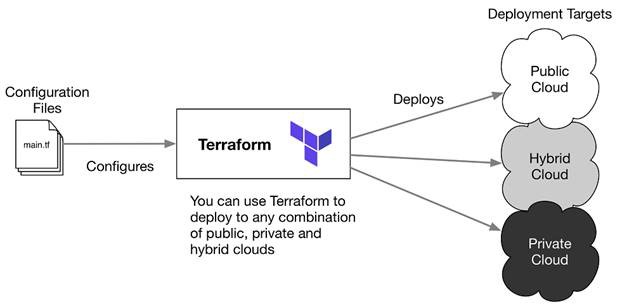


Considering you are here, you must be having a slight idea about **Terraform**and its uses. So, if you want to build something good using it, you’ve landed on the right page.😉

It is always good, to begin with the basics.😬

[**Terraform**](https://www.terraform.io/intro/index.html)is an intelligent tool for building, changing, and versioning infrastructure safely and efficiently. The simplest explanation of Terraform can be seen in the image below. We just have to write the [**Infrastructure as Code (IAC)**](https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/) in the configuration files having **.tf** as an extension to build the whole infrastructure in just one click!



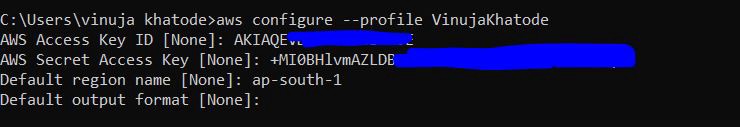
*For this practical, we will need****Terraform setup****,****Account on AWS Console****,****AWS CLI setup****, and****little curiosity to learn****something new.😄*Here we are launching an infrastructure to deploy websites.

***Automating the Infrastructure setup using Terraform and AWS Cloud.***

Here is the detailed problem statement:

1. Create the **key**and **security group**which allows the port 80 and 22.
2. Launch **EC2 instance**. In this EC2 instance use the key and security group created above.
3. Launch one **Volume (EBS)** and mount that volume into /var/www/html.  
   4. **Copy**the GitHub repo code that is uploaded by the developer into /var/www/html  
   5. Create an **S3 bucket**, and copy the images from GitHub repo into this s3 bucket and change the permission to public readable.  
   6. Create a **Cloudfront using s3 bucket**(which contains images) and use the Cloudfront URL to update in code in /var/www/html

The first thing to do is, [Download the **AWS CLI**](https://s3.amazonaws.com/aws-cli/AWSCLI64PY3.msi) and finish the installation. If done with this installation go the AWS Console, and under services go to the IAM and Add a new user. After adding the user, we will get access credentials, save that CSV file of credentials. Now in command prompt write the command as below: (put “*json*” in output format)\*

