begin with a letter and contain only alphanumeric
characters."

},

```

```

"DBUser" : {

"NoEcho": "true",

"Description" : "The WordPress database admin account username",

"Type": "String",

"MinLength": "1",

"MaxLength": "16",

"AllowedPattern" : "[a-zA-Z][a-zA-Z0-9]*",

"ConstraintDescription" : "must begin with a letter and contain only alphanumeric
characters."

},

"DBPassword" : {

"NoEcho": "true",

"Description" : "The WordPress database admin account password",

"Type": "String",

"MinLength": "8",

"MaxLength": "41",

"AllowedPattern" : "[a-zA-Z0-9]*",

"ConstraintDescription" : "must contain only alphanumeric characters."

},

"DBRootPassword" : {

"NoEcho": "true",

```

```

"Description" : "MySQL root password",

"Type": "String",

"MinLength": "8",

"MaxLength": "41",

"AllowedPattern" : "[a-zA-Z0-9]*",

"ConstraintDescription" : "must contain only alphanumeric characters."

}

},

"Mappings" : {

"AWSInstanceType2Arch" : {

"t1.micro"      : { "Arch" : "HVM64"  },

"t2.nano"      : { "Arch" : "HVM64"  },

"t2.micro"      : { "Arch" : "HVM64"  },

"t2.small"     : { "Arch" : "HVM64"  },

"AWSRegionArch2AMI" :

    "us-east-1"  : { "HVM64" : "ami-0080e4c5bc078760e", "HVMG2" : "ami-0aeb704d503081ea6"},

    "us-west-2"  : { "HVM64" : "ami-01e24be29428c15b2", "HVMG2" : "ami-0fe84a5b4563d8f27"},

    "us-west-1"  : { "HVM64" : "ami-0ec6517f6edbf8044", "HVMG2" : "ami-0a7fc72dc0e51aa77"},

},

```

```

"Resources" : {

  "WebServerSecurityGroup" : {

    "Type" : "AWS::EC2::SecurityGroup",

    "Properties" : {

      "GroupDescription" : "Enable HTTP access via port 80 locked down to the load balancer
+ SSH access",

      "SecurityGroupIngress" : [

        { "IpProtocol" : "tcp", "FromPort" : "80", "ToPort" : "80", "CidrIp" : "0.0.0.0/0" },

        { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : { "Ref" :
"SSHLocation" } }

      ]

    },

    "WebServer" : {

      "Type" : "AWS::EC2::Instance",

      "install_cfn" : {

        "files" : {

          "/etc/cfn/cfn-hup.conf" : {

            "content" : { "Fn::Join" : [ "", [

              "[main]\n",

              "stack=", { "Ref" : "AWS::StackId" }, "\n",

              "region=", { "Ref" : "AWS::Region" }, "\n"

            ]

```

```

"mode" : "000400",

"owner" : "root",

"group" : "root"

}},

"install_wordpress" : {

"packages" : {

"yum" : {

"php70" : [],

"php70-mysqlnd" : [],

"php70-imap" : [],

"php70-pecl-memcache" : [],

"php70-pecl-apcu" : [],

"php70-gd" : [],

"mysql" : [],

"mysql-server" : [],

"mysql-devel" : [],

"mysql-libs" : [],

"httpd24" : []

}},

"02_create_database" : {

```



```

"command" : { "Fn::Join" : [ "", [ "mysql -u root --password=", { "Ref" :
"DBRootPassword" }, " " < /tmp/setup.mysql" ] ] },

"test" : { "Fn::Join" : [ "", [ "$(mysql ", { "Ref" : "DBName" }, " -u root --password=", {
"Ref" : "DBRootPassword" }, " " > /dev/null 2>&1 < /dev/null); (( $? != 0 ))" ] ] }

},

"03_configure_wordpress" : {

"command" : "/tmp/create-wp-config",

"cwd" : "/var/www/html/wordpress"

}}}},

"#!/bin/bash -xe\n",

"yum update -y aws-cfn-bootstrap\n",

"/opt/aws/bin/cfn-init -v ", "--stack ", { "Ref" : "AWS::StackName" }, "--resource
WebServer ", "--configsets wordpress_install ", "--region ", { "Ref" : "AWS::Region" },
"\n", "/opt/aws/bin/cfn-signal -e $? ", "--stack ", { "Ref" : "AWS::StackName" }, "--resource
WebServer ", "--region ", { "Ref" : "AWS::Region" }, "\n" ] ] }

