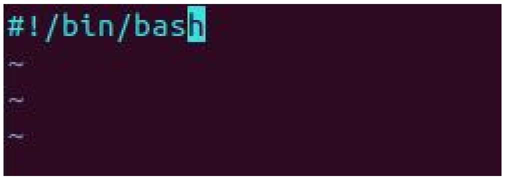
ASSESSMENT ON : TERRAFORM 2

TO THE NEW

1. Launch an ASG in AWS and do Rolling Deployment with change in User Data in LaunchConfig using terraform.

STEP 1: User-data (empty now)

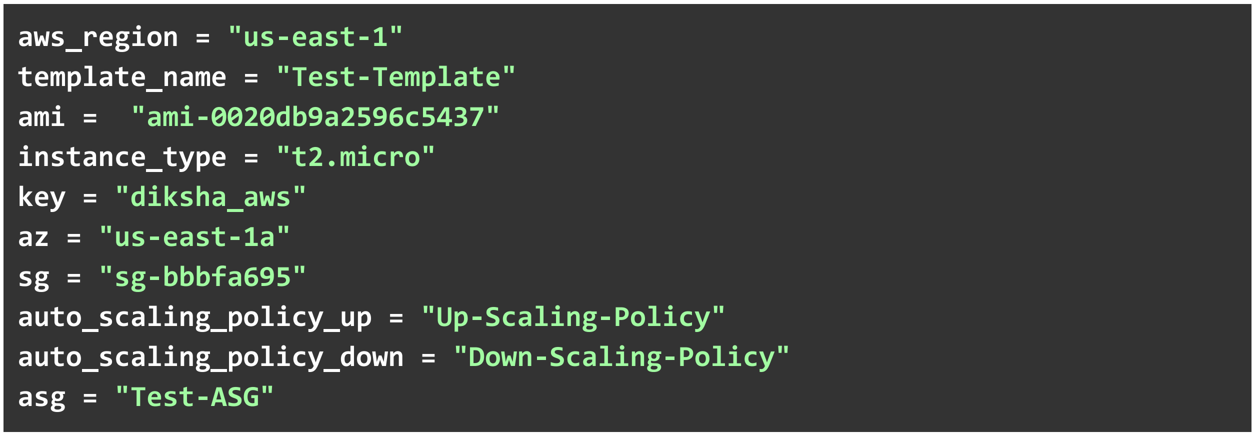


```
#!/bin/bash
~
~
~
```

STEP 2: main.tf

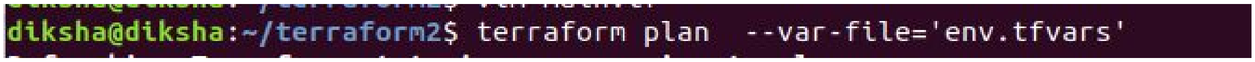
```
variable "aws_region" {}
variable "template_name" {}
variable "ami" {}
variable "instance_type" {}
variable "key" {}
variable "az" {}
variable "sg" {}
variable "auto_scaling_policy_up" {}
variable "auto_scaling_policy_down" {}
variable "asg" {}
provider "aws" {
        region = var.aws_region
}
resource "aws_launch_template" "template" {
        name = var.template_name
        image_id = var.ami
        instance_initiated_shutdown_behavior = "terminate"
        instance_type = var.instance_type
        key_name = var.key
        placement {
                availability_zone = var.az
        }
        vpc_security_group_ids = [var.sg]
        user_data = filebase64("userdata.sh")
        lifecycle {
                create_before_destroy = true
        }
}
resource "aws_autoscaling_group" "asg" {
        name = var.asg
        availability_zones = ["us-east-1a"]
        desired_capacity = 1
        max_size = 2
        min_size = 1
        health_check_grace_period = 300
        health_check_type = "EC2"
        force_delete = true
        launch_template {
                id = aws_launch_template.template.id
                version = "$Latest"
        }
        vpc_zone_identifier = ["subnet-7857a027"]

}
resource "aws_autoscaling_policy" "up_policy" {
        name = var.auto_scaling_policy_up
        scaling_adjustment = 1
        adjustment_type = "ChangeInCapacity"
        cooldown = 300
        autoscaling_group_name = aws_autoscaling_group.asg.name
}
resource "aws_autoscaling_policy" "down_policy" {
        name = var.auto_scaling_policy_down
        scaling_adjustment = -1
        adjustment_type = "ChangeInCapacity"
        cooldown = 300
        autoscaling_group_name = aws_autoscaling_group.asg.name
}
resource "aws_cloudwatch_metric_alarm" "cpu-high" {
        alarm_name = "High-CPU-Utilization"
        comparison_operator = "GreaterThanOrEqualToThreshold"
        evaluation_periods = 2
        metric_name = "CPU-Utilization"
        period = 300
        statistic = "Average"
        threshold = 80
        # Namespace must never be NULL
        namespace = "System/Linux"
        alarm_actions = [aws_autoscaling_policy.up_policy.arn]
        dimensions = {
                AutoScalingGroupName = aws_autoscaling_group.asg.name
        }
}
```

