**AWS AUTOMATION USING TERRAFORM:**

**Task**: Have to create/launch Application using Terraform

1. Create the key and security group which allow the port 80.

2. Launch EC2 instance.

3. In this Ec2 instance use the key and security group which we have created in step 1.

4. Launch one Volume (EBS) and mount that volume into /var/www/html

5. Developer have uploded the code into github repo also the repo has some images.

6. Copy the github repo code into /var/www/html

7. Create S3 bucket, and copy/deploy the images from github repo into the s3 bucket and change the permission to public readable.

8 Create a Cloudfront using s3 bucket(which contains images) and use the Cloudfront URL to update in code in /var/www/html

Screenshot_27

provider "aws" {

region = "ap-south-1"

profile = "kajal"

}

data "aws\_vpc" "selected" {

default = true

}

**/\* Generate a private key and encode it as pem \*/**

resource "tls\_private\_key" "example" {

algorithm = "RSA"

}

resource "local\_file" "private-key" {

content = "tls\_private\_key.key.private\_key\_pem"

filename = "mykey.pem"

file\_permission = 0400

}

resource "aws\_key\_pair" "key-pair" {

key\_name = "mykey"

public\_key = tls\_private\_key.example.public\_key\_openssh

}

**/\* Create security group and allow HTTP, SSH and ICMP protocols \*/**

resource "aws\_security\_group" "allow\_tls" {

name = "allow\_tls1"

description = "Allow\_tls"

vpc\_id = "${data.aws\_vpc.selected.id}"

ingress {

description = "http"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "ssh"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "ping-icmp"

from\_port = -1

to\_port = -1

protocol = "icmp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "allow\_tls1"

}

}

**/\* Launch an OS, attach the key generated and the security groups created. SSH into the OS and install the webserver and the required SDK's \*/**

resource "aws\_instance" "web" {

ami = "ami-0447a12f28fddb066"

instance\_type = "t2.micro"

key\_name = aws\_key\_pair.key-pair.key\_name

security\_groups = [ aws\_security\_group.allow\_tls.name ]

connection {

type = "ssh"

user = "ec2-user"

private\_key = tls\_private\_key.example.private\_key\_pem

host = aws\_instance.web.public\_ip

}

provisioner "remote-exec" {

inline = [

"sudo yum install httpd php git -y",

"sudo systemctl restart httpd",

"sudo systemctl enable httpd",

]

}

tags = {

Name = "lwos1"

}

}

**/\* Fetch the availability zone of the instance and create an EBS volume in the same zone. \*/**

resource "aws\_ebs\_volume" "lw\_ebs" {

availability\_zone = aws\_instance.web.availability\_zone

size = 1

tags = {

Name = "lw\_ebs"

}

}

**/\* Attach the volume created to the instance \*/**

resource "aws\_volume\_attachment" "ebs\_att" {

device\_name = "/dev/sdh"

volume\_id = "${aws\_ebs\_volume.lw\_ebs.id}"

instance\_id = "${aws\_instance.web.id}"

force\_detach = true

}

output "myos\_ip" {

value = aws\_instance.web.public\_ip

}

***/\* [Optional]\*/***

resource "null\_resource" "nullocal2" {

provisioner "local-exec" {

command = "echo ${aws\_instance.web.public\_ip} > publicip.txt"

}

}

**/\* Provisioners are used to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.\*/**

**/\* Provisioners need a null-resource that is a do-nothing container for the actions taken by a provisioner. \*/**

resource "null\_resource" "nullremote3" {

**/\* Tells Terraform that EBS volume must be formatted, mounted and store data only after the created volume has been attched to the instance. \*/**

depends\_on = [

aws\_volume\_attachment.ebs\_att,

]

connection {

type = "ssh"

user = "ec2-user"

private\_key = tls\_private\_key.example.private\_key\_pem

host = aws\_instance.web.public\_ip

}

provisioner "remote-exec" {

inline = [

"sudo mkfs.ext4 /dev/xvdh",

"sudo mount /dev/xvdh /var/www/html",

"sudo rm -rf /var/www/html/\*",

"sudo git clone https://github.com/kajal1706043/multi\_cloudTask1.git /var/www/html"

]

}

}

**/\*Tells Terraform that the IP address must be viewed onto the browser only after the instance is completely ready with the webpage. \*/**

resource "null\_resource" "nulllocal1" {

depends\_on = [

null\_resource.nullremote3,

]

provisioner "local-exec" {

command = "start chrome ${aws\_instance.web.public\_ip}"

}

}

**/\*Create an S3 bucket and grant public access to it \*/**

resource "aws\_s3\_bucket" "b" {

bucket = "tsk1bucket"

acl = "public-read"

tags = {

Name = "mybucket"

