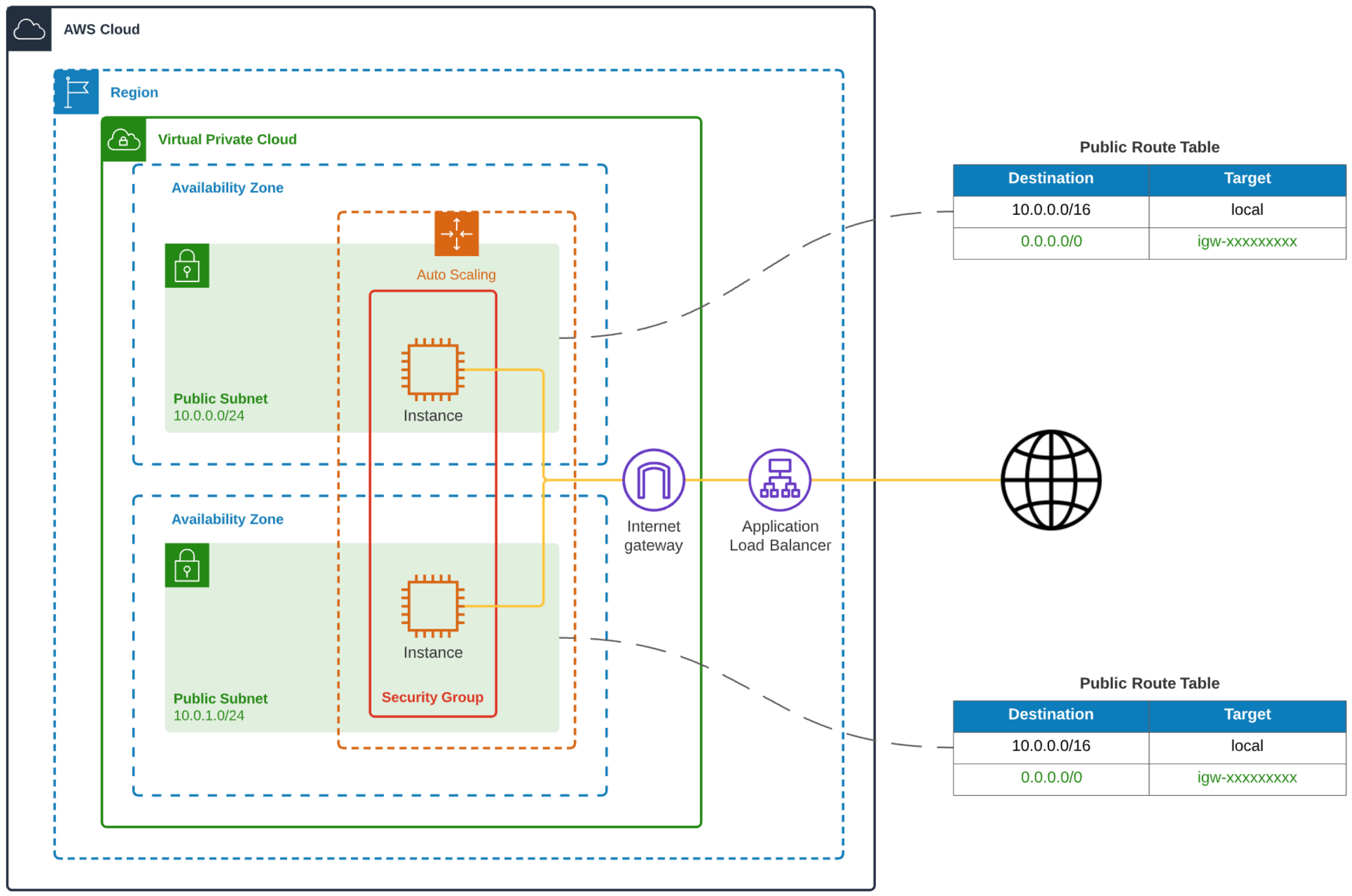
Terraform – Managing auto Scaling Group and Load Balancrer

Table of contents

As soon as you learn how to manage basic network infrastructure in AWS using Terraform (see “Terraform recipe – Managing AWS VPC – Creating Public Subnet” and “Terraform recipe – Managing AWS VPC – Creating Private Subnets”), you want to start creating auto-scalable infrastructures.



