

Name - Nandan Shailesh Kasat

Batch - Jan-Feb 2024

Terraform Task_2

Q1. Install Terraform in local machine, configure AWS provider. Initialize Terraform configuration.

Create a VPC with CIDR block 192.168.0.0/16

Create two public subnets in different availability zones within the VPC.

Create an Internet Gateway and attach it to the VPC.

Create a route table and associate it with the public subnets, setting the default route to the Internet Gateway.

Create security groups allowing HTTP (port 80) and SSH (port 22) access.

Note: Provide access_key, secret_key and region, CIDR for vpc and subnets using input variable.

And display vpc-id, sg-id

Ans:

1. Install Terraform and Configure AWS Provider:

2. Create a VPC:

Define your VPC resource in Terraform. Specify the CIDR block (e.g., 192.168.0.0/16) for your VPC.

```

GNU nano 7.2 main.tf
terraform {
  required_version = "~> 1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  access_key = var.access_key
  secret_key = var.secret_key
  region = var.region_name
}

resource "aws_instance" "myec2" {
  ami           = var.ami_id
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.mysg.id]
  key_name     = "tf-key-pair"
  tags = {
    Name = "terraform1"
  }
}

resource "aws_vpc" "my_vpc" {
  cidr_block = "192.168.0.0/16"
}

```

3. Create Public Subnets:

Create two public subnets in different availability zones (AZs) within your VPC.

```

GNU nano 7.2 main.tf
resource "aws_subnet" "public_subnet1" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "192.168.1.0/24"
  availability_zone = "ap-southeast-1"
}

resource "aws_subnet" "public_subnet2" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "192.168.2.0/24"
  availability_zone = "ap-southeast-2"
}

```

4. Create an Internet Gateway (IGW):

Deploy an IGW and associate it with your VPC to enable internet traffic.

```

GNU nano 7.2 main.tf
resource "aws_internet_gateway" "my_igw" {
  vpc_id = aws_vpc.my_vpc.id
}

```

5. Create a Route Table:

Define a route table for your VPC. Associate the public subnets with this route table. Set the default route to the IGW.

```
ubuntu@ip-172-31-39-206: ~/   
GNU nano 7.2 main.tf  
}  
resource "aws_route_table" "public_route_table" {  
  vpc_id = aws_vpc.my_vpc.id  
}  
  
resource "aws_route" "public_route" {  
  route_table_id = aws_route_table.public_route_table.id  
  destination_cidr_block = "0.0.0.0/0"  
  gateway_id = aws_internet_gateway.my_igw.id  
}  
  
resource "aws_route_table_association" "public_subnet1_association" {  
  subnet_id = aws_subnet.public_subnet1.id  
  route_table_id = aws_route_table.public_route_table.id  
}  
  
resource "aws_route_table_association" "public_subnet2_association" {  
  subnet_id = aws_subnet.public_subnet2.id  
  route_table_id = aws_route_table.public_route_table.id  
}
```

6. Create Security Groups:

Define security groups allowing HTTP (port 80) and SSH (port 22) access.

```
ubuntu@ip-172-31-39-206: ~/ × + v
GNU nano 7.2 main.tf

resource "aws_route_table_association" "public_subnet2_association" {
  subnet_id      = aws_subnet.public_subnet2.id
  route_table_id = aws_route_table.public_route_table.id
}

resource "aws_security_group" "mysg" {
  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_key_pair" "tf-key-pair" {
  key_name      = "tf-key-pair"
  public_key    = tls_private_key.rsa.public_key_openssh
}
resource "tls_private_key" "rsa" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
resource "local_file" "tf-key" {
  content  = tls_private_key.rsa.private_key_pem
  filename = "tf-key-pair"
}
}
```

In Terraform.tfvars mention the Access Key, Secret Key and ami_id.

```
ubuntu@ip-172-31-39-206: ~/ × + v
GNU nano 7.2 terraform.tfvars

access_key = "AKIAU6GDXUSUAQ46W2T3"
secret_key = "RiLV/1G+0lUU4VZjZuPXCwVhOep8h4nt6Mosl75w"
ami_id     = "ami-01376101673c89611"
```

7. Display VPC ID and Security Group ID:

After applying your Terraform configuration, you can retrieve the VPC ID and security group ID using Terraform outputs.

```
ubuntu@ip-172-31-39-206: ~/ + ▼
GNU nano 7.2 output.tf *
output "vpc_id" {
  value = aws_vpc.my_vpc.id
}

output "security_group_id" {
  value = aws_security_group.my_sg.id
}
```