}

}

**/\* Deploy an image into the bucket from Github. \*/**

resource "aws\_s3\_bucket\_object" "deployimage" {

bucket = aws\_s3\_bucket.b.bucket

key = "cloudtask1.jpg"

source = "git\_image/Hybrid-Cloud.jpg"

acl = "public-read"

}

**/\* null-resources are the first to be executed by Terraform. Thus, the image on github is first download onto the local machine\*/**

resource "null\_resource" "nulllocal4" {

provisioner "local-exec" {

command = "git clone https://github.com/kajal1706043/task1\_s3.git git\_image"

}

**/\* To remove the image from the local system when the infrastructure is destroyed \*/**

provisioner "local-exec" {

when = destroy

command = "rmdir /s /q git\_image"

}

}

resource "aws\_cloudfront\_origin\_access\_identity" "origin\_access\_identity" {

comment = "Some comment"

}

**#Create a CloudFront Distribution with the created S3 bucket as Origin**

locals {

s3\_origin\_id = "myS3Origin"

image-url = "${aws\_cloudfront\_distribution.s3\_distribution.domain\_name}/${aws\_s3\_bucket\_object.deployimage.key}"

}

resource "aws\_cloudfront\_distribution" "s3\_distribution" {

origin {

domain\_name = aws\_s3\_bucket.b.bucket\_domain\_name

origin\_id = "${local.s3\_origin\_id}"

s3\_origin\_config {

origin\_access\_identity = "${aws\_cloudfront\_origin\_access\_identity.origin\_access\_identity.cloudfront\_access\_identity\_path}"

}

}

default\_cache\_behavior {

allowed\_methods = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]

cached\_methods = ["GET", "HEAD"]

target\_origin\_id = "${local.s3\_origin\_id}"

forwarded\_values {

query\_string = false

cookies {

forward = "none"

}

}

viewer\_protocol\_policy = "allow-all"

}

enabled = true

restrictions {

geo\_restriction {

restriction\_type = "none"