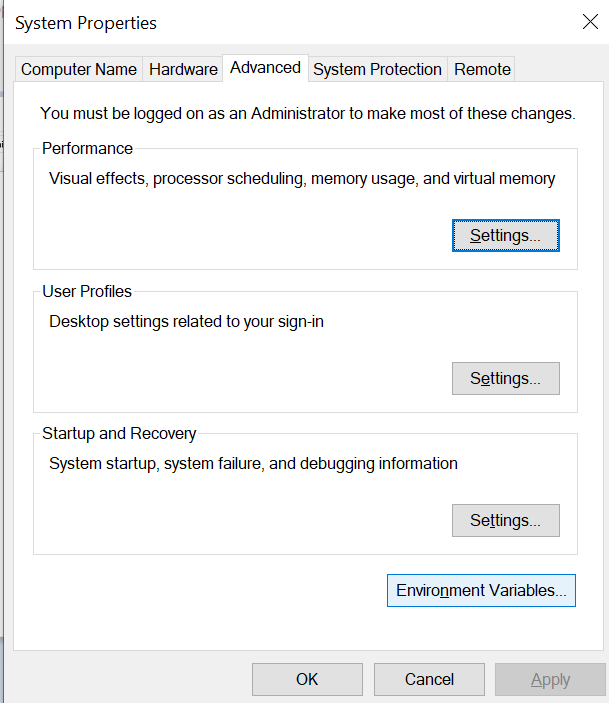


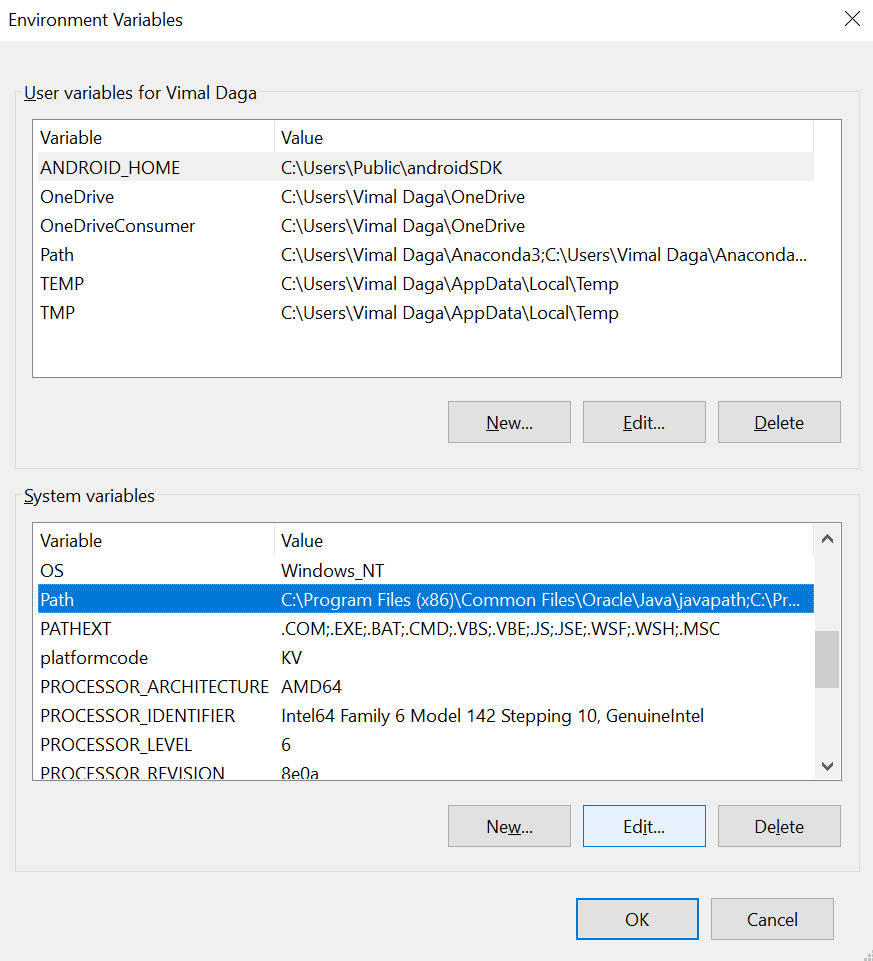
Configure profile

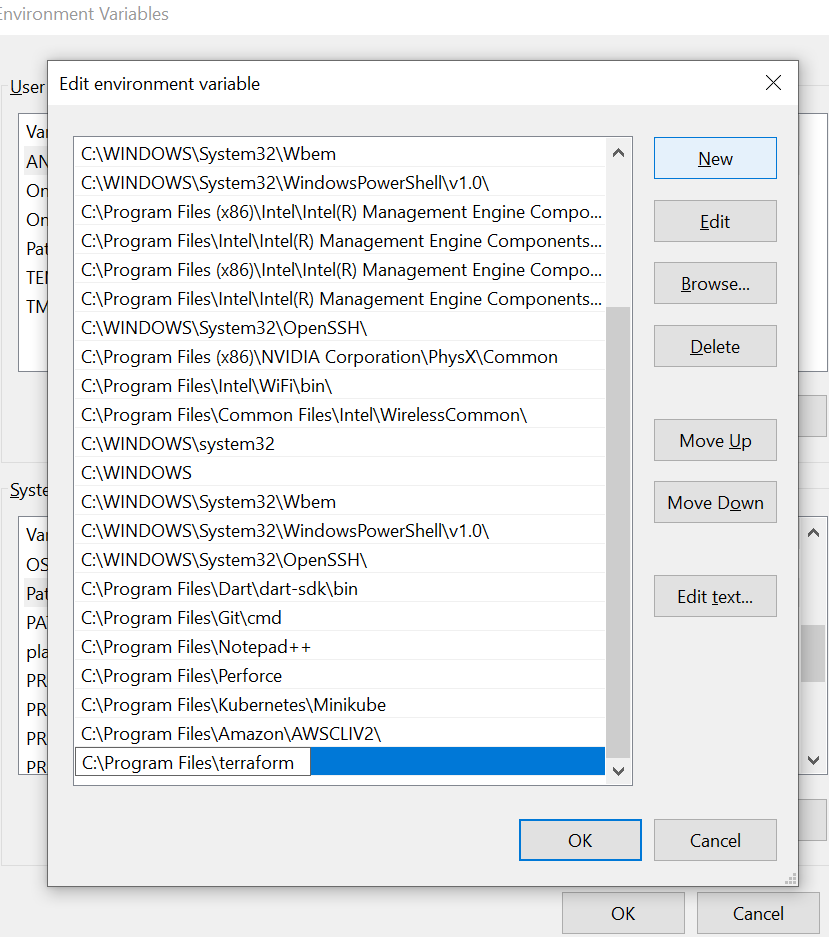
https://miro.medium.com/max/517/1*P9K811hVs3LiVCWVBCex4A.jpeg

Lists profile

* *Done with the profile configuration!!*
* Now, Download the **[Terraform](https://www.terraform.io/downloads.html" \t "_blank)**and extract it in a folder. **Copy the path** of the folder wherever you have extracted and add that path to your system’s **environment variables.**







That's all for the Terraform setup, for confirmation run this command in the command prompt.

https://miro.medium.com/max/747/1*6Y3NUvynwkm9SWzpGnLQWA.png

*Yesss, your Terraform setup is ready!!!*

**Time to do some serious task now. 😁**

**Create a workspace** for this practical. For reference see the above picture. There you can see the **Task1 folder**. That is the folder where we will be saving all the files and doing all the work!

*Yeah I know that's not the serious work, It starts now👇*

* cd into your workspace, and **start notepad** by

notepad Task1.tf

* Here Task1 is the name of the configuration file that we will be using to write code.
* First, we have to tell Terraform that we are using **AWS provider**. The way to do this is:

provider "aws"{  
 region="ap-south-1"  
 profile = "VinujaKhatode"  
}

* So, the next step is to **generate keys**and creating security groups using Terraform. For the same, the following code can be used:

**//Creating KEY**resource "tls\_private\_key" "tls\_key" {  
 algorithm = "RSA"  
}**//Generating Key-Value Pair**resource "aws\_key\_pair" "generated\_key" {  
 key\_name = "Task1keyvin"  
 public\_key ="${tls\_private\_key.tls\_key.public\_key\_openssh}"  
   
 depends\_on = [  
 tls\_private\_key.tls\_key  
 ]  
}**//Saving Private KEY PEM File**resource "local\_file" "key-file" {  
 content = "${tls\_private\_key.tls\_key.private\_key\_pem}"  
 filename = "Task1keyvin.pem" depends\_on = [  
 tls\_private\_key.tls\_key,  
 aws\_key\_pair.generated\_key  
 ]  
}

* The above code will generate the key named Task1keyvin.pem in the same folder.
* ***In order to execute and validate, you have to write code till launching the instance at once! So that you won’t have to face the error of key generation.***
* Let's move forward to the**Security group.** Here we will allow SSH Port and HTTP Port and also the Localhost. Naming this Security group as Task1sec.

