AWS Terraform Project

Explanation of each section of the Terraform script step by step:

```
provider "aws" {
  region = "us-east-1"
}
```

This block specifies the cloud provider (in this case, AWS) and the region to use.

```
data "aws_availability_zones" "available" {}
```

This block retrieves information about the available availability zones in the specified region.

```
data "aws_vpc" "default" {
  default = true
}
```

This block retrieves information about the default VPC in the specified region.

```
data "aws_subnets" "default" {
  filter {
    name = "default-for-az"
    values = ["true"]
  }
}
```

This block retrieves information about the default subnets in the specified region. It filters for subnets that are marked as default for their availability zone.

```
resource "aws_security_group" "test_sg" {
  name_prefix = "test-sg"
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 5000
    to_port = 5000
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 22
    to_port     = 22
    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 = "test-sg"
  }
  vpc_id = data.aws_vpc.default.id
}
```

This block creates an AWS security group named "test_sg" with inbound rules allowing traffic on ports 80, 8080, and 22. It also allows all outbound traffic. The security group is associated with the default VPC.

```
resource "aws_launch_configuration" "example" {
  name_prefix      = "example-lc-"
  image_id         = "ami-007855ac798b5175e"
  instance_type    = "t2.micro"
  key_name         = "vinay"
  security_groups  = [aws_security_group.terraform_sg.id]
  user_data        = <<-EOF
    #!/bin/bash
    sudo apt-get update -y
    sudo apt install docker.io -y
    docker pull digitalocean/flask-helloworld
    docker run -d -p 5000:5000 digitalocean/flask-helloworld
    EOF
}
```

This code block creates an AWS launch configuration that specifies the Amazon Machine Image (AMI), instance type, key name, security group, and user data to use when launching EC2 instances.

The user data installs Docker and pulls the flask-helloworld Docker image, which is then run on port 5000.

```
resource "aws_autoscaling_group" "example" {
  name                 = "terraform-asg"
  max_size             = 3
  min_size             = 1
  desired_capacity     = 1
  health_check_grace_period = 300
  health_check_type = "EC2"
  force_delete = true
  launch_configuration = aws_launch_configuration.example.id
  vpc_zone_identifier  = [data.aws_subnets.default.ids[0],
data.aws_subnets.default.ids[0]]
  termination_policies = ["OldestInstance", "ClosestToNextInstanceHour",
"Default"]
  tag {
    key                 = "Name"
    value               = "example"
    propagate_at_launch = true
  }
}
```

Auto Scaling Group is a group of Amazon Elastic Compute Cloud (EC2) instances that are created from a common Amazon Machine Image (AMI) and that automatically adjust capacity to maintain steady and predictable performance for an application.

The block sets various parameters for the auto scaling group, including the name, max_size, min_size, and desired_capacity.

The health_check_grace_period specifies the amount of time, in seconds, that the Auto Scaling Group waits before checking the health status of an EC2 instance that has come into service.

health_check_type specifies the type of health check that the Auto Scaling Group performs. In this case, it is set to "EC2".

force_delete is set to true, which means that any instances that are running as part of the group will be terminated if the group is deleted.

The launch_configuration parameter specifies the ID of the launch configuration to use for instances in the Auto Scaling Group.

The vpc_zone_identifier parameter specifies the IDs of the subnets in which the instances should be launched.

The termination_policies parameter specifies the order in which instances should be terminated when scaling down. In this case, the order is "OldestInstance", "ClosestToNextInstanceHour", and "Default".

Finally, the tag block specifies a tag to apply to the instances in the Auto Scaling Group. The tag's key is set to "Name", its value is set to "example", and propagate_at_launch is set to true, which means that the tag will be applied to any instances that are launched as part of the Auto Scaling Group.

```
resource "aws_cloudwatch_metric_alarm" "cpu_alarm" {
  alarm_name          = "example-cpu-alarm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic           = "Average"
  threshold           = "50"
  alarm_description   = "This metric monitors CPU utilization for the
example application"
  alarm_actions       = [aws_autoscaling_policy.example.arn]
  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.example.name
  }
}
```

this block creates a CloudWatch alarm called "cpu_alarm" that monitors the CPU utilization of instances in the Auto Scaling group. If the CPU utilization is greater than or equal to 50% for 2 consecutive periods of 2 minutes, the alarm triggers and sends a notification to the specified target. The target is an autoscaling policy.

```
resource "aws_autoscaling_policy" "example" {
  name                   = "example-autoscaling-policy"
  policy_type            = "TargetTrackingScaling"
  estimated_instance_warmup = 120
  target_tracking_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ASGAverageCPUUtilization"
    }
    target_value = 50.0
  }
  autoscaling_group_name  = aws_autoscaling_group.example.name
}
```

This block defines an autoscaling policy called "example" that uses target tracking scaling to adjust the desired capacity of the Auto Scaling group based on the ASG average CPU utilization metric. The policy increases or decreases the desired capacity of the group to maintain a target ASG average CPU utilization of 50%. The policy has an estimated instance warm-up time of 120 seconds.

```
resource "aws_cloudwatch_metric_alarm" "memory_alarm" {
  alarm_name          = "example-memory-alarm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "MemoryUtilization"
  namespace           = "System/Linux"
  period              = "120"
  statistic           = "Average"
  threshold           = "50"
  alarm_description   = "This metric monitors memory utilization for the
example application"
  alarm_actions       = [aws_autoscaling_policy.example.arn]
  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.example.name
  }
}
```

This block creates a CloudWatch alarm called "memory_alarm" that monitors the memory utilization of instances in the Auto Scaling group. If the memory utilization is greater than or equal to 50% for 2 consecutive periods of 2 minutes, the alarm triggers and sends a notification to the autoscaling policy defined in the above block.

Credentials:
jenkins url: http://ec2-18-233-167-141.compute-1.amazonaws.com:8080/
Username: vinay
Password: 123456

flask_application_url:

1. http://ec2-35-92-137-184.us-west-2.compute.amazonaws.com:5000

Note: these IP may change get IP from EC2 dashboard (public DNS)

——————————————————————————————————*——————————————————————————————————

Brief of project:

I created one EC2 instance named script-test and connected it using vinay.pem file (sent in whatsapp). And installed terraform

1. ssh -i "vinay.pem" ubuntu@ec2-35-88-162-201.us-west-2.compute.amazonaws.com
2. sudo su
3. sudo apt-get update
4. curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
5. sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
6. sudo apt-get update
7. sudo apt-get install terraform
8. terraform --version
9. Created a script.tf file and initiated terraform
   a. (go to directory where terraform.tf is located)
   b. terraform init
   c. terraform plan
   d. terraform apply --auto-approve
      to delete every created resource using terraform run
   e. terraform destroy --auto-approve

Retrieve jenkins password:

10. Login to aws and goto EC2 and connect to created instance and Run sudo su
11. Docker ps -a (list all container)
12. docker exec -it <container id> /bin/bash

```
 Container id of jenkins container

13. Cd /var/jenkins_home/secrets/
14. Cat intialpasswords
   Copy password

—--------------------------------------*---------------------
```