"Outputs" : {

"WebsiteURL" : {

"Value" : { "Fn::Join" : [ "", [ "http://", { "Fn::GetAtt" : [ "WebServer", "PublicDnsName"
] }, "/wordpress" ] ] },

"Description" : "WordPress Website"

}}

```

## CHAPTER 06

### TESTING

#### 6.1 TESTING METHODOLOGIES

Amazon Web Services (AWS) offers AWS Developer Tools, a set of services designed to enable customers to rapidly and safely deliver software. Together, these services help you follow continuous integration and continuous delivery practices, including secure storage and version control, and enable you to automatically build, validate, and deploy your code. Many customers use AWS CloudFormation to manage their infrastructure as code and to help deploy AWS resources in a controlled and predictable way. As with other source code, DevOps teams are commonly tasked with validating AWS CloudFormation templates before launch to ensure they follow industry best practices and satisfy company-specific business and governance requirements. Often, they will use AWS Developer Tools to create their own development and deployment pipelines to validate templates against a set of predefined business, architectural, and security rules.

AWS CloudFormation `validate-template` does not only check if you use proper JSON / YAML. It also performs other checks. E.g. it checks your Refs. But it doesn't check if your properties will work. The only way to really know is to create a stack and then run a few tests against the running stack (e.g. ssh in, send http request, ...)

#### LINTING

For linting (checking CloudFormation-template code for syntax/grammar correctness), you can use the `ValidateTemplate` API to check basic template structure, and the `CreateChangeSet` API to verify your Resource properties in more detail.

- Note that `ValidateTemplate` performs a much more thorough check than a simple JSON/YAML syntax checker- it validates correct Template Anatomy, correct syntax/usage of Intrinsic Functions, and correct resolution of all Ref values.

ValidateTemplate checks basic CloudFormation syntax, but doesn't verify your template's Resources against specific property schemas. For checking the structure of your template's Parameters,

## UNIT TESTING

Performing unit testing first requires an answer to the question: what is the smallest self-contained unit of functionality that can/should be tested? For CloudFormation, I believe that the smallest testable unit is the Resource. The official AWS Resource Types are supported/maintained by AWS (and are proprietary implementations anyway) so don't require any additional unit tests written by end-user developers.

However, your own Custom Resources could and should be unit-tested. This can be done using a suitable testing framework in the implementation's own language (e.g., for Lambda-backed Custom Resources, perhaps a library like lambda-tester would be a good starting point).

## 6.2 INTEGRATION TESTING

This is the most important and relevant type of testing for CloudFormation stacks (which mostly serve to tie various Resources together into an integrated application), and also the type that could use more refinement and best-practice development. Here are some initial ideas on how to integration-test CloudFormation code by actually creating/updating full stacks containing real AWS resources:

- Using a scripting language, perform a CloudFormation stack creation using the language's AWS SDK. Design the template to return Stack Outputs reflecting behavior that you want to test. After the stack is created by the scripting language, compare the stack outputs against expected values (and then optionally delete the stack afterwards in a cleanup process).
- Use `AWS::CloudFormation::WaitCondition` resources to represent successful tests/assertions, so that a successful stack creation indicates a successful integration-test run, and a failed stack creation indicates a failed integration-test run.

## 6.3 ACCEPTANCE TESTING

The objective of user acceptance testing is to present the current release to a testing team representing the final user base to determine if the project requirements and specification are met. When users can test the software earlier, they can spot conceptual weaknesses that have been introduced during the analysis phase, or clarify grey areas in the project requirements. By testing the software more frequently, users can identify functional implementation errors and user interface or application flow misconceptions earlier, lowering the cost and impact of correcting them. Flaws detected by user acceptance testing may be very difficult to detect by other means. The more often you conduct acceptance tests, the better for the project, since end users provide valuable feedback to development teams as requirements evolve.

However, like any other test practice, acceptance tests require resources to run the environment where the application to be tested will be deployed. AWS provides on-demand capacity as needed in a cost-effective way, which is also very appropriate for acceptance testing. Using some of the techniques described above, AWS enables complete automation of the process of provisioning new test environments and of disposing environments no longer needed. Test environments can be provided for certain times only, or continuously from the latest source code version, or for every major release. By deploying the acceptance test environment within Amazon VPC, internal users can transparently access the application to be tested. Besides, such an application can also be integrated with other production services inside the company, like LDAP, email servers, etc., offering a test environment to the end users that is even closer to the real and final production environment.

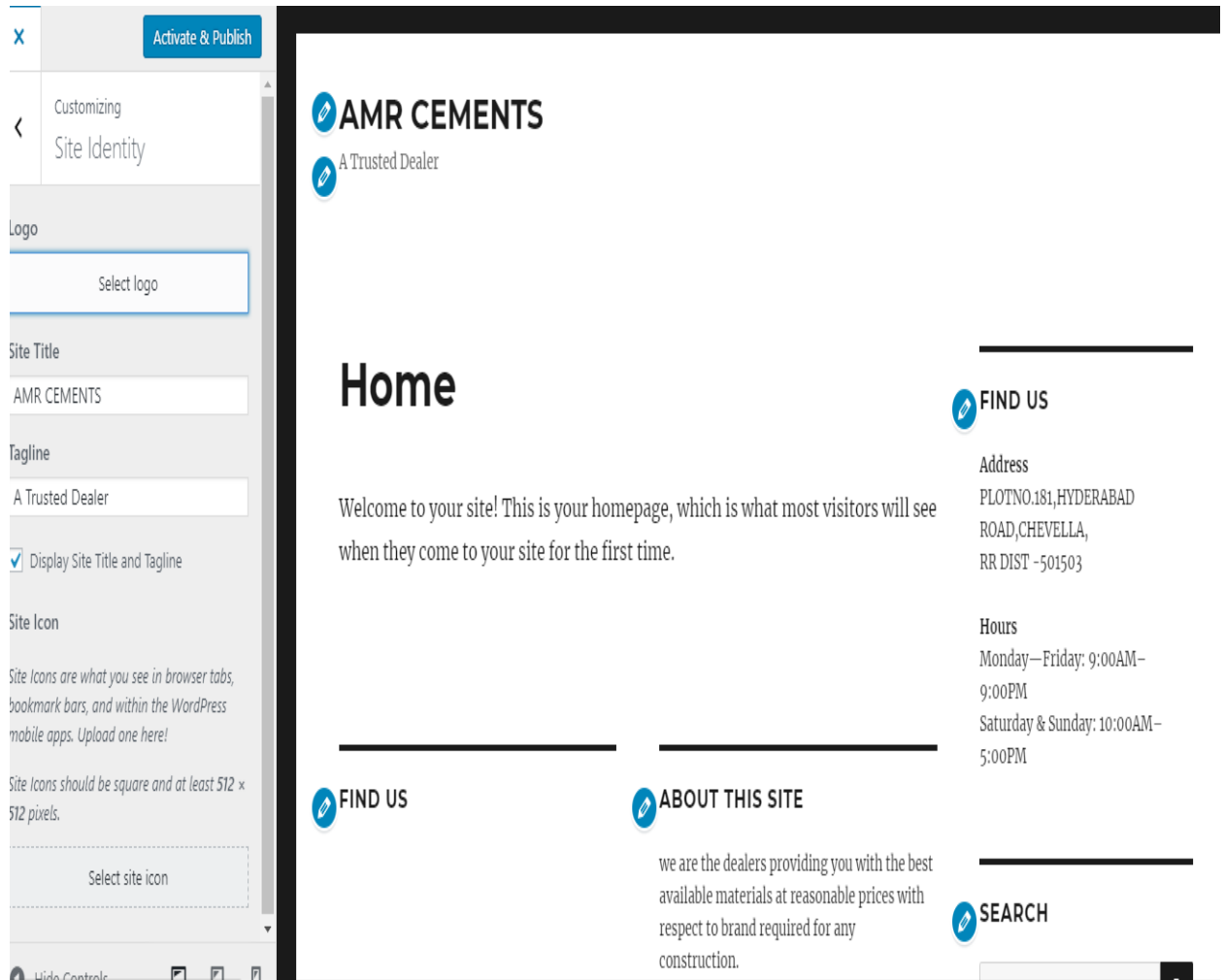
## 6.4 TEST CASES

Test Case Number	Test case	Input	Output	Success /Failure
1	Create cloudformation stack in aws management console	Create STACK	Initiated user Wpsite	success
2	Create a Stack	Specify template file	Uploaded template file	success
3	Specify stack details	Specify stack Name and parameters(ssh keyname,dbname,wpname,pwd,dbpwd)	Stack details updated	success
4	Configure stack options	Upload all stack details	Review wpsite	Failure

Test Case Number	Test Case	Input	Output	Success/Failure
5	Rollback	Uploaded again till stack details	Review wpsite	success
6	Launch the cloudformation stack	Click on running instance	Copy IP address	success
7	Browse using copied IP address	To install wordpress,Apache,PHP Configuration	Installed successfully	success
8	Login into Wordpress	Specify login details	Directed to wordpress	success


## CHAPTER 07

# OUTPUT SCREENS



## WORDPRESS SITE

## PUTTY SCREEN

 root@ip-172-31-15-159:/var/www/html

login as: ec2-user

Authenticating with public key "imported-openssh-key"

Last login: Sun Jul 7 15:58:41 2019 from 103.68.176.2