Public Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-xxxxxxxxx

Public Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-xxxxxxxxx

Auto Scaling Groups

Usually, Auto Scaling Groups are used to control the number of instances executing the same task like rendering dynamic web pages for your website, decoding videos and images, or calculating machine learning models.

Auto Scaling Groups also allows you to dynamically control your server pool size – increase it when your web servers are processing more traffic or tasks than usual, or decrease it when it becomes quieter.

In any case, this feature allows you to save your budget and make your infrastructure more fault-tolerant significantly.

Let’s build a simple infrastructure, which consists of several web servers for serving website traffic. In the following article, we’ll add RDS DB to our infrastructure.

Setting up VPC.

Let’s assemble it in a new `infrastructure.tf` file. First of all, let’s declare VPC, two Public Subnets, Internet Gateway and Route Table

```
resource "aws_vpc" "my_vpc" {
  cidr_block       = "10.0.0.0/16"
  enable_dns_hostnames = true

  tags = {
    Name = "My VPC"
  }
}

resource "aws_subnet" "public_us_east_1a" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "Public Subnet us-east-1a"
  }
}

resource "aws_subnet" "public_us_east_1b" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1b"

  tags = {
    Name = "Public Subnet us-east-1b"
  }
}

resource "aws_internet_gateway" "my_vpc_igw" {
  vpc_id = aws_vpc.my_vpc.id

  tags = {
    Name = "My VPC - Internet Gateway"
  }
}

resource "aws_route_table" "my_vpc_public" {
    vpc_id = aws_vpc.my_vpc.id

    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.my_vpc_igw.id
    }

    tags = {
        Name = "Public Subnets Route Table for My VPC"
    }
```

```
}

resource "aws_route_table_association" "my_vpc_us_east_1a_public" {
    subnet_id = aws_subnet.public_us_east_1a.id
    route_table_id = aws_route_table.my_vpc_public.id
}

resource "aws_route_table_association" "my_vpc_us_east_1b_public" {
    subnet_id = aws_subnet.public_us_east_1b.id
    route_table_id = aws_route_table.my_vpc_public.id
```

Next, we need to describe the Security Group for our web-servers, which will allow HTTP connections to our instances:

```
resource "aws_security_group" "allow_http" {
  name        = "allow_http"
  description = "Allow HTTP inbound connections"
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    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 = "Allow HTTP Security Group"
  }
```

Launch configuration

```
resource "aws_launch_configuration" "web" {
  name_prefix = "web-"

  image_id = "ami-0947d2ba12ee1ff75" # Amazon Linux 2 AMI (HVM), SSD
Volume Type
  instance_type = "t2.micro"
  key_name = "Lenovo T410"

  security_groups = [ aws_security_group.allow_http.id ]
  associate_public_ip_address = true

  user_data = < /usr/share/nginx/html/index.html
chkconfig nginx on
service nginx start
  USER_DATA

  lifecycle {
    create_before_destroy = true
  }
}
```

The new ones are a `user_data` and a lifecycle:

- `user_data` – is a special interface created by AWS for EC2 instances automation. Usually this option is filled with scripted instructions to the instance, which need to be executed at the instance boot time.
- `lifecycle` – special instruction, which is declaring how new launch configuration rules applied during update. We're using `create_before_destroy` here to create new instances from a new launch configuration before destroying the old ones. This option commonly used during rolling deployments.

The `user-data` option is filled with a simple bash-script, which installs the Nginx web server and puts the instance's local IP address to the index.html file, so we can see it after the instance is up and running.