**//Creating Security Group**resource "aws\_security\_group" "Task1sec" {  
 name = "Task1sec"  
 description = "Security Group for Task1 SSH and HTTPD"**//Adding Rules to Security Group** ingress {  
 description = "SSH Port"  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 ingress {  
 description = "HTTP Port"  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 ingress {  
 description = "Localhost"  
 from\_port = 8080  
 to\_port = 8080  
 protocol = "tcp"  
 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 = "Task1sec"  
 }  
}

* After configuring security groups, the next step is to **launch the EC2 instance**using the above-created security group and keys. We have to give **ami-id and the instance type** to launch that instance.

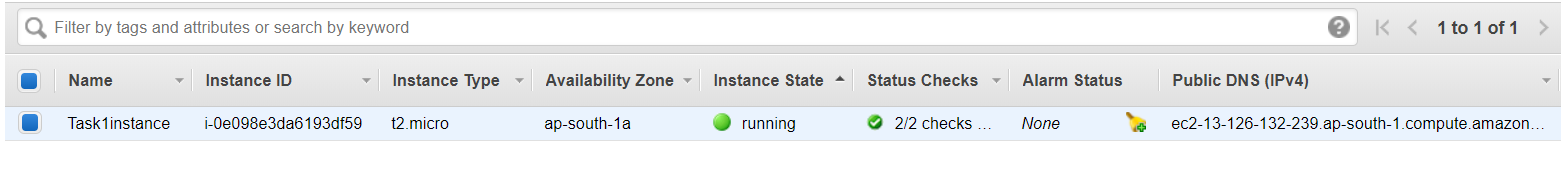
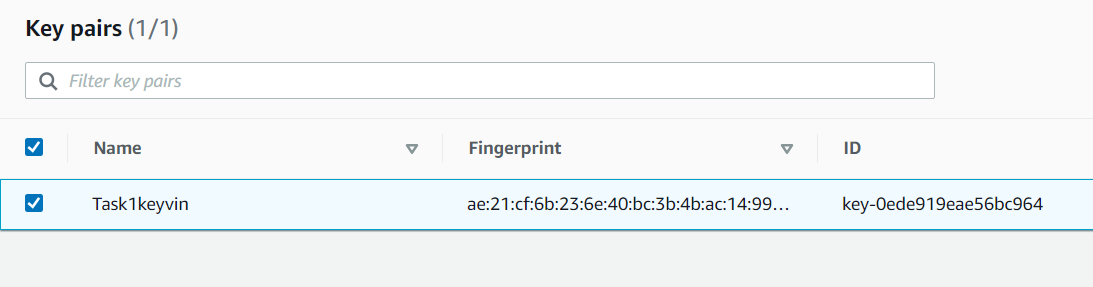
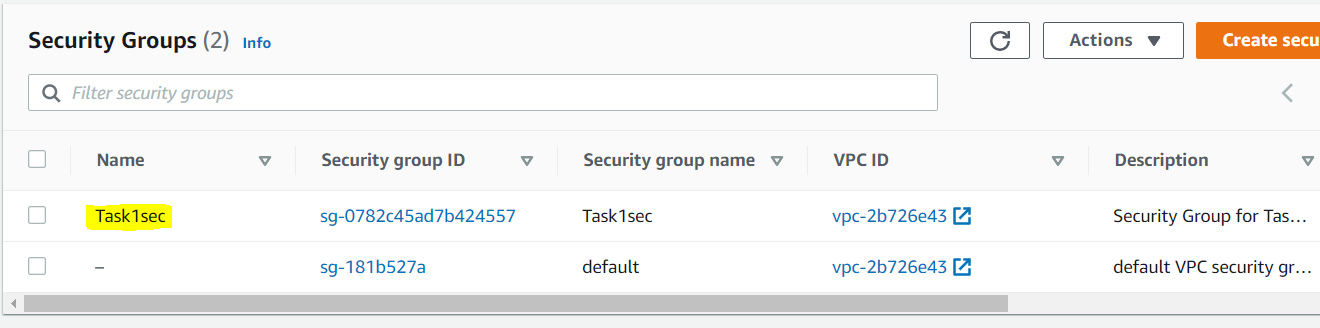
**// Creating instance with above created key and security group**resource "aws\_instance" "Task1instance" {  
 ami = "ami-005956c5f0f757d37"  
 instance\_type = "t2.micro"  
 key\_name = "${aws\_key\_pair.generated\_key.key\_name}"  
 security\_groups = ["${aws\_security\_group.Task1sec.name}"]  
 tags = {  
 Name = "Task1instance"   
 }  
}