}

}

viewer\_certificate {

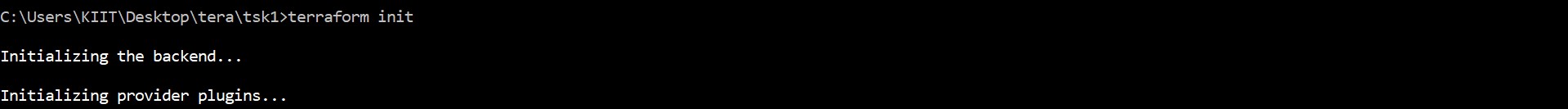
cloudfront\_default\_certificate = true

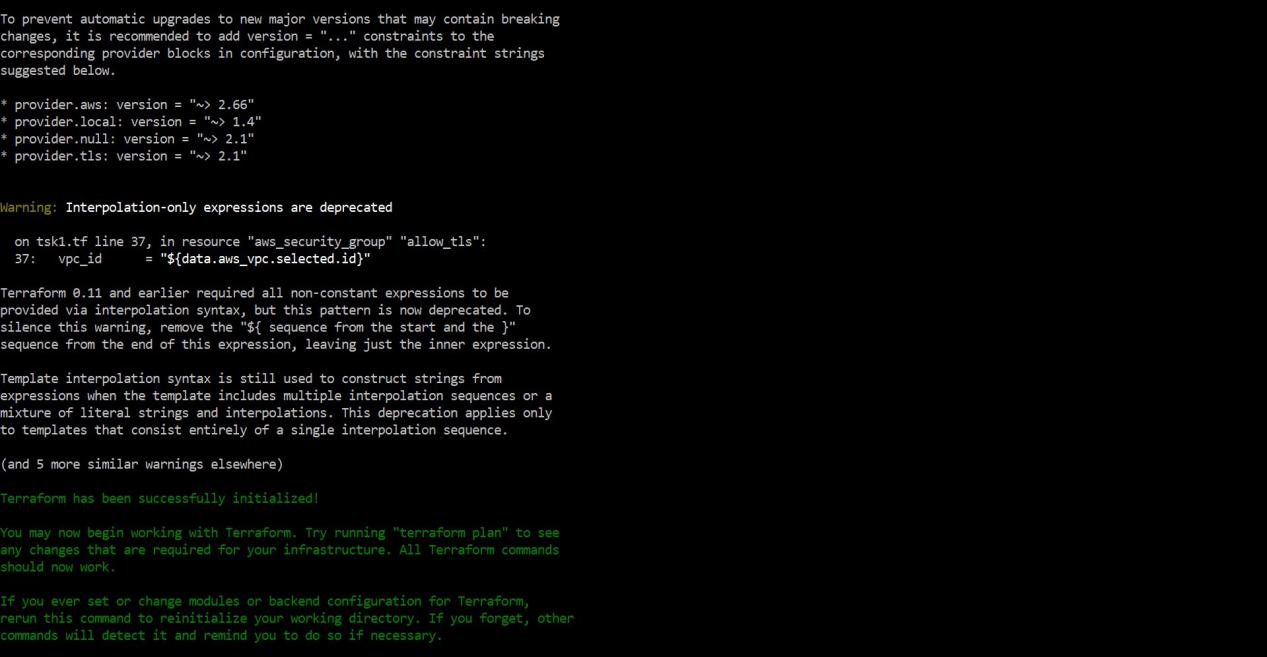
}

}

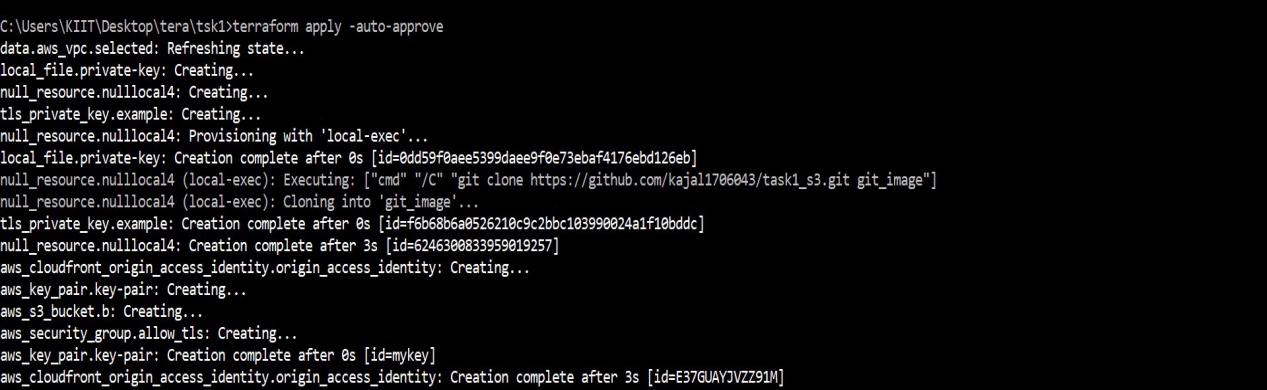
Commands:

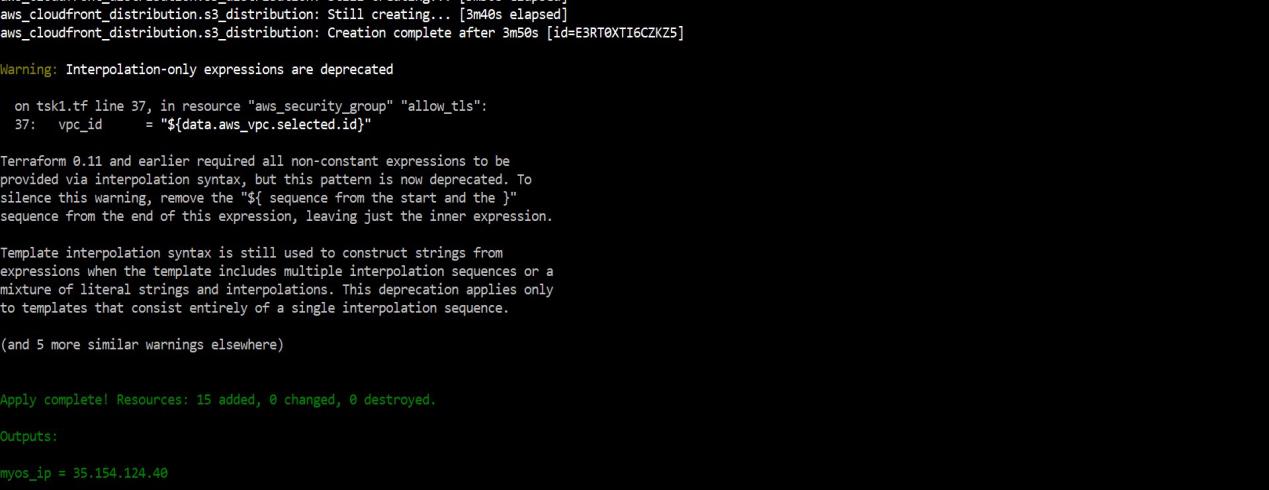
* terraform init //to download the necessary plugins.