Load balancer

Before we create an Auto Scaling Group, we need to declare a Load Balancer. There are three Load Balances available for you in AWS right now:

- Elastic or Classic Load Balancer (ELB) – previous generation of Load Balancers in AWS.
- Application Load Balancer (ALB) – operates on application network layer and provides reach feature set to manage HTTP and HTTPS traffic for your web applications.
- Network Load Balancer (NLB) – operates on connection layer and capable for handling millions of requests per second.

For simplicity, let's create an Elastic Load Balancer in front of our EC2 instances (I'll show how to use other types of them in future articles).

```
resource "aws_security_group" "elb_http" {
  name        = "elb_http"
  description = "Allow HTTP traffic to instances through Elastic Load
Balancer"
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    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 = "Allow HTTP through ELB Security Group"
  }
}

resource "aws_elb" "web_elb" {
  name = "web-elb"
  security_groups = [
    aws_security_group.elb_http.id
  ]
  subnets = [
    aws_subnet.public_us_east_1a.id,
    aws_subnet.public_us_east_1b.id
  ]

  cross_zone_load_balancing   = true

  health_check {
    healthy_threshold = 2
    unhealthy_threshold = 2
    timeout = 3
    interval = 30
    target = "HTTP:80/"
  }

  listener {
    lb_port = 80
    lb_protocol = "http"
    instance_port = "80"
    instance_protocol = "http"
  }

}
```

Here we're setting up Load Balancer `name`, it's own Security Group, so we could make traffic rules more restrictive later if we want to.

We're specifying 2 `subnets`, where our Load Balancer will look for (`listener` configuration) launched instances and turned on `cross_zone_load_balancing` feature, so we could have our instances in different Availability Zones.

And finally, we've specified `health_check` configuration, which determines when Load Balancer should transition instances from healthy to unhealthy state and back depending on its ability to reach HTTP port 80 on the target instance.

If ELB can not reach the instance on the specified port, it will stop sending traffic.

Auto scaling group

Now we're ready to create an Auto Scaling Group by describing it using aws_autoscaling_group resource:

```
resource "aws_autoscaling_group" "web" {
  name = "${aws_launch_configuration.web.name}-asg"

  min_size             = 1
  desired_capacity     = 2
  max_size             = 4

  health_check_type    = "ELB"
  load_balancers = [
    aws_elb.web_elb.id
  ]

  launch_configuration = aws_launch_configuration.web.name

  enabled_metrics = [
    "GroupMinSize",
    "GroupMaxSize",
    "GroupDesiredCapacity",
    "GroupInServiceInstances",
    "GroupTotalInstances"
  ]

  metrics_granularity = "1Minute"

  vpc_zone_identifier  = [
    aws_subnet.public_us_east_1a.id,
    aws_subnet.public_us_east_1b.id
  ]

  # Required to redeploy without an outage.
  lifecycle {
    create_before_destroy = true
  }

  tag {
    key                 = "Name"
    value               = "web"
    propagate_at_launch = true
  }

}
```

Here we have the following configuration:

- There will be minimum one instance to serve the traffic.
- Auto Scaling Group will be launched with 2 instances and put each of them in separate Availability Zones in different Subnets.
- Auto Scaling Group will get information about instance availability from the `ELB`.
- We're set up collection for some Cloud Watch metrics to monitor our Auto Scaling Group state.
- Each instance launched from this Auto Scaling Group will have `Name` tag set to `web`.

Now we are almost ready, let's get the Load Balancer DNS name as an output from the Terraform infrastructure description:

```
output "elb_dns_name" {
  value = aws_elb.web_elb.dns_name
}
```

And try to deploy our infrastructure:

```
terraform init
terraform plan
terraform apply
```

Starting from this point, you can open provided ELB URL in your browser and refresh the page several times to see different local IP addresses of your just launched instances.

Auto scaling policies

But this configuration is static. We discussed no rules at the top of the article, which will add or remove instances based on specific metrics.

To make our infrastructure dynamic, we need to create several Auto Scaling Policies and CloudWatch Alarms.