STEP 3: env.tfvars



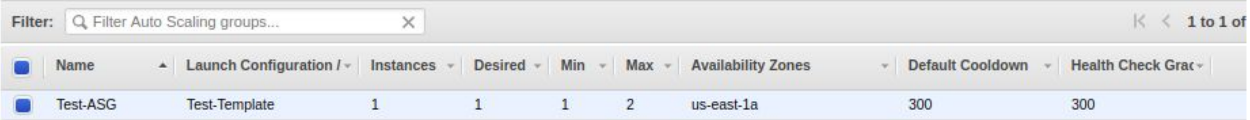
```
aws_region = "us-east-1"
template_name = "Test-Template"
ami =  "ami-0020db9a2596c5437"
instance_type = "t2.micro"
key = "diksha_aws"
az = "us-east-1a"
sg = "sg-bbbfa695"
auto_scaling_policy_up = "Up-Scaling-Policy"
auto_scaling_policy_down = "Down-Scaling-Policy"
asg = "Test-ASG"
```

STEP 4: terraform plan and apply



```
diksha@diksha:~/terraform2$ terraform plan  --var-file='env.tfvars'
```

STEP 5: Check console for ASG and all the infrastructure



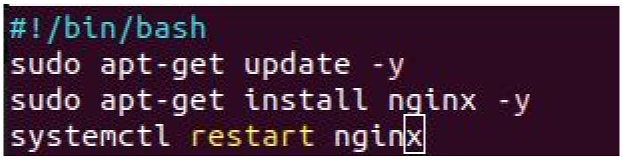
Filter: Filter Auto Scaling groups... 1 to 1 of

	Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
■	Test-ASG	Test-Template	1	1	1	2	us-east-1a	300	300

****

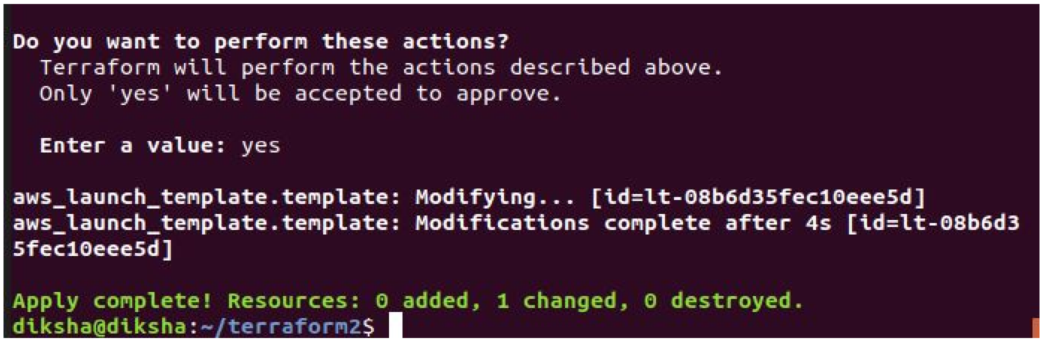
2. Deploy a sample nginx/tomcat/react service on it.

