Automating infrastructure scaling based on load or resource utilization Terraform

Scaling with existing Autoscaling group

- The Kops cluster already created autoscaling groups for control plane and worker node
  Worker Node: **nodes-us-west-1b.dptcluster.k8s.local**
  Control Plane: **control-plane-us-west1b.masters.dptcluster.k8s.local**

- ASG's created by Kops can be managed through Terraform
  For the worker node we can Adjust the **desired_capacity, min_size**, and **max_size** values.

- First, we can import the existing ASG's into terraform using the commands
  terraform import aws_autoscaling_group.worker_nodes nodes-us-west-1b.dptcluster.k8s.local
  terraform import aws_autoscaling_group.control_plane control-plane-us-west-1b.masters.dptcluster.k8s.local

  We can run **terraform plan** to verify the imported configuration.

- Adjusting the autoscaling configuration for worker node

```
resource "aws_autoscaling_group" "worker_nodes" {
  name                 = "nodes-us-west-1b.dptcluster.k8s.local"
  min_size             = 2
  max_size             = 10
  desired_capacity     = 3
  launch_configuration = aws_launch_configuration.worker_nodes.id
  availability_zones   = ["us-west-1b"]

  tag {
    key                 = "kubernetes.io/cluster/dptcluster.k8s.local"
    value               = "owned"
    propagate_at_launch = true
  }

  # Attach CloudWatch Alarms for dynamic scaling
  target_group_arns = [aws_lb_target_group.worker_nodes.arn]
}
```

- Then we configure CloudWatch alarms based on CPU or memory utilization for triggering scaling to scale up and scale down.

```
resource "aws_cloudwatch_metric_alarm" "scale_up" {
  alarm_name          = "ScaleUpAlarm"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 60
  statistic           = "Average"
  threshold           = 70
  alarm_actions       = [aws_autoscaling_policy.scale_up.arn]

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.worker_nodes.name
  }
}
```

```
resource "aws_cloudwatch_metric_alarm" "scale_down" {
  alarm_name          = "ScaleDownAlarm"
  comparison_operator = "LessThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 60
  statistic           = "Average"
  threshold           = 20
  alarm_actions       = [aws_autoscaling_policy.scale_down.arn]

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.worker_nodes.name
  }
}
```

- Then we can define autoscaling policies to scale up and scale down

```
resource "aws_autoscaling_policy" "scale_up" {
  name                   = "scale-up"
  scaling_adjustment     = 1
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 300
  autoscaling_group_name = aws_autoscaling_group.worker_nodes.name
}

resource "aws_autoscaling_policy" "scale_down" {
  name                   = "scale-down"
  scaling_adjustment     = -1
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 300
  autoscaling_group_name = aws_autoscaling_group.worker_nodes.name
}
```

- We can run **terraform plan and terraform apply** to plan, validate and apply the changes.
- To Test scaling, we can generate high resource utilization on the worker nodes.