**Architecture of scalable and secure static web application in AWS:**

🡪To implement this project, we can use several configuration management tools like Ansible, AWS Cloud Formation, Terraform but here I will drive through using AWS Cloud and Ansible.

🡪 Using AWS Cloud Stack that is by utilizing AWS resources like AWS S3, AWS Cloud Front, AWS Certificate Manager.

**Development Part:**

* The very first step is to create an html file with the given code stub then save with .html file extension then open with any browser to test the webpage.

**Manual Deployment Part:**

* The deployment of this static web application by making use of AWS resources.
* We can implement this by making use of AWS EC2, AWS S3 and AWS … but among this it is more efficient to make use of Amazon S3 as it easy to implement and cost effective too.
* Now go to AWS then redirect to Amazon S3 then create a new S3 bucket to host the website with the unique name which should match exactly with the domain name.
* After bucket creation upload the html file to S3 bucket.
* Now for making the website as static there is an option available in S3 properties feature - static web hosting - enable it - then give the website home page file or html file then save the changes.
* Now editing the permissions of the bucket policy to make the bucket content as public.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": [

"s3:GetObject"

],

"Resource": [

"arn:aws:s3:::Bucket-Name/\*"

]

}

]

}

* Now when we investigate website hosting feature it gives us a link through which we can access the website, but the link will be not secured one so now to make it as secure static web application we need to attach an SSL certificate.
* So, we need to create a certificate using the certificate manager by selecting option and giving the domain name and DNS validation on our own domains.
* Now to create a cloud front distribution go to amazon CloudFront and create a distribution by giving the details of the domain link and choosing the option of redirecting the website from HTTP to HTTPS and provide the alternate domain name, choose SSL certificate.
* Once it is created it starts deploying the web application and gives the CloudFront URL link where we can access the secure static web application now update the domain name in DNS settings.

**Automated Deployment Process:**

* Now to automate the above process we need to make use of the AWS Cloud Formation templates.
* Ansible - Install then store AWS login credentials as environment variables file.
* Create a new project directory and

mkdir project\_sv

cd project\_sv

* Now create the ansible playbook file by making use of the YAML scripting.
* Now Create a new file named **projectsv\_playbook.yml** inside the project directory.
* The playbook defines the tasks to set up the AWS resources and deploy the static web application automatically.

- name: To Deploy Scalable and Secure Static Web Application in AWS

hosts: localhost

gather\_facts: false

vars:

s3\_bucket\_name: "xxbucketname"

cloudfront\_distribution\_id: "xx”

domain\_name: "xx.com"

region: "xxus-east-1"

tasks:

- name: To create S3 bucket

community.aws.s3\_bucket:

name: "{{ s3\_bucket\_name }}"

state: present

region: "{{ region }}"

acl: private

register: s3\_bucket\_result

- name: sv.html to S3 bucket

community.aws.s3:

bucket: "{{ s3\_bucket\_name }}"

object: " sv.html "

src: " sv.html"

mode: "public-read"

when: s3\_bucket\_result.changed

- name: CloudFront distribution creation

community.aws.cloudfront\_distribution:

state: present

aliases: "{{ domain\_name }}"

origin:

s3:

domain\_name: "{{ s3\_bucket\_name }}.s3.amazonaws.com"

origin\_path: ""

origin\_protocol\_policy: "http-only"

default\_cache\_behavior:

target\_origin\_id: "{{ s3\_bucket\_name }}"

viewer\_protocol\_policy: "redirect-to-https"

comment: "static Web Server distribution”

region: "{{ region }}"

register: cloudfront\_distribution\_result

- name: Display CloudFront distribution information

debug:

var: cloudfront\_distribution\_result.distribution\_id

* Create the html file sv.html inside the project directory that contains below code.

<html>

<head>

<title>Hello World</title>

</head>

<body>

<h1>Hello World!</h1>

</body>

</html>

* Now run the playbook using the command

ansible-playbook **projectsv\_playbook.yml**

* Now set up all the security related things in AWS like redirecting HTTP to HTTPS.Repeat the same steps performed in manual deployment.

1. Request an SSL/TLS Certificate using AWS Certificate Manager (ACM)
2. Attach the SSL/TLS Certificate to the CloudFront Distribution

* Validating the application as per the requirements which implemented in manual deployment.

Automated Testing using Ansible:

* In the ansible playbook by making use of the assert module.

- name: Validate S3 bucket content

assert:

that:

- s3\_bucket\_result is success

- "sv.html" in s3\_bucket\_result. object\_keys

fail\_msg: "Failed to validate S3 bucket content"

tags:

- validation

- name: Validate CloudFront distribution configuration

assert:

that:

- cloudfront\_distribution\_result is success

fail\_msg: "Failed to validate CloudFront distribution configuration.”

tags:

- validation

* Run the Yaml file ansible-playbook **projectsv\_playbook.yml** --tags validation
* To monitor and scaling this application we can make use of the AWS Amazon Cloud Watch , AWS Lambda, CloudFront access logs and S3 access logs, AWS auto scaling.

**Coding Task:**

import java.util.regex.\*;

public class CCsv{

public static boolean validcc(String cN)

{

String svpatteren = "^(?:(?:4|5|6)\\d{3})(?:-?\\d{4}){3}$";

Pattern sv\_P = Pattern.compile(svpatteren);

Matcher sv\_M = sv\_P.matcher(cN);

return sv\_M.matches();

}

public static void main(String[] args) {

String[] ccN = {

"6",

"4123456789123456",

"5123-4567-8912-3456",

"61234-567-8912-3456",

"5133-3367-8912-3456",

"5123--4567-8912-3456"

};

for (String cN : ccN) {

boolean isValid = validcc (cN);

System.out.println(cN + " is " + (isValid ? "valid" : "invalid"));

}

}

}

Output:

6 is invalid

4123456789123456 is valid

5123-4567-8912-3456 is valid

61234-567-8912-3456 is invalid

5133-3367-8912-3456 is valid

5123--4567-8912-3456 is invalid