STEP 1: Make changes in the user-data



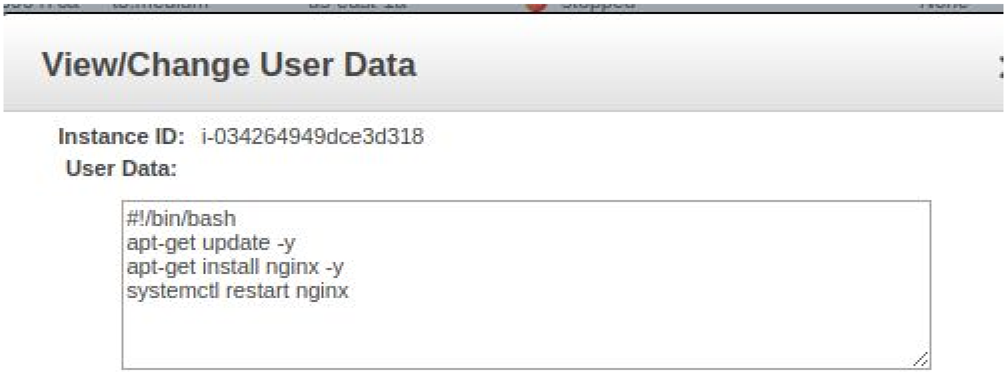
```
#!/bin/bash
sudo apt-get update -y
sudo apt-get install nginx -y
systemctl restart nginx
```

STEP 2: Running # terraform apply again:
-> One change = Launch Template Modified



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_launch_template.template: Modifying... [id=lt-08b6d35fec10eee5d]
aws_launch_template.template: Modifications complete after 4s [id=lt-08b6d3
5fec10eee5d]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
diksha@diksha:~/terraform2$
```

STEP 3: Checking User Data:

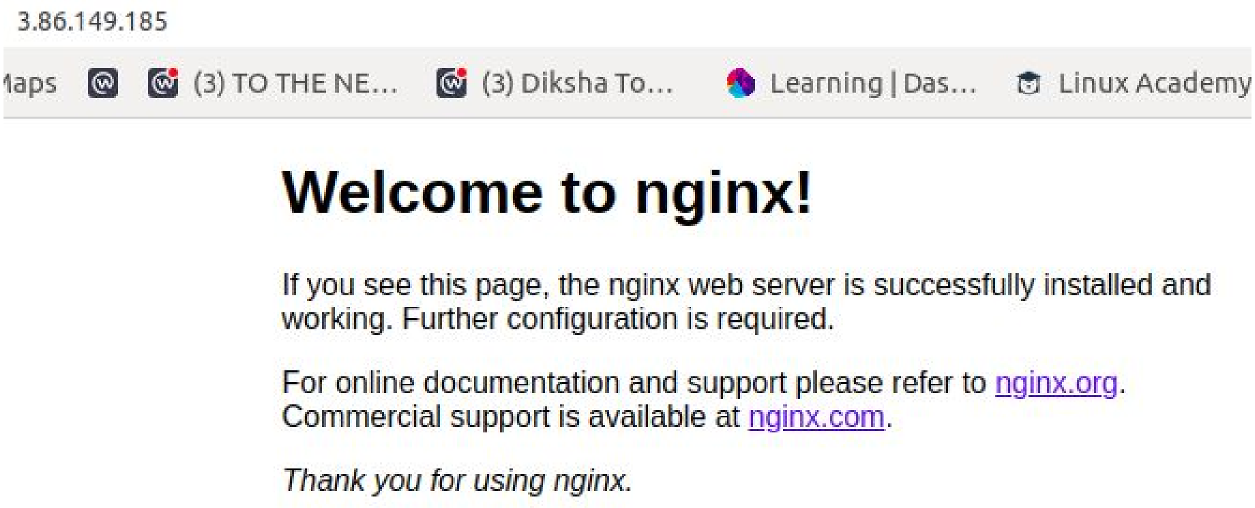


View/Change User Data

Instance ID: i-034264949dce3d318

User Data:

```
#!/bin/bash
apt-get update -y
apt-get install nginx -y
systemctl restart nginx
```



3.86.149.185

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

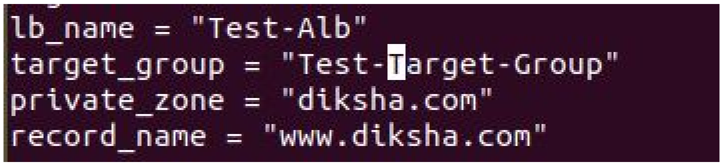
****

3. Attach a LB and create R53 endpoint pointing to lab, service should be accessible from the endpoint.

STEP 1: Creating Application Load Balancer, Target Group and Listener Rules and creating Private Hosted Zone and Record Set:

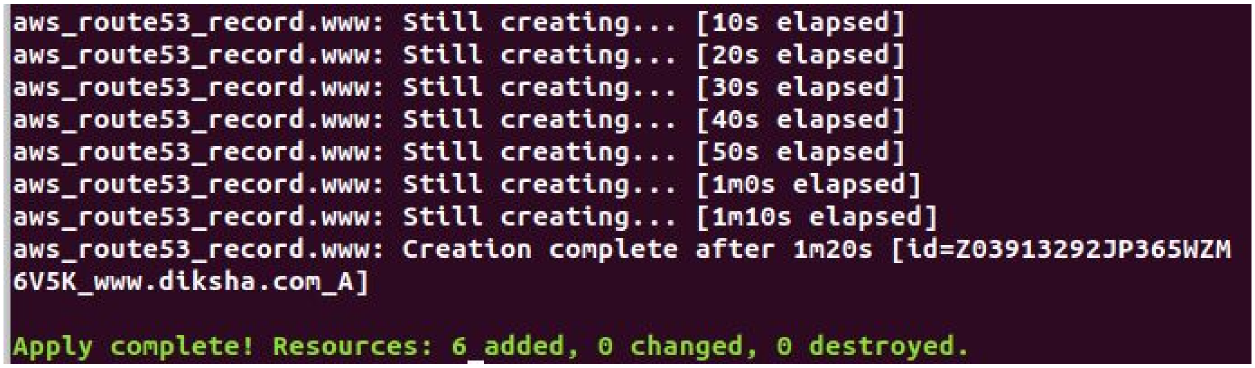
```
resource "aws_lb" "lb" {
        name = var.lb_name
        internal = "false"
        load_balancer_type = "application"
        enable_cross_zone_load_balancing = true
        security_groups = [var.sg]
        subnets = ["subnet-00986421", "subnet-0981bb37"]
}
resource "aws_lb_target_group" "tg" {
        name = var.target_group
        port = 80
        protocol = "HTTP"
        vpc_id = "vpc-306e6c4a"
}
resource "aws_lb_listener" "listener" {
        load_balancer_arn = aws_lb.lb.arn
        port = 80
        protocol = "HTTP"
        default_action {
                type = "forward"
                target_group_arn = aws_lb_target_group.tg.arn
        }
}
# Attatching Target-Group with Auto-Scaling Group ( Here, the Target of Target-Group is Auto-Scaling-Group and not Instances alone )
resource "aws_autoscaling_attachment" "attach_tg" {
        autoscaling_group_name = aws_autoscaling_group.asg.name
        alb_target_group_arn = aws_lb_target_group.tg.arn
}