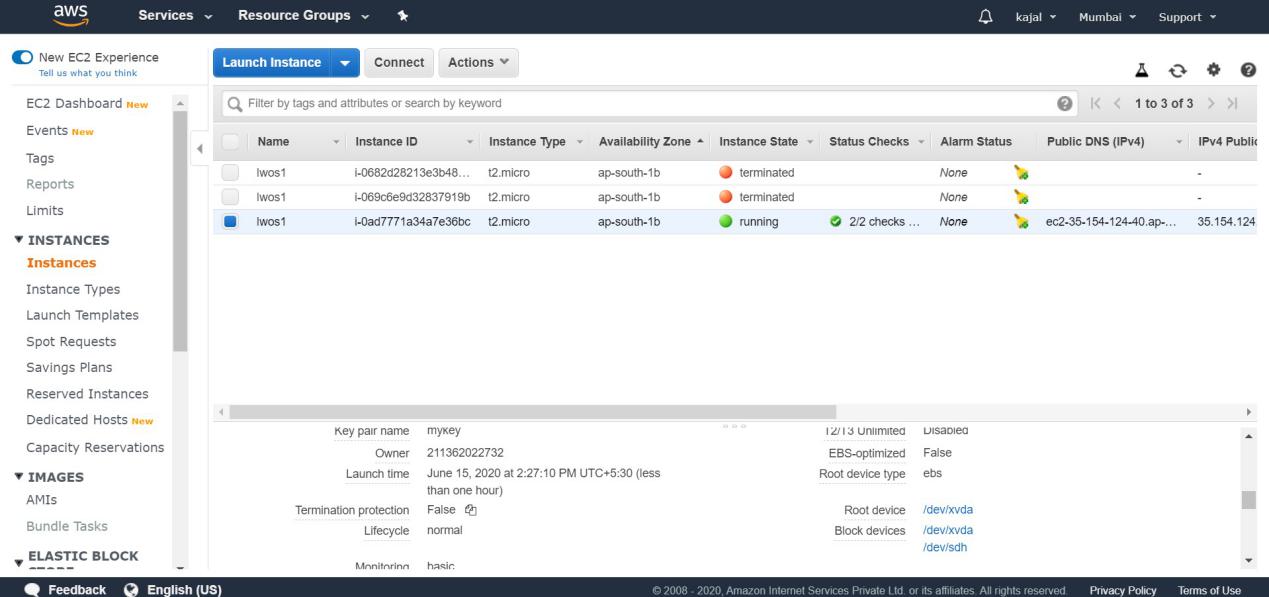


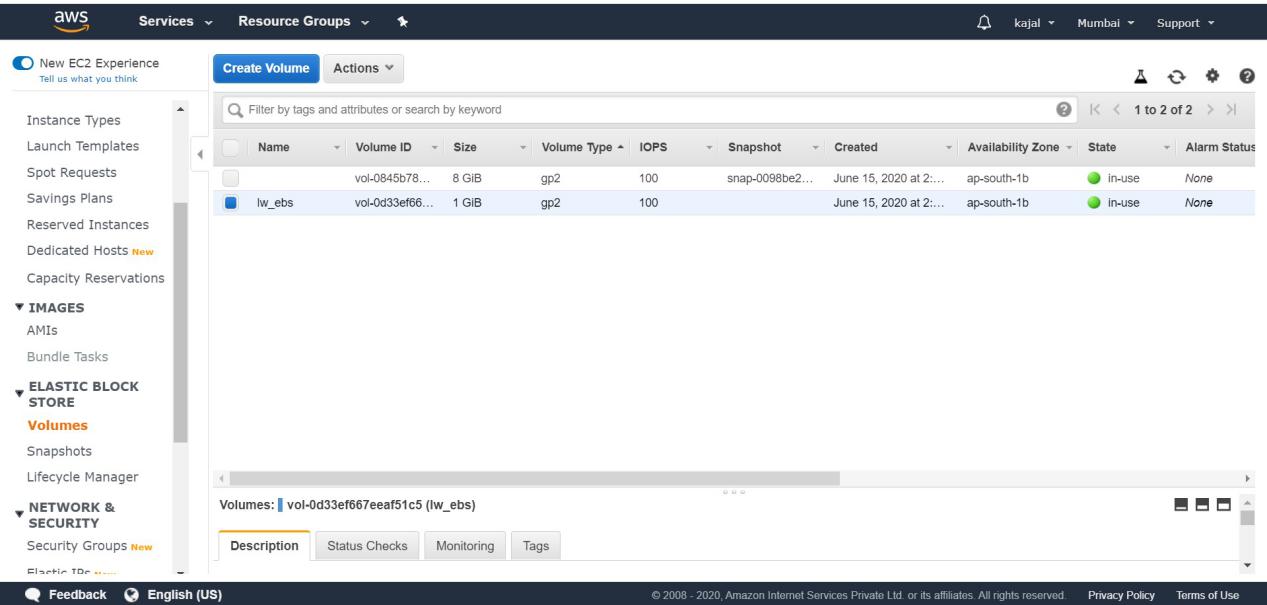
* Terraform apply -auto-approve //to build the infrastructure.