* If we run this tf file now, this code will launch the instance named Task1instance with the private key Task1keyvin and Task1sec as a Security group.
* So to run this code, **Save** the Task1.tf file and Open Command Prompt and give commands as follow:

terraform init

* You always have to run this command **every time you make changes**in the configuration file. It installs all the**required plugins**. After the successful completion of the above command, the next command is:

terraform apply -auto-approve

* This command will apply all the code written inside the tf file. The attribute ***-auto-approve***is optional. It basically gives approval for further procedures.
* 
* 
* 

*Hope you are doing good till now! ✨*

*Okay, now moving towards the main steps.*😵

* In this step, we have to **launch one volume**and mount it to the var/www/html folder. Here we will create a volume of 1GB.
* **// Creating new EBS Volume and attachin it to the above created instance**resource "aws\_ebs\_volume" "ebs1" {  
   availability\_zone = aws\_instance.Task1instance.availability\_zone  
   size = 1  
   tags = {  
   Name = "Task1ebs"  
   }  
  }**// To attach the Volume created**resource "aws\_volume\_attachment" "ebs\_attach" {  
   device\_name = "/dev/sdh"  
   volume\_id = "${aws\_ebs\_volume.ebs1.id}"  
   instance\_id = "${aws\_instance.Task1instance.id}"  
   force\_detach = true  
  }**// Outputing the IP of the instance**output "myos\_ip" {  
   value = aws\_instance.Task1instance.public\_ip  
  }
* Okay, so in order to use and mount the volume in the instance, we have to **remote login to the ec2** instance and then first **partition, format, and then mount the volume.**
* Add the below code to your configuration file:
* **// In order to use Volume partition, format nad mounting is necessary**resource "null\_resource" "nullremote" { depends\_on = [  
   aws\_volume\_attachment.ebs\_attach,  
   aws\_security\_group.Task1sec,  
   aws\_key\_pair.generated\_key   
   ]connection {  
   type = "ssh"  
   user = "ec2-user"  
   private\_key = file("C:/Users/vinuja khatode/VWorkspace/Terraform/Task1/Task1keyvin.pem")  
   host = aws\_instance.Task1instance.public\_ip  
   }provisioner "remote-exec" {  
   inline = [  
   "sudo yum install httpd php git -y",  
   "sudo service httpd start",  
   "sudo chkconfig httpd on",   
   "sudo mkfs.ext4 /dev/xvdh",  
   "sudo mount /dev/xvdh /var/www/html",  
   "sudo rm -rf /var/www/html/\*",  
   "sudo git clone <https://github.com/vinujakhatode/Webserver-Terraform-AWS.git> /var/www/html/"  
   ]  
   }  
  }
* We used a “**connection**” block to remote log in and **provisioner**block to execute commands inside the instance. The above code will make **partitions, format, and mount the volume**to html folder and will also clone the GitHub repository there. It will also install **httpd**, **PHP,**and **git**services.
* ***Here is a fact about git command, it only clones into an empty folder, so make our operation successful we are forcefully removing the contents of the HTML folder.***

*Till now, you can launch an instance with web hosting environment and volume attached to it in just one click!✨*