resource "aws_route53_zone" "private_zone" {
        name = var.private_zone
        vpc {
                vpc_id = "vpc-306e6c4a"
        }
}
resource "aws_route53_record" "www" {
        zone_id = aws_route53_zone.private_zone.id
        name    = var.record_name
        type    = "A"
        alias {
                name = aws_lb.lb.dns_name
                zone_id = aws_lb.lb.zone_id
                evaluate_target_health = false
        }
}
```

STEP 2: Add the below variables in env.tfvars



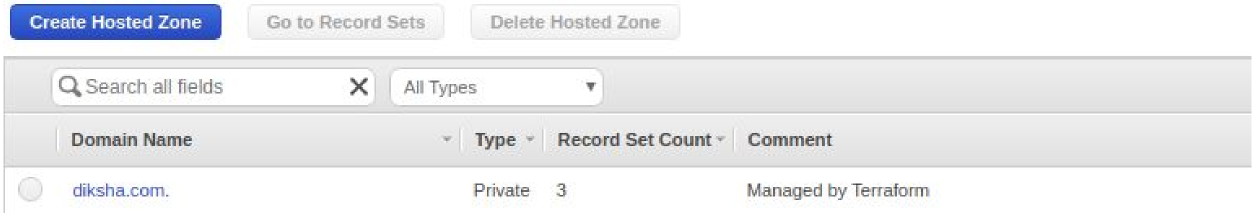
```
lb_name = "Test-Alb"
target_group = "Test-Target-Group"
private_zone = "diksha.com"
record_name = "www.diksha.com"
```

STEP 3: Apply terraform



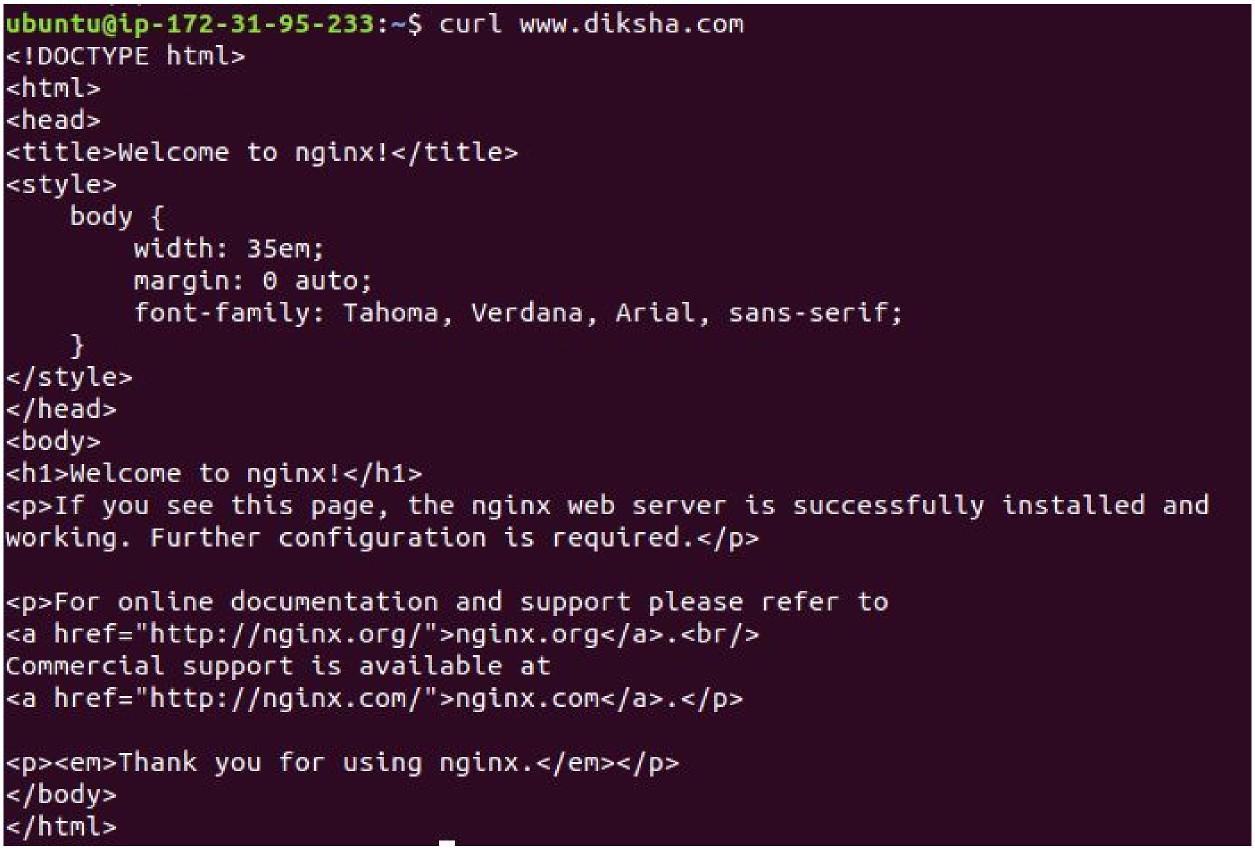
```
aws_route53_record.www: Still creating... [10s elapsed]
aws_route53_record.www: Still creating... [20s elapsed]
aws_route53_record.www: Still creating... [30s elapsed]
aws_route53_record.www: Still creating... [40s elapsed]
aws_route53_record.www: Still creating... [50s elapsed]
aws_route53_record.www: Still creating... [1m0s elapsed]
aws_route53_record.www: Still creating... [1m10s elapsed]
aws_route53_record.www: Creation complete after 1m20s [id=Z03913292JP365WZM
6V5K_www.diksha.com_A]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
```

STEP 4: Check for the DNS that you created



	Domain Name	Type	Record Set Count	Comment
○	diksha.com.	Private	3	Managed by Terraform

STEP 5: Checking the Output using DNS:



```
ubuntu@ip-172-31-95-233:~$ curl www.diksha.com
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

4. Variablize all parameters and pass values as env.tfvars file.

```
diksha@diksha:~/terraform2$ cat env.tfvars
aws_region = "us-east-1"
template_name = "Test-Template"
ami =  "ami-07ebfd5b3428b6f4d"
instance_type = "t2.micro"
key = "diksha_aws"
az = "us-east-1a"sg = "sg-bbbfa695"
auto_scaling_policy_up = "Up-Scaling-Policy"
auto_scaling_policy_down = "Down-Scaling-Policy"
asg = "Test-ASG"
lb_name = "Test-Alb"
target_group = "Test-Target-Group"
private_zone = "diksha.com"
record_name = "www.diksha.com"
```

In main.tf specify all variables at the top

```
diksha@diksha:~/terraform2$ cat main.tf variable "aws_region" {}
variable "template_name" {}
variable "ami" {}
variable "instance_type" {}
variable "key" {}
variable "az" {}
variable "sg" {}
variable "auto_scaling_policy_up" {}
variable "auto_scaling_policy_down" {}
variable "asg" {}
variable "lb_name" {}
variable "target_group" {}
variable "private_zone" {}
variable "record_name" {}
```

5. Create ASG from Launch Template and use a mix of on demand and on spot instance type in the ASG. Instance Type for On Demand and Spot should be diferent.

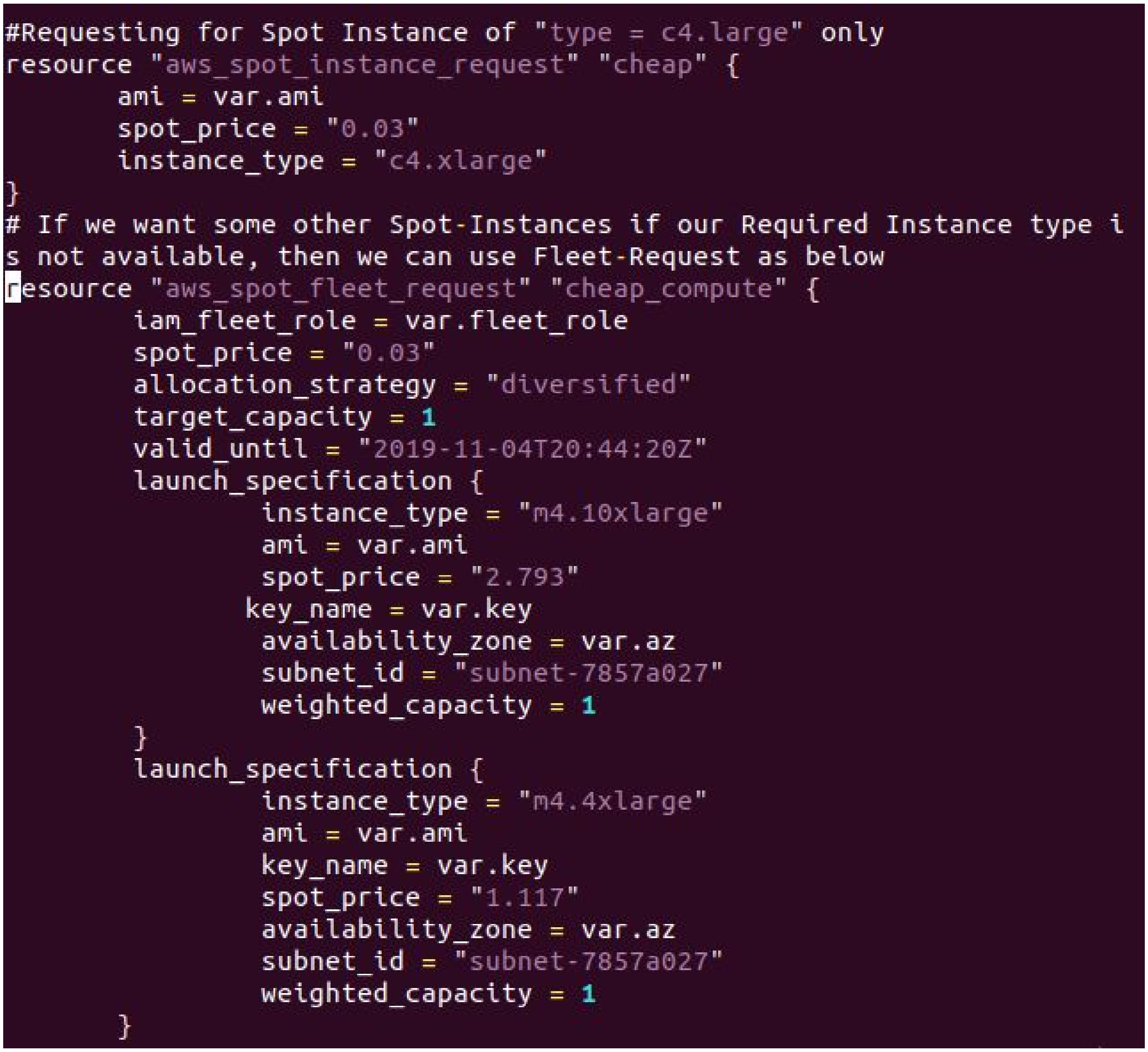
STEP 1: Destroyed previous Infrastructure>Editing main.tf (Changing the aws_autoscaling_group connfiguration):