8. Final Output:

```
ubuntu@ip-172-31-39-206: ~/ + ▼
local_file.tf-key: Refreshing state... [id=cce86a15ab70a37cc1570939fc14a1465cf5f58c]
aws_security_group.msg: Refreshing state... [id=sg-09f4541751ffb60c9]
aws_vpc.my_vpc: Refreshing state... [id=vpc-091cd577c34511bcc]
aws_instance.myc2: Refreshing state... [id=i-04b7f621bb412d32a]
aws_subnet.public_subnet1: Refreshing state... [id=subnet-01cf9e997d02f6e87]
aws_subnet.public_subnet2: Refreshing state... [id=subnet-04acaa9e974c303c0]
aws_route_table.public_route_table: Refreshing state... [id=rtb-00124cc0de9837273]
aws_internet_gateway.my_igw: Refreshing state... [id=igw-03208721d58dd5fd7]
aws_route.public_route: Refreshing state... [id=r-rtb-00124cc0de98372731080289494]
aws_route_table_association.public_subnet2_association: Refreshing state... [id=rtbassoc-02e71bbc68375364c]
aws_route_table_association.public_subnet1_association: Refreshing state... [id=rtbassoc-0cb31805822ffaaaf1]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
+ create

Terraform will perform the following actions:

# aws_key_pair.tf-key-pair will be created
+ resource "aws_key_pair" "tf-key-pair" {
+   arn                = (known after apply)
+   fingerprint        = (known after apply)
+   id                 = (known after apply)
+   key_name            = "tf-key-pair4"
+   key_name_prefix     = (known after apply)
+   key_pair_id         = (known after apply)
+   public_key          = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCTBe6BjTyy9ePJAgF35h3noyrWX0L2ZTN7p9dtunzyj2VNXm7LXL
3Ly4Ruh0xSURUf2FdbGDxHhr6LBimA8VSVNBuzDkZ3kEge0Mb9Sm6XyHGAf3wSEcwb5+5hAH/6y8N/ezU6J26GSoZ+MXgxi+BP/3HtJFEhHerN3ZLI/
bjthXxht0mcMuyeTqp9scVTly589KGrTke6gLNcJpKUYwWUTGZjr40PmWbeCVYRJjS9h9vkvfM/y0JoU9CxNZXhXcvHJ5LBKUP9MJzBWVDRbGtXzHKpAE
awIriVR6jhmK4A70ZtcN07oQ2yooIbirkD0JTYpwQLqv51opqjGsGw8tjmzq6w3IiQCsb0agCPBJ5jevTzQrEaVPfA2wPILw+rXAwnEeWrzXz4FjunbYX
BUMJ6k0U8WobbujIM2FCXSk/eqtyd+GeVmzZFPGYLXCb+SVix3sE4W/BGQfTUvEODwUWcvBfYowcJwoxQRebUkdDKhtYaj3239rSAivDXlLBpdqTJh9i
+   tags_all           = (known after apply)
}

Plan: 1 to add, 0 to change, 0 to destroy.
aws_key_pair.tf-key-pair: Creating...
aws_key_pair.tf-key-pair: Creation complete after 0s [id=tf-key-pair4]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

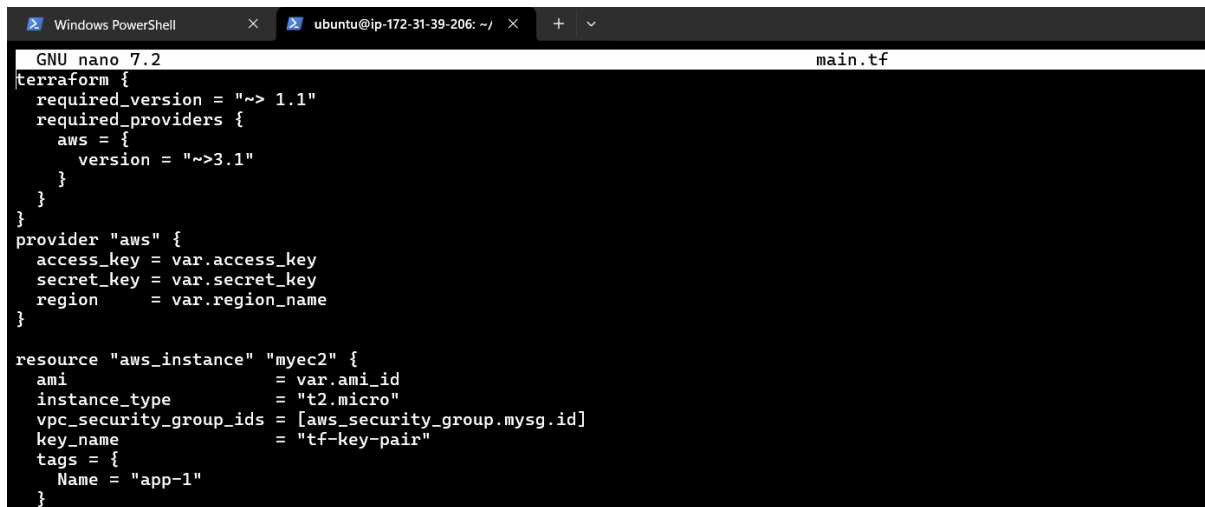
Outputs:
security_group_id = "sg-09f4541751ffb60c9"
vpc_id = "vpc-091cd577c34511bcc"
ubuntu@ip-172-31-39-206: ~/variable_demo$ |
```

Q2. Launch an EC2 instances with names “app-1” and install apache, create two pages at its default location using provisioner block. Display webpages on browser.

Ans:

1. Launch an EC2 instance name “app_1”

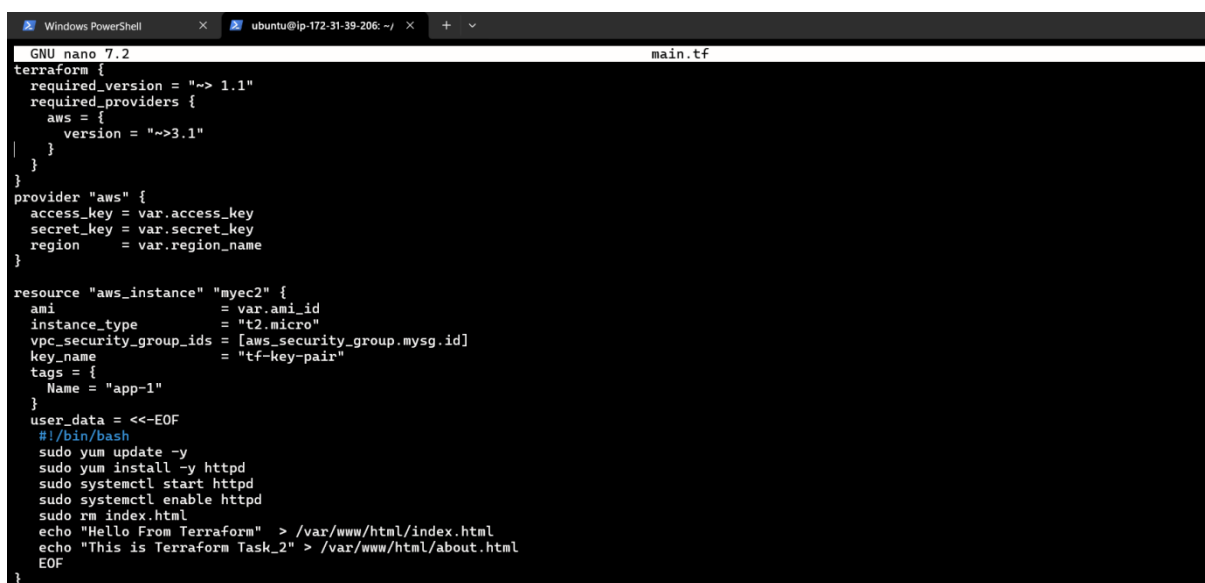
Define your EC2 instance resource in Terraform. Set the instance name to “app-1” and choose an appropriate instance type (e.g., t2.micro).



```
GNU nano 7.2 main.tf
terraform {
  required_version = "~> 1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  access_key = var.access_key
  secret_key = var.secret_key
  region     = var.region_name
}

resource "aws_instance" "myec2" {
  ami           = var.ami_id
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.msg.id]
  key_name      = "tf-key-pair"
  tags = {
    Name = "app-1"
  }
}
```

2. Install Apache:



```
GNU nano 7.2 main.tf
terraform {
  required_version = "~> 1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  access_key = var.access_key
  secret_key = var.secret_key
  region     = var.region_name
}

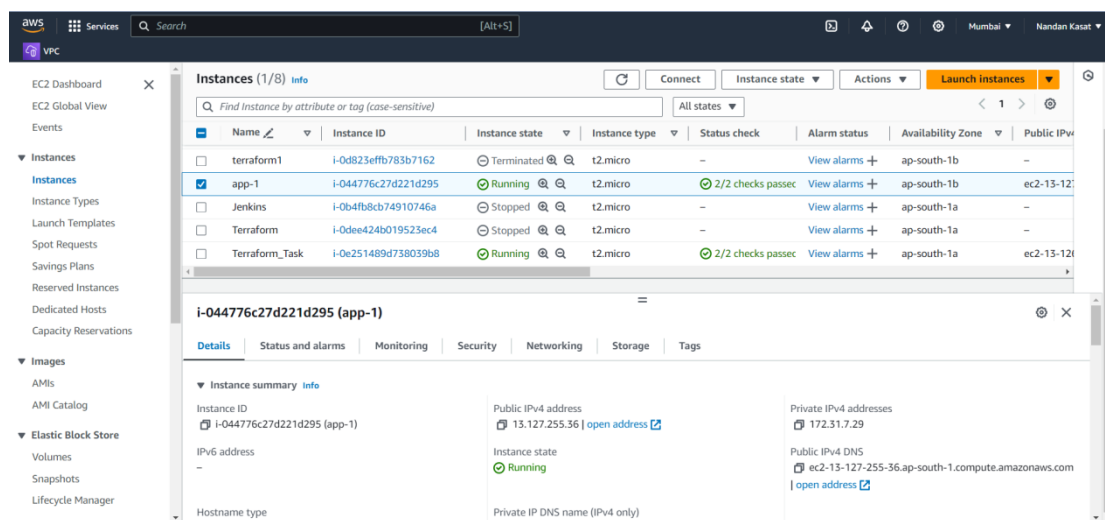
resource "aws_instance" "myec2" {
  ami           = var.ami_id
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.msg.id]
  key_name      = "tf-key-pair"
  tags = {
    Name = "app-1"
  }
  user_data = <<-EOF
  #!/bin/bash
  sudo yum update -y
  sudo yum install -y httpd
  sudo systemctl start httpd
  sudo systemctl enable httpd
  sudo rm index.html
  echo "Hello From Terraform" > /var/www/html/index.html
  echo "This is Terraform Task_2" > /var/www/html/about.html
  EOF
}
```

3. Create Web Pages:

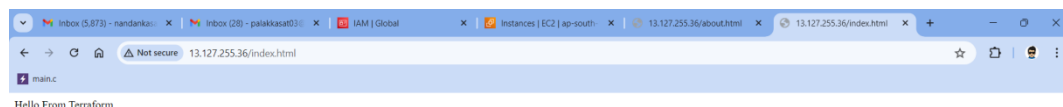
After installing Apache, create two HTML pages (e.g., index.html and about.html) in the default web server location (/var/www/html).

```
echo "Hello From Terraform" > /var/www/html/index.html
echo "This is Terraform Task_2" > /var/www/html/about.html
EOF
}
```

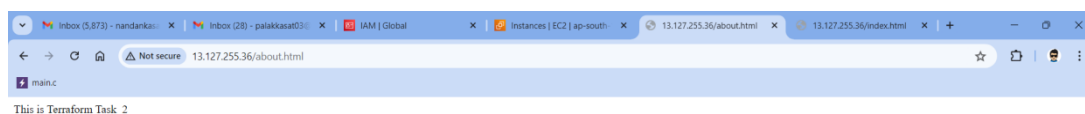
4. You can see instance app_1 on dashboard.



5. Index.html



6. About.html



Q3. Create an Auto Scaling Group with a Launch Configuration to manage the EC2 instances, using Terraform.

Ans:

1. Start the Instance:

2. Define a Launch Configuration:

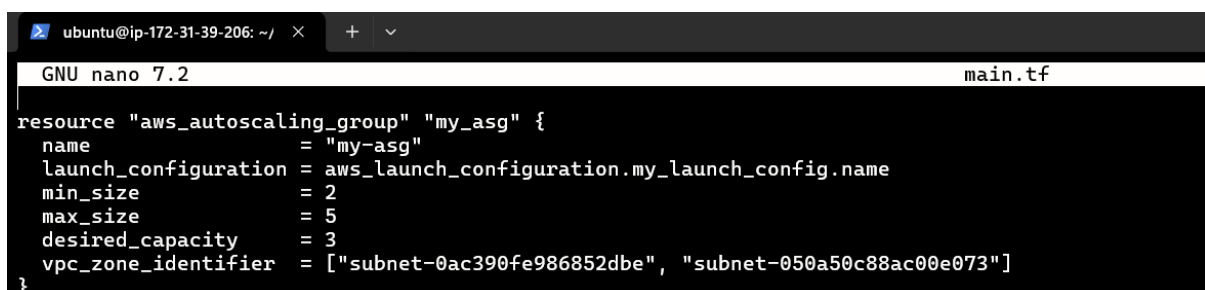
First, create a launch configuration that defines the instance specifications.



```
ubuntu@ip-172-31-39-206: ~/ + ▼
GNU nano 7.2 main.tf
terraform{
  required_version = "~> 1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  access_key = var.access_key
  secret_key = var.secret_key
  region     = var.region_name
}
resource "aws_instance" "myec2" {
  ami           = var.ami_id
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.mysg.id]
  key_name      = "tf-key-pair"
  tags = {
    Name = "app-1"
  }
}
resource "aws_launch_configuration" "my_launch_config" {
  name_prefix = "my-launch-config"
  image_id    = var.ami_id
  instance_type = "t2.micro"
}
```

3. Create an Auto Scaling Group:

Define your ASG resource, referencing the launch template.



```
ubuntu@ip-172-31-39-206: ~/ + ▼
GNU nano 7.2 main.tf
resource "aws_autoscaling_group" "my_asg" {
  name                  = "my-asg"
  launch_configuration = aws_launch_configuration.my_launch_config.name
  min_size              = 2
  max_size              = 5
  desired_capacity      = 3
  vpc_zone_identifier   = ["subnet-0ac390fe986852dbe", "subnet-050a50c88ac00e073"]
}
```


4. Scaling Policies:

We can define scaling policies based on metrics like CPU utilization or custom metrics.

Attach these policies to your ASG.

```
resource "aws_autoscaling_policy" "scale_up_policy" {
  name = "scale-up-policy"
  autoscaling_group_name = aws_autoscaling_group.my_asg.name
  adjustment_type = "ChangeInCapacity"
  scaling_adjustment = 1
}
```

5. The Auto Scaling Group is created on the Dashboard.

```
ubuntu@ip-172-31-39-206: ~$ terraform apply
+ health_check_type = (known after apply)
+ id = (known after apply)
+ launch_configuration = "my-launch-config202407050543076608800000002"
+ max_size = 5
+ metrics_granularity = "1Minute"
+ min_size = 2
+ name = "my-asg"
+ name_prefix = (known after apply)
+ protect_from_scale_in = false
+ service_linked_role_arn = (known after apply)
+ vpc_zone_identifier = [
  + "subnet-050a50c88ac00e073",
  + "subnet-0ac390fe906852dbe",
]
+ wait_for_capacity_timeout = "10m"
}

# aws_autoscaling_policy.scale_up_policy will be created
+ resource "aws_autoscaling_policy" "scale_up_policy" {
+ adjustment_type = "ChangeInCapacity"
+ arn = (known after apply)
+ autoscaling_group_name = "my-asg"
+ id = (known after apply)
+ metric_aggregation_type = (known after apply)
+ name = "scale-up-policy"
+ policy_type = "SimpleScaling"
+ scaling_adjustment = 1
}

Plan: 2 to add, 0 to change, 0 to destroy.
aws_autoscaling_group.my_asg: Creating...
aws_autoscaling_group.my_asg: Still creating... [10s elapsed]
aws_autoscaling_group.my_asg: Still creating... [20s elapsed]
aws_autoscaling_group.my_asg: Still creating... [30s elapsed]
aws_autoscaling_group.my_asg: Still creating... [40s elapsed]
aws_autoscaling_group.my_asg: Still creating... [50s elapsed]
aws_autoscaling_group.my_asg: Still creating... [1m0s elapsed]
aws_autoscaling_group.my_asg: Creation complete after 1m7s [id=my-asg]
aws_autoscaling_policy.scale_up_policy: Creating...
aws_autoscaling_policy.scale_up_policy: Creation complete after 0s [id=scale-up-policy]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