* **So the next step is to create an S3 Bucket:**
* **// Creating S3 bucket**resource "aws\_s3\_bucket" "task1bucketvinuja00vinuja00" {  
   bucket = "task1bucketvinuja00vinuja00"  
   acl = "private"  
   tags = {  
   Name = "task1bucketvinuja00vinuja00"  
   }  
  }**// Allow Public Access**resource "aws\_s3\_bucket\_public\_access\_block" "S3PublicAccess" {  
   bucket = "${aws\_s3\_bucket.task1bucketvinuja00vinuja00.id}"  
   block\_public\_acls = true  
   block\_public\_policy = true  
   restrict\_public\_buckets = true  
  }
* This script will create the S3 bucket named **task1bucketvinuja00vinuja00.**Make sure whatever name you choose must be in **small letters and unique** i.e there should not exist any other bucket with the same name. Here we are also allowing public access.
* Now we have to fetch the**images from S3 using CloudFront.** For that, we first have to put images into the S3.
* **// Uploading files to S3 bucket**

**// Uploading files to S3 bucket**resource "aws\_s3\_bucket\_object" "bucketObject" {  
 for\_each = fileset("C:/Users/vinuja khatode/Desktop/girly/assets", "\*\*/\*.jpg")  
 bucket = "${aws\_s3\_bucket.task1bucketvinuja00vinuja00.bucket}"  
 key = each.value  
 source = "C:/Users/vinuja khatode/Desktop/girly/assets/${each.value}"  
 content\_type = "image/jpg"  
}

The images have been uploaded to the S3 bucket.

**//Creating Cloudfront to access images from S3**locals {  
 s3\_origin\_id = "S3Origin"  
}**// Creating Origin Access Identity for CloudFront**resource "aws\_cloudfront\_origin\_access\_identity" "origin\_access\_identity" {  
 comment = "task1bucketvinuja00vinuja00"  
}resource "aws\_cloudfront\_distribution" "Task1CF" {origin {  
 domain\_name = "${aws\_s3\_bucket.task1bucketvinuja00vinuja00.bucket\_regional\_domain\_name}"  
 origin\_id = "${local.s3\_origin\_id}"  
 s3\_origin\_config {  
 origin\_access\_identity = "${aws\_cloudfront\_origin\_access\_identity.origin\_access\_identity.cloudfront\_access\_identity\_path}"  
 }  
}enabled = true  
 is\_ipv6\_enabled = true  
 comment = "accessforTask1"  
 default\_cache\_behavior {  
 allowed\_methods = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]  
 cached\_methods = ["GET", "HEAD"]  
 target\_origin\_id = "${local.s3\_origin\_id}"  
 forwarded\_values {  
 query\_string = false  
 cookies {  
 forward = "none"  
 }  
 }  
 viewer\_protocol\_policy = "allow-all"  
 min\_ttl = 0  
 default\_ttl = 3600  
 max\_ttl = 86400  
 }  
**// Cache behavior with precedence 0**ordered\_cache\_behavior {  
 path\_pattern = "/content/immutable/\*"  
 allowed\_methods = ["GET", "HEAD", "OPTIONS"]  
 cached\_methods = ["GET", "HEAD", "OPTIONS"]  
 target\_origin\_id = "${local.s3\_origin\_id}"  
 forwarded\_values {  
 query\_string = false  
 headers = ["Origin"]  
 cookies {  
 forward = "none"  
 }  
 }min\_ttl = 0  
 default\_ttl = 86400  
 max\_ttl = 31536000  
 compress = true  
 viewer\_protocol\_policy = "redirect-to-https"  
}  
**// Cache behavior with precedence 1**ordered\_cache\_behavior {  
 path\_pattern = "/content/\*"  
 allowed\_methods = ["GET", "HEAD", "OPTIONS"]  
 cached\_methods = ["GET", "HEAD"]  
 target\_origin\_id = "${local.s3\_origin\_id}"  
 forwarded\_values {  
 query\_string = false  
 cookies {  
 forward = "none"  
 }  
 }  
min\_ttl = 0  
default\_ttl = 3600  
max\_ttl = 86400  
compress = true  
viewer\_protocol\_policy = "redirect-to-https"  
}  
price\_class = "PriceClass\_200"  
restrictions {  
geo\_restriction {  
restriction\_type = "whitelist"  
locations = ["IN"]  
}  
}  
tags = {  
Name="Task1CFDistribution"  
Environment = "production"  
}  
viewer\_certificate {  
cloudfront\_default\_certificate = true  
}  
retain\_on\_delete = truedepends\_on=[  
 aws\_s3\_bucket.task1bucketvinuja00vinuja00  
]  
}

This script will create a **CloudFront**distribution using an S3 bucket. In this bucket, we have stored all of the **assets**of our site like images, icons, etc. This CloudFront distribution will provide us one URL. By using this URL, we can access the objects inside the bucket.

Here we require that whenever the Infrastructure is destroyed, the CloudFront distribution should not get destroyed. Because, if we create new CloudFront distribution each time, we will be required to change the assets URLs every time. To overcome this problem we will set ***retain\_on\_delete*to*true****. This will disable* the distribution instead of deleting it when destroying the resource through Terraform.

***WOAH! Too much huh? Just a little more to go!***

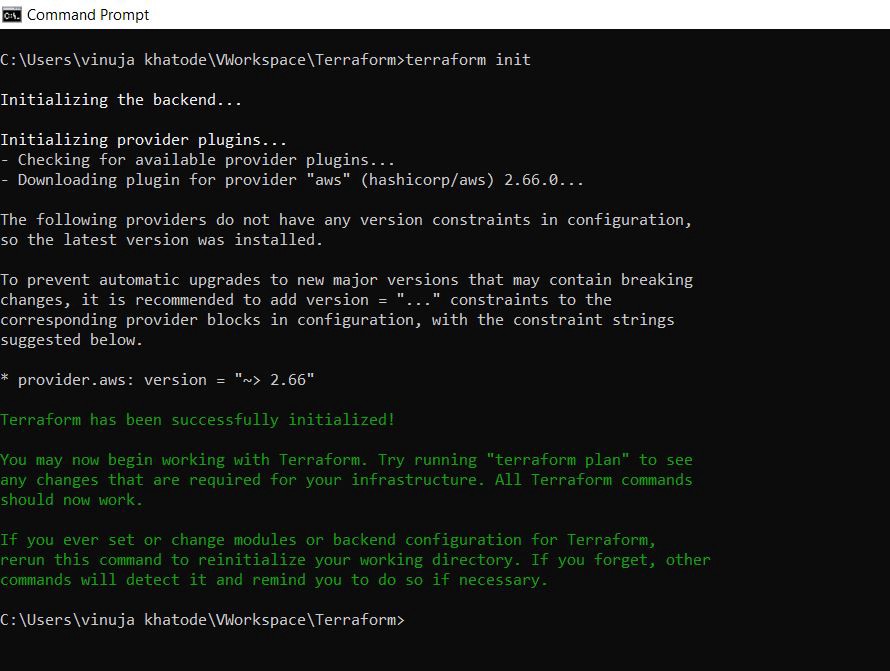
* Creating Bucket Policy for CloudFront.

// AWS Bucket Policy for CloudFront  
data "aws\_iam\_policy\_document" "s3\_policy" {  
statement {  
actions = ["s3:GetObject"]  
resources = ["${aws\_s3\_bucket.task1bucketvinuja00vinuja00.arn}/\*"]  
principals {  
type = "AWS"  
identifiers = ["${aws\_cloudfront\_origin\_access\_identity.origin\_access\_identity.iam\_arn}"]  
}  
}  
statement {  
actions = ["s3:ListBucket"]  
resources = ["${aws\_s3\_bucket.task1bucketvinuja00vinuja00.arn}"]  
principals {  
type = "AWS"  
identifiers = ["${aws\_cloudfront\_origin\_access\_identity.origin\_access\_identity.iam\_arn}"]  
}  
}  
}  
resource "aws\_s3\_bucket\_policy" "s3BucketPolicy" {  
bucket = "${aws\_s3\_bucket.task1bucketvinuja00vinuja00.id}"  
policy = "${data.aws\_iam\_policy\_document.s3\_policy.json}"  
}

This will create a Bucket Policy for the CloudFront.

***So, FINALLY, Our code is ready to launch the whole infrastructure in just one click!!***

**Perform *terraform init* and *terraform apply!!***

*Let us see the outputs no*

*Let us see the outputs no*