First, let's determine how AWS need to scale our group UP by declaring aws_autoscaling_policy and aws_cloudwatch_metric_alarm resources:

```
resource "aws_autoscaling_policy" "web_policy_up" {
  name = "web_policy_up"
  scaling_adjustment = 1
  adjustment_type = "ChangeInCapacity"
  cooldown = 300
  autoscaling_group_name = aws_autoscaling_group.web.name
}

resource "aws_cloudwatch_metric_alarm" "web_cpu_alarm_up" {
  alarm_name = "web_cpu_alarm_up"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = "2"
  metric_name = "CPUUtilization"
  namespace = "AWS/EC2"
  period = "120"
  statistic = "Average"
  threshold = "60"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.web.name
  }

  alarm_description = "This metric monitor EC2 instance CPU utilization"
  alarm_actions = [ aws_autoscaling_policy.web_policy_up.arn ]
}
```

`aws_autoscaling_policy` defines how AWS should change Auto Scaling Group instances count in case of `aws_cloudwatch_metric_alarm`.

`cooldown` option is needed to give our infrastructure some time (300 seconds) before increasing Auto Scaling Group again.

`aws_cloudwatch_metric_alarm` is a straightforward alarm, which will be fired, if the total CPU utilization of all instances in our Auto Scaling Group is greater or equal `threshold` (60% CPU utilization) during 120 seconds.

Pretty much the same resources we need to declare to scale our Auto Scaling Group down:

```
resource "aws_autoscaling_policy" "web_policy_down" {
  name = "web_policy_down"
  scaling_adjustment = -1
  adjustment_type = "ChangeInCapacity"
  cooldown = 300
  autoscaling_group_name = aws_autoscaling_group.web.name
}

resource "aws_cloudwatch_metric_alarm" "web_cpu_alarm_down" {
  alarm_name = "web_cpu_alarm_down"
  comparison_operator = "LessThanOrEqualToThreshold"
  evaluation_periods = "2"
  metric_name = "CPUUtilization"
  namespace = "AWS/EC2"
  period = "120"
  statistic = "Average"
  threshold = "10"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.web.name
  }

  alarm_description = "This metric monitor EC2 instance CPU utilization"
  alarm_actions = [ aws_autoscaling_policy.web_policy_down.arn ]
}
```

Here we're decreasing Auto Scaling Group size by one instance every 300 seconds if its total CPU utilization is less or equals 10%.

Apply these rules by running the following commands:

```
terraform plan
terraform apply
```

TF FILE

```
resource "aws_vpc" "my_vpc" {
  cidr_block       = "10.0.0.0/16"
  enable_dns_hostnames = true

  tags = {
    Name = "My VPC"
  }
}

resource "aws_subnet" "public_us_east_1a" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "Public Subnet us-east-1a"
  }
}

resource "aws_subnet" "public_us_east_1b" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1b"

  tags = {
    Name = "Public Subnet us-east-1b"
  }
}

resource "aws_internet_gateway" "my_vpc_igw" {
  vpc_id = aws_vpc.my_vpc.id

  tags = {
    Name = "My VPC - Internet Gateway"
  }
}

resource "aws_route_table" "my_vpc_public" {
    vpc_id = aws_vpc.my_vpc.id

    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.my_vpc_igw.id
    }

    tags = {
        Name = "Public Subnets Route Table for My VPC"
    }
}

resource "aws_route_table_association" "my_vpc_us_east_1a_public" {
    subnet_id = aws_subnet.public_us_east_1a.id
    route_table_id = aws_route_table.my_vpc_public.id
}

resource "aws_route_table_association" "my_vpc_us_east_1b_public" {
    subnet_id = aws_subnet.public_us_east_1b.id
    route_table_id = aws_route_table.my_vpc_public.id
}
```

```
resource "aws_security_group" "allow_http" {
  name        = "allow_http"
  description = "Allow HTTP inbound connections"
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    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 = "Allow HTTP Security Group"
  }
}

resource "aws_launch_configuration" "web" {
  name_prefix = "web-"

  image_id = "ami-0947d2ba12ee1ff75" # Amazon Linux 2 AMI (HVM), SSD
Volume Type
  instance_type = "t2.micro"
  key_name = "Lenovo T410"

  security_groups = [ aws_security_group.allow_http.id ]
  associate_public_ip_address = true

  user_data = <<USER_DATA
#!/bin/bash
yum update
yum -y install nginx
echo "$(curl http://169.254.169.254/latest/meta-data/local-ipv4)" > /usr
/share/nginx/html/index.html
chkconfig nginx on
service nginx start
  USER_DATA

  lifecycle {
    create_before_destroy = true
  }
}
```

```
resource "aws_security_group" "elb_http" {
  name        = "elb_http"
  description = "Allow HTTP traffic to instances through Elastic Load
Balancer"
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    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 = "Allow HTTP through ELB Security Group"
  }
}

resource "aws_elb" "web_elb" {
  name = "web-elb"
  security_groups = [
    aws_security_group.elb_http.id
  ]
  subnets = [
    aws_subnet.public_us_east_1a.id,
    aws_subnet.public_us_east_1b.id
  ]

  cross_zone_load_balancing   = true

  health_check {
    healthy_threshold = 2
    unhealthy_threshold = 2
    timeout = 3
    interval = 30
    target = "HTTP:80/"
  }

  listener {
    lb_port = 80
    lb_protocol = "http"
    instance_port = "80"
    instance_protocol = "http"
  }

}

resource "aws_autoscaling_group" "web" {
  name = "${aws_launch_configuration.web.name}-asg"

  min_size             = 1
  desired_capacity     = 2
  max_size             = 4

  health_check_type    = "ELB"
  load_balancers = [
    aws_elb.web_elb.id
  ]

  launch_configuration = aws_launch_configuration.web.name

  enabled_metrics = [
    "GroupMinSize",
    "GroupMaxSize",
    "GroupDesiredCapacity",
    "GroupInServiceInstances",
    "GroupTotalInstances"
  ]

  metrics_granularity = "1Minute"

  vpc_zone_identifier  = [
    aws_subnet.public_us_east_1a.id,
    aws_subnet.public_us_east_1b.id
  ]

  # Required to redeploy without an outage.
  lifecycle {
    create_before_destroy = true
  }

  tag {
    key                 = "Name"
    value               = "web"
    propagate_at_launch = true
  }

}

resource "aws_autoscaling_policy" "web_policy_up" {
  name = "web_policy_up"
  scaling_adjustment = 1
  adjustment_type = "ChangeInCapacity"
  cooldown = 300
  autoscaling_group_name = aws_autoscaling_group.web.name
}

resource "aws_cloudwatch_metric_alarm" "web_cpu_alarm_up" {
  alarm_name = "web_cpu_alarm_up"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = "2"
  metric_name = "CPUUtilization"
  namespace = "AWS/EC2"
  period = "120"
  statistic = "Average"
  threshold = "60"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.web.name
  }

  alarm_description = "This metric monitor EC2 instance CPU utilization"
  alarm_actions = [ aws_autoscaling_policy.web_policy_up.arn ]
}

resource "aws_autoscaling_policy" "web_policy_down" {
  name = "web_policy_down"
  scaling_adjustment = -1
  adjustment_type = "ChangeInCapacity"
  cooldown = 300
  autoscaling_group_name = aws_autoscaling_group.web.name
}

resource "aws_cloudwatch_metric_alarm" "web_cpu_alarm_down" {
  alarm_name = "web_cpu_alarm_down"
  comparison_operator = "LessThanOrEqualToThreshold"
  evaluation_periods = "2"
  metric_name = "CPUUtilization"
  namespace = "AWS/EC2"
  period = "120"
  statistic = "Average"
  threshold = "10"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.web.name
  }

  alarm_description = "This metric monitor EC2 instance CPU utilization"
  alarm_actions = [ aws_autoscaling_policy.web_policy_down.arn ]
}

output "elb_dns_name" {
  value = aws_elb.web_elb.dns_name
```

```
    }
```