The Instance Types of On-Demand and Spot-Instances are also Different.

```
#Requesting for Spot Instance of "type = c4.large" only
resource "aws_spot_instance_request" "cheap" {
       ami = var.ami
       spot_price = "0.03"
       instance_type = "c4.xlarge"
```

6.Enable Spot Feature to use multiple instance type if requested instance type is not available.

```
# If we want some other Spot-Instances if our Required Instance type is not available, then we can use Fleet-Request as below
resource "aws_spot_fleet_request" "cheap_compute" {
        iam_fleet_role = var.fleet_role
        spot_price = "0.03"
        allocation_strategy = "diversified"
       target_capacity = 1
        valid_until = "2019-11-04T20:44:20Z"
        launch_specification {
                instance_type = "m4.10xlarge"
                ami = var.ami
                spot_price = "2.793"
               key_name = var.key
                availability_zone = var.az
                subnet_id = "subnet-7857a027"
                weighted_capacity = 1
        }
        launch_specification {
                instance_type = "m4.4xlarge"
                ami = var.ami
                key_name = var.key
                spot_price = "1.117"
                availability_zone = var.az
                subnet_id = "subnet-7857a027"
                weighted_capacity = 1
       }
}
```



```
#Requesting for Spot Instance of "type = c4.large" only
resource "aws_spot_instance_request" "cheap" {
       ami = var.ami
       spot_price = "0.03"
       instance_type = "c4.xlarge"
}
# If we want some other Spot-Instances if our Required Instance type i
s not available, then we can use Fleet-Request as below
resource "aws_spot_fleet_request" "cheap_compute" {
        iam_fleet_role = var.fleet_role
        spot_price = "0.03"
        allocation_strategy = "diversified"
        target_capacity = 1
        valid_until = "2019-11-04T20:44:20Z"
        launch_specification {
                instance_type = "m4.10xlarge"
                ami = var.ami
                spot_price = "2.793"
               key_name = var.key
                availability_zone = var.az
                subnet_id = "subnet-7857a027"
                weighted_capacity = 1
        }
        launch_specification {
                instance_type = "m4.4xlarge"
                ami = var.ami
                key_name = var.key
                spot_price = "1.117"
                availability_zone = var.az
                subnet_id = "subnet-7857a027"
                weighted_capacity = 1
       }
```