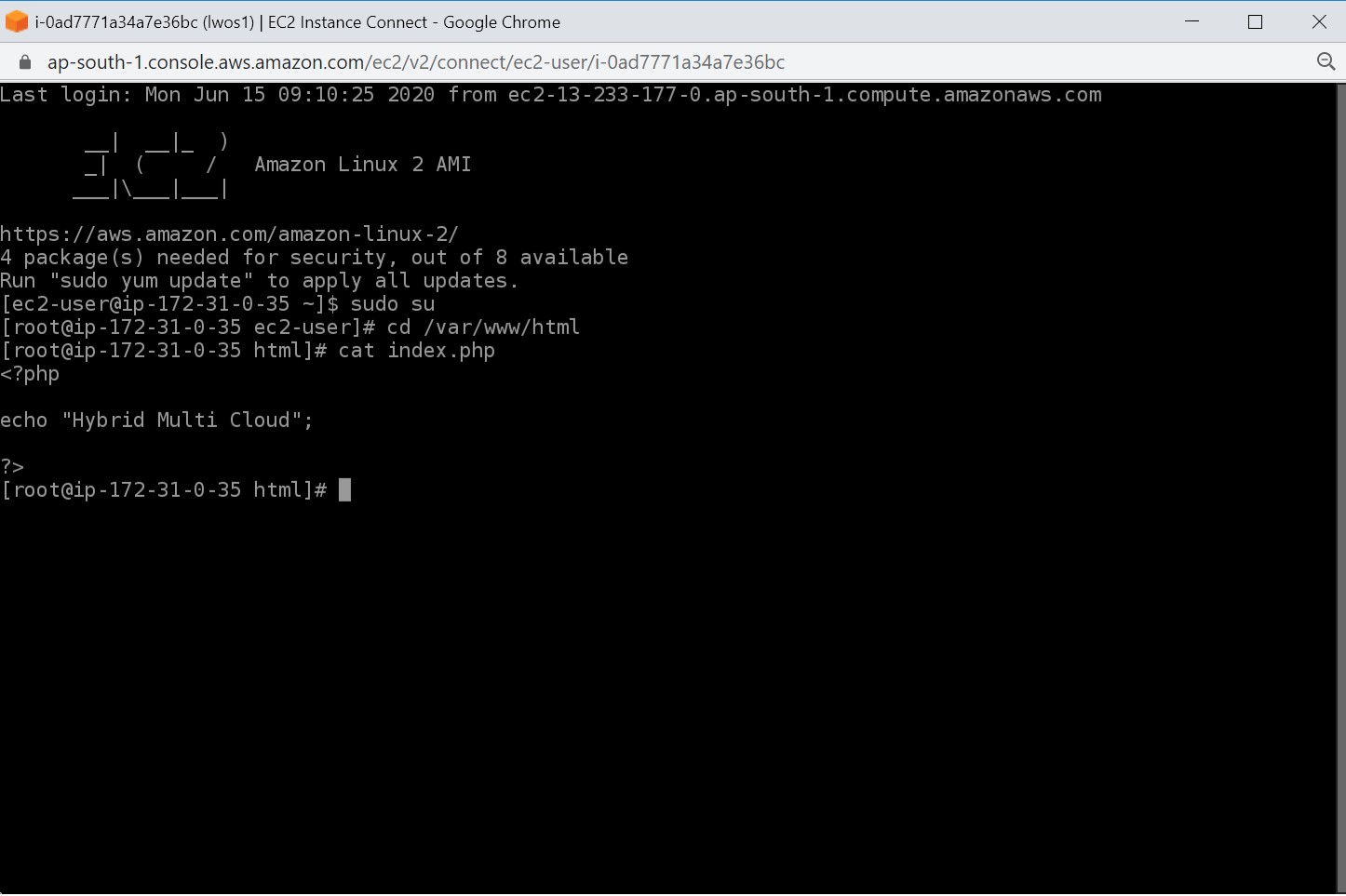


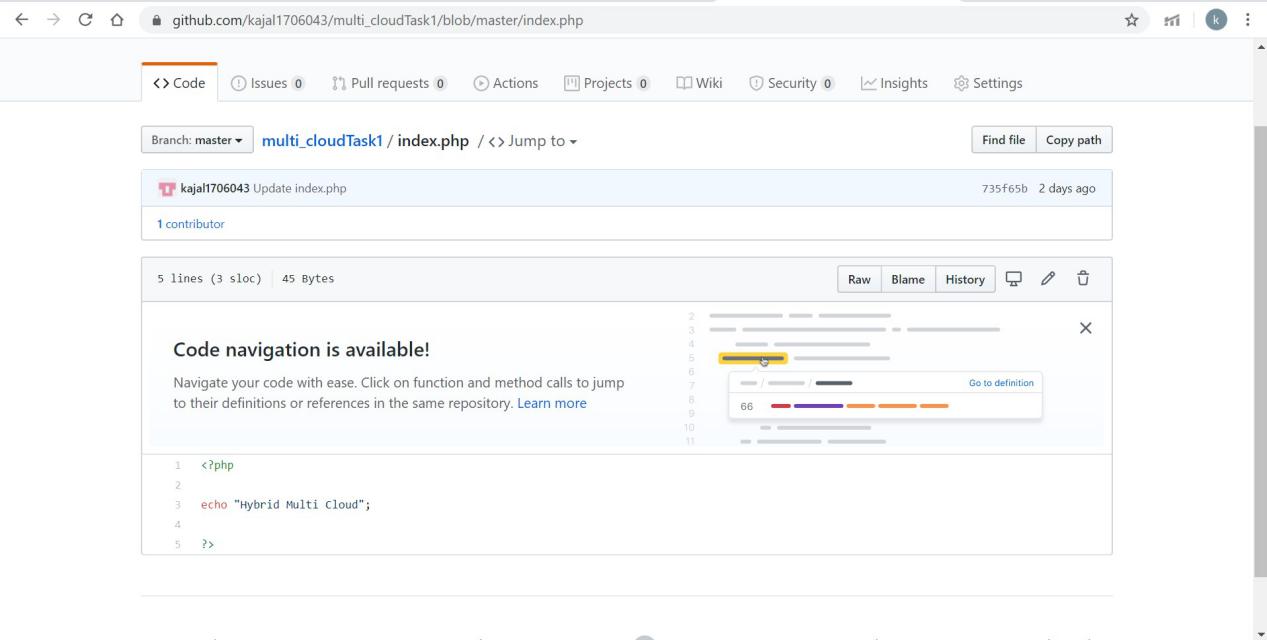