```
  _|  _|_ )  
  _| (  _ /  Amazon Linux AMI  
  __| \__|__|
```

<https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/>

16 package(s) needed for security, out of 22 available

Run "sudo yum update" to apply all updates.

[ec2-user@ip-172-31-15-159 ~]\$ sudo su -

Last login: Sun Jul 7 15:58:50 UTC 2019 on pts/0

[root@ip-172-31-15-159 ~]# cd /var/www/html

[root@ip-172-31-15-159 html]# ls

wordpress

[root@ip-172-31-15-159 html]# █



## **CHAPTER 08**

### **CONCLUSION**

AWS CloudFormation is a service that helps you model and set up your Amazon Web Services resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and AWS CloudFormation takes care of provisioning and configuring those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; AWS CloudFormation handles all of that simplifies infrastructure management.

It lets you focus on building amazing applications and services for users without having to spend lot of time manually configuring instances, software and database, less complexity and more security.

## CHAPTER 09

### FUTURE ENHANCEMENT

Infrastructure as Code has defined the last five years of systems engineering, and will likely define the next five. But as we became better and better at manipulating infrastructure with declarative languages, we started to look beyond JSON to find more full-fledged, dynamic programming languages to create and configure cloud resources. In many ways, this movement is mimicking the evolution of web development as web “sites” became web “applications” — in a few years, systems engineers will be coding infrastructure applications, not hard-coding declarative templates.

Amazon Web Services’ Infrastructure as Code (IaC) offering, CloudFormation, enables engineers to manage infrastructure through the use of declarative templates, utilizing the data format language, JSON. Templates can be checked into a source code management system, linted and validated using tools and IDE’s. Entire environments can be quickly duplicated, modified and deployed in quick time to keep pace with the rapid change ever present in today’s IT world. CloudFormation can also kick off bootstrapping through User Data, creating a seamless bridge into the operating systems where machine images and configuration management can take over. In addition, any infrastructure change can have an audit trail and proper process built around it, automated to reduce human error and disparate configurations.

At its heart, CloudFormation is simply declarative JSON. And that is a good thing. What is needed is not to make CloudFormation more dynamic, but rather to build tools which can dynamically generate templates and manage stacks. The expectation is that this practice will be the future. In some ways, the state of CloudFormation, and perhaps IaC in general today, is history repeating itself. CloudFormation is here to stay and is an excellent service offered by Amazon to implement effective IaC. However, in a few years, we will look back and “remember the time we had to hand-code a ton of JSON.

## CHAPTER 10

### BIBLIOGRAPHY

#### **Sample solution templates:**

- <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-applications-us-west-2.html>
- [https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/WordPress\\_Single\\_Instance.template](https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/WordPress_Single_Instance.template)
- Book:AWS CloudFormation User Guide Kindle Edition
- Aws Management Console: <https://aws.amazon.com/console/>
- Aws CloudFormation: <https://aws.amazon.com/cloudformation/>