**The infrastructure is built:**

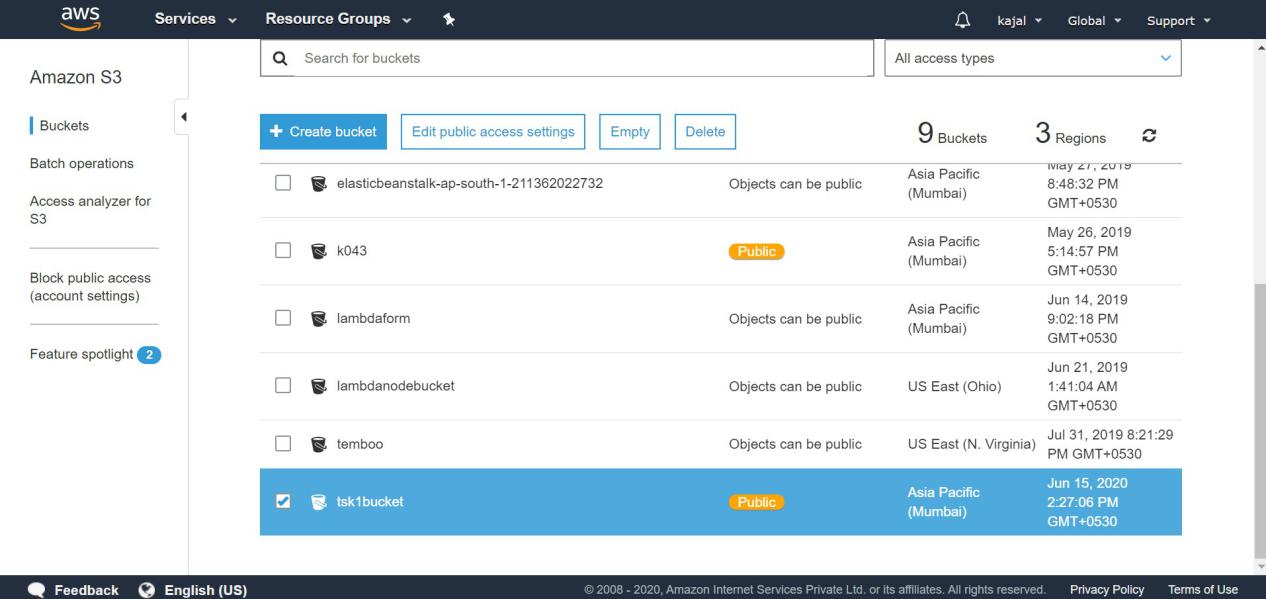


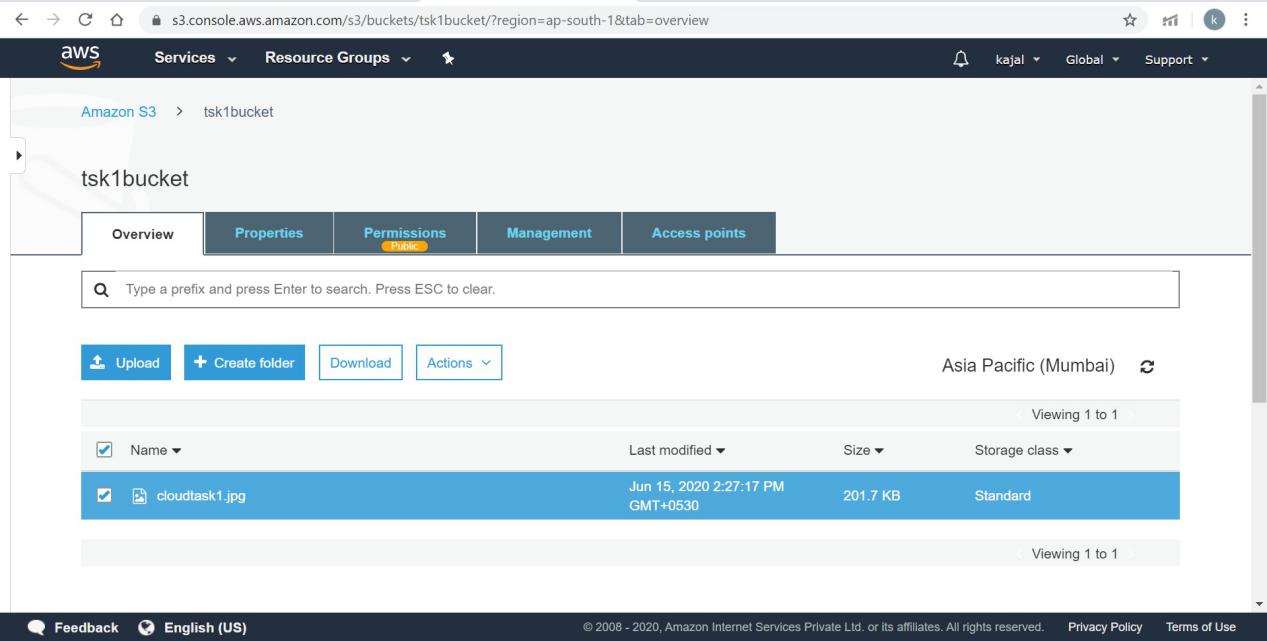


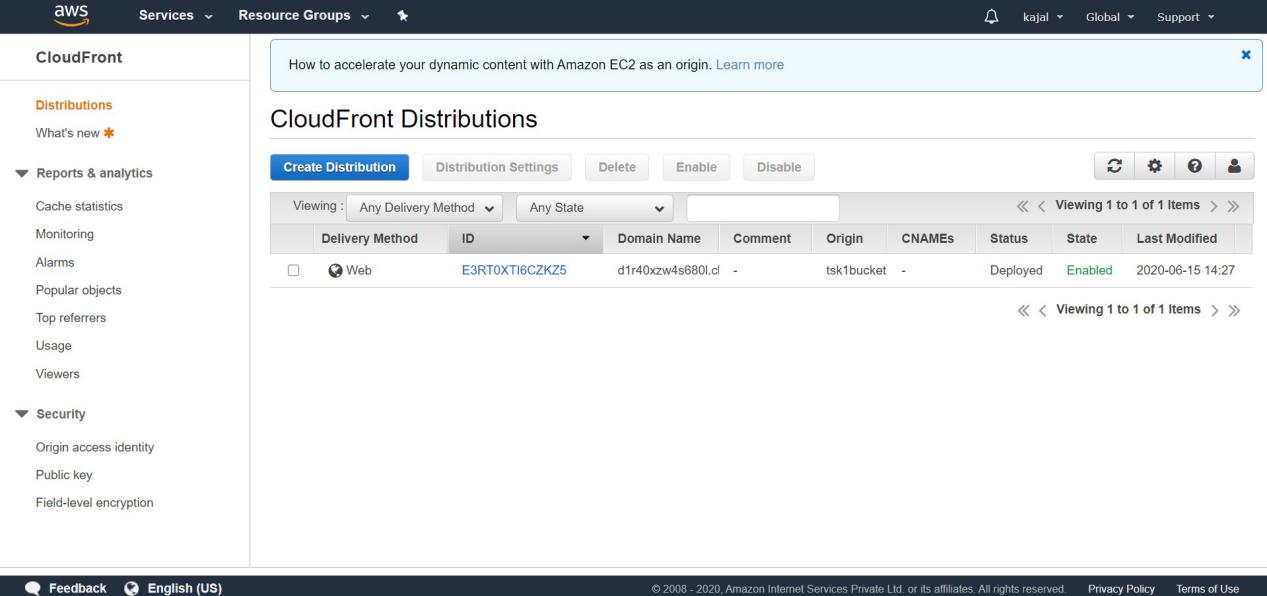


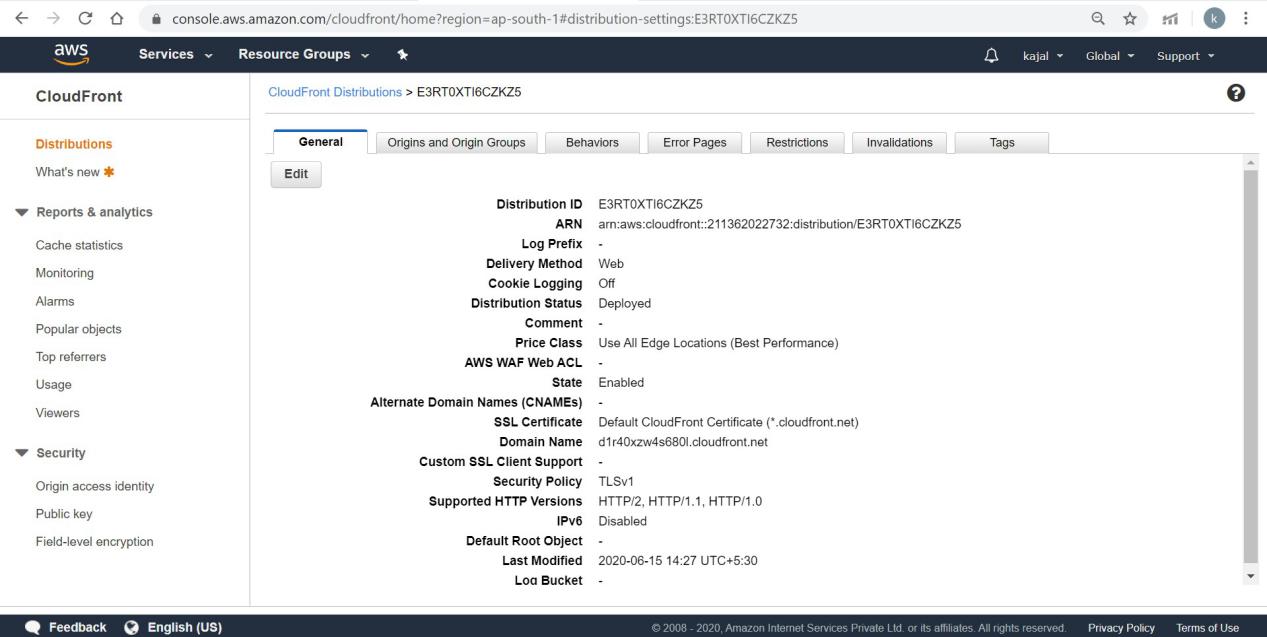


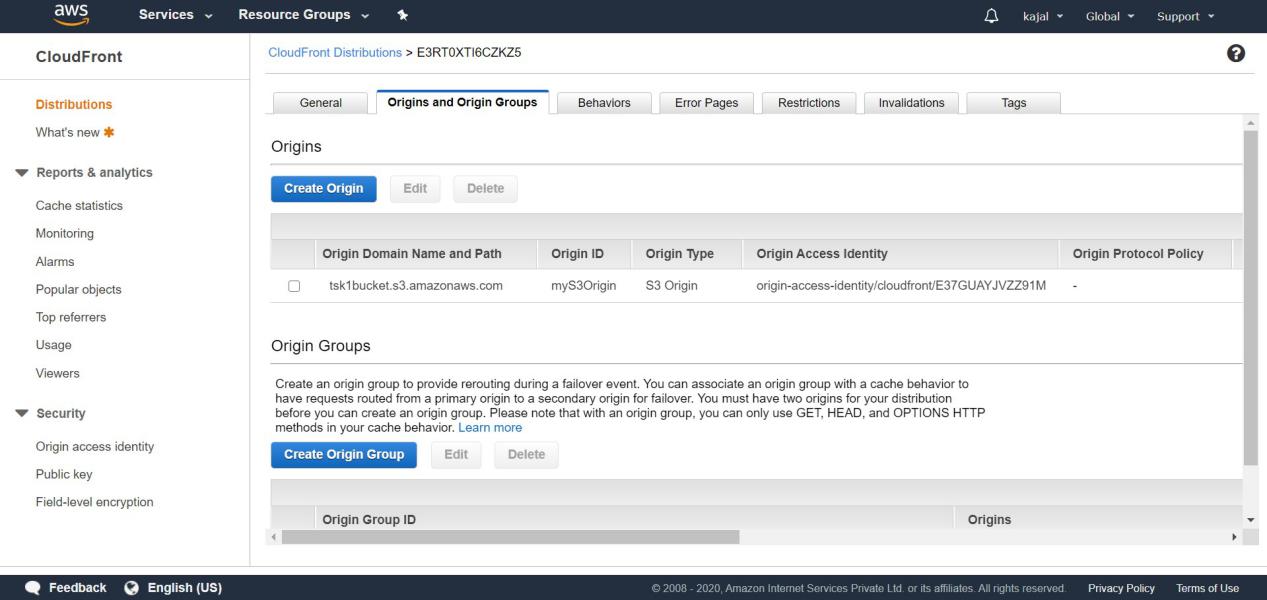












* To destroy the infrastructure:



