Lab5b. Application Load Balancer with Autoscaling

Contents

Overview
This lab will deploy an application load balancer to distribute web traffic across an auto-scaling group of EC2 instances.

Solution
The solution to this lab can be found in awslabs/solutions/05b. Try to use this only as a last resort if you are struggling to complete the step-by-step processes.

Setup

1. Ensure you have completed Lab0 and Lab5a before attempting this lab.

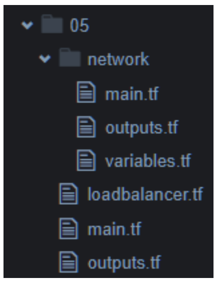
2. In the IDE terminal pane, enter the following commands…

   cd ~/environment/awslabs/05

3. This shifts your current working directory to awslabs/labs/05. Ensure all commands are executed in this directory

4. Close any open files and use the Explorer pane to navigate to and open the awslabs/05 folder.

5. Take note of the contents of this folder. There are 3 files at the root; maint.tf, loadbalancer.tf and outputs.tf. There is a subfolder named network with 3 files; main.tf, outputs.tf and variables.tf….

   ```
   05
     network
       main.tf
       outputs.tf
       variables.tf
     loadbalancer.tf
     main.tf
     outputs.tf
   ```

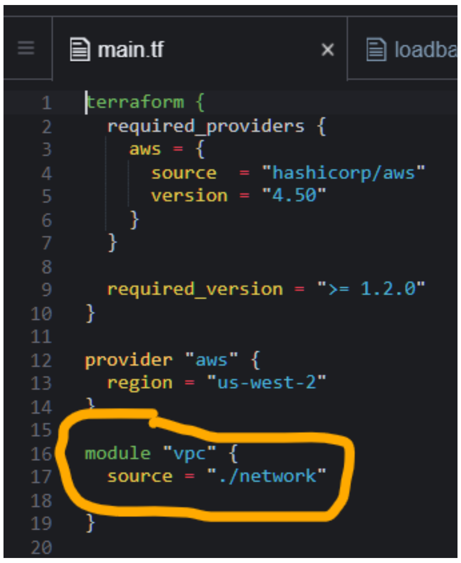
   In Lab05a you reconfigured this terraform configuration to use a remote backend, and you migrated your statefile to S3.

6. Re-deploy the resources using **terraform apply**

   You have now (re)deployed a VPC into the us-west-2 region. There is a public and private subnet in each of the 3 availability-zones. There is a public routing table which uses an internet gateway and a private routing table which uses a NAT Gateway. The subnets have been associated with these routing tables. There are 2 security groups, one for ec2 instances, the other for load-balancers.

The Challenge

You have been provided with a code module named **vpc**, (the local network folder), that deploys all necessary networking components…



```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "4.50"
    }
  }

  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-west-2"
}

module "vpc" {
  source = "./network"

}
```

You are required to deploy the front-facing layer of an n-tier solution into this network utilizing parameters exposed from this module.

This will be an http web service hosted on linux ec2 instances. For resilience, multiple web instances must be deployed, distributed across availability zones.

For cost efficiency, there should be a minimum of 2 instances running at any time, automatically increasing to a maximum of 5 should demand dictate. An autoscaling group should be used to achieve this.

Instances should not be allocated public ip addresses and should be isolated from the internet.

Traffic should be distributed across the running instances using an Application Load Balancer.

Successful completion of the challenge will be the generation of an http url which, when accessed, will display a message and an indication as to which instance the page is being served from.

**** You are not to modify any files that currently contain code. This simulates an environment whereby modules have been pre-created and are 'locked-down' ****

You have the choice of attempting all tasks yourself or following detailed step-by-step instructions.

Lab Objectives

1. Write all build code to the empty **loadbalancer.tf** or **outputs.tf** files as modification to existing code files is not permitted.

2. Create a data source **aws_ami.amazon-linux-2** which will return the most recent amazon owned ami (filter search on amzn2-ami-kernel-5.10-*-x86_64-ebs)

3. Create a **userdata.sh** file populated with ...

```
#!/bin/bash
sudo yum update -y
sudo yum install -y httpd.x86_64
sudo systemctl start httpd.service
sudo systemctl enable httpd.service
sudo chmod -R 777 /var/www/html
sudo echo “Welcome from $(hostname -f)” >/var/www/html/index.html
sudo systemctl restart httpd.service
```

4. Create a launch template resource **aws_launch_template** with the following settings...

   resource name: lab_launch_template1
   name: “lab-launch-template1”
   image_id: data.aws_ami.amazon-linux-2.id
   instance_type: “t3.micro”
   vpc_security_group_ids: [module.vpc.ec2instance-sg]
   user_data: filebase64(“userdata.sh”)

5. Create an Auto Scaling Group **aws_autoscaling_group.lab_asg** configured to use launch template **aws_launch_template.lab_launch_template1**. Set `**desired_capacity**` as `**2**`, `**max_size**` as `**5**` and `**min_size**` as `**2**` The instances created should be named **Lab ASG member**

6. Create an Application Load Balancer resource **aws_lb.lab_alb** configured to distribute traffic across the subnets **public-subnets** and using the security group **loadbal-sg**, both of which are defined in the vpc module. Do not define the listener or target group at this stage.

7. Create an **HTTP:80** target group **aws_lb_target_group.alb_targetgroup**. Name the target group **backend-target-group**.

8. Create an **HTTP:80** listener **aws_lb_listener.alb_listener** as a forwarder to target group **aws_lb_target_group.alb_targetgroup** Associate the listener with load balancer **aws_lb.lab_alb**

9. Create an autoscaling attachment **aws_autoscaling_attachment.lab_asg_attachment** that attaches the ASG **aws_autoscaling_group.lab_asg** with the target group **aws_lb_target_group.alb_targetgroup**

10. Output the URL of the load balancer for testing.

Try it yourself

Based on the above objectives, you are encouraged to attempt this lab yourself if you feel confident to do so. Refer to the proposed solution for guidance if necessary.

Step-by-Step

Task1. Creating an AWS Auto Scaling Group

Step1. Create a Launch Template

*A **Launch Template** is a resource in that provides a blueprint for launching Amazon EC2 instances. It allows you to predefine key instance configuration details, which can then be reused whenever you create or scale EC2 instances. Launch Templates simplify instance management and make it easier to create and maintain consistent instance configurations across environments.*

1. Our EC2 instances require a web service to be installed. This can be achieved by passing a script file into the instance at start-up. Create a file for this purpose, name it **userdata.sh**....

```
touch userdata.sh
```

2. Populate **userdata.sh** with the following commands. Save the changes.

```
#!/bin/bash
sudo yum update -y
sudo yum install -y httpd.x86_64
sudo systemctl start httpd.service
sudo systemctl enable httpd.service
sudo chmod -R 777 /var/www/html
sudo echo “Welcome from $(hostname -f)” >/var/www/html/index.html
sudo systemctl restart httpd.service
```

3. To retrieve the latest ami to use for our instances, refer to the Terraform AWS registry for information on Data Source: aws_ami

   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami

4. Copy the example into **loadbalancer.tf** and rename it from **example** to **amazon_linux_2**

5. Modify the data block by removing all filters except **name** and leaving just **owners** and **most_recent** main arguments. Set owners to **“amazon”** and set the filter values to **“amzn2-ami-kernel-5.10-*-x86_64-ebs”**

   This fetches the most recent Amazon Linux 2 AMI with kernel version 5.10, provided by Amazon, ensuring it meets specific criteria such as 64-bit x86 architecture and EBS-backed storage. By querying AWS's official AMI repository, it guarantees the use of a secure and up-to-date image for creating EC2 instances. This setup is particularly useful for deploying consistent environments, automating infrastructure provisioning, and ensuring compatibility with Amazon Linux 2’s latest updates and features.

6. Refer to the Terraform AWS registry for information on resource type **aws_launch_template**..

   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/launch_template

   The example contains many optional arguments that we will not be adopting. A launch template can be created with as few as 2 mandatory arguments; a name and an associated security group. All other arguments can be provided at the time the template is invoked. In our case though, we want to mandate...

   name: lab-launch-template1
   ami: latest version of amazon linux image
   instance type: t3.micro

security group: ec2instance security group from vpc module
user data: userdata.sh file

7. Create a boiler-plate resource block by pasting the code below into loadbalancer.tf

   resource "aws_launch_template" "lab_launch_template1" {}

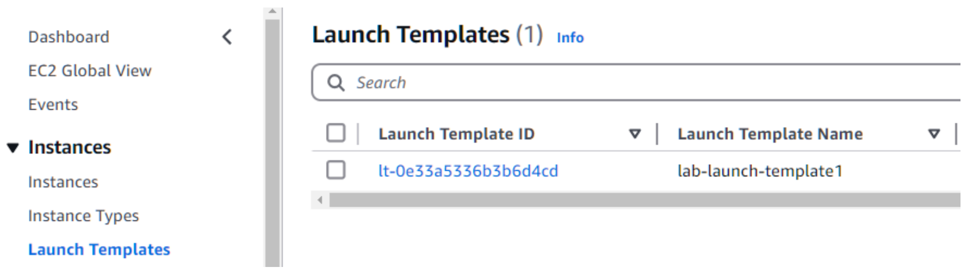
8. ***Referencing the registry documentation for the correct argument names***, populate the block with the following data;

   name: "lab-launch-template1"
   ami: data.aws_ami.amazon_linux_2.id
   instance: "t3.micro"
   security group: [module.vpc.ec2instance-sg] **(no quotes!)**
   user data: filebase64("userdata.sh")

9. Save any file changes and run **terraform plan** to identify any possible issues.

10. Resolve any issues and then run **terraform apply** to deploy the launch template.

11. Use the console, navigating to the EC2 service for Oregon, to verify the creation of the launch template….



Launch Template ID	Launch Template Name
lt-0e33a5336b3b6d4cd	lab-launch-template1

*This block creates an AWS Launch Template named **lab-launch-template1**. It specifies the latest Amazon Linux 2 AMI, t3.micro instance type, and a security group ID passed as a parameter from the VPC module. It also includes a base64-encoded user data script (**userdata.sh**) for instance initialization. This template provides a reusable configuration for launching EC2 instances with consistent settings.*

Step2. Create an Auto Scaling Group

An Auto Scaling Group (ASG) is a service that automatically manages the scaling of Amazon EC2 instances to ensure that an application always has the right number of instances to handle its workload. ASGs dynamically adjust the number of instances based on demand, policies, and metrics, helping to maintain performance while optimizing costs

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/autoscaling_group

1. Refer to the aws registry for information on resource type **aws_autoscaling_group**

2. Refer to the **2nd** sample code block, the code relating to **Latest Version Of Launch Template**. Copy the **lower** block of this example code, as we have already defined our launch template.

3. Rename the resource block from **bar** to **lab_asg**

   The VPC module code we are using outputs the subnets created within the vpc rather than the availability zones. The documentation tells us we can either deploy instances to specified AZs or to subnets. We need to distribute our instances across the 3 private subnets (they must isolated from the internet remember) and therefore need to use the **vpc_zone_identifier** argument instead of **availability_zones**

4. Remove the **availability_zone** argument, replacing it with...

   vpc_zone_identifier = [for subnet in module.vpc.private-subnets : subnet.id]

5. Set `**desired_capacity**` as **2**, `**max_size**` as **5** and `**min_size**` as **2**

Question: Why do you think we are mandating that at least 2 instances are always running. Consider and discuss later.

6. Update the launch_template.id argument value to reference the launch template we defined in Task 1 ...

   id = aws_launch_template.lab_launch_template1.id

7. The resource block thus far would be sufficient to deploy the ASG, but all EC2 instances created would be un-named and we have not considered the points mentioned in the documentation regarding the use of this ASG in conjunction with load-balancers. Add the following sections to the block...

```
tag {
  key = "Name"
  value = "Lab ASG member"
  propagate_at_launch = true
 }

lifecycle {
  ignore_changes = [load_balancers, target_group_arns]
 }
```

8. Save\Plan\Apply and use the console to verify that the Auto Scaling Group with 2 EC2 instances named 'Lab ASG member' are created.

Summary of what we have done so far..

Fetched the Latest Amazon Linux 2 AMI:
The aws_ami data block queries AWS for the most recent Amazon Linux 2 AMI with kernel version 5.10, provided by Amazon, and stores its ID for use in instance creation.

Defined a Launch Template:
The aws_launch_template resource creates a reusable configuration for EC2 instances, specifying:
The latest Amazon Linux 2 AMI.
A t3.micro instance type.
A security group provided as a parameter from the VPC module.
A base64-encoded userdata.sh script for instance initialization.

Created an Auto Scaling Group (ASG):
The aws_autoscaling_group resource manages EC2 instances in private subnets provided by the VPC module, ensuring:
A minimum of 2 instances.
A maximum of 5 instances.
A desired capacity of 2 instances.
It uses the defined launch template for instance configuration.
It Tags instances with Name = "Lab ASG member" to identify them.
It ignores changes to load_balancers and target_group_arns to prevent unnecessary redeployments.

Task2. Creating an Application Load Balancer

Step1. Create the basic load balancer

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lb

1. Refer to the Terraform AWS registry for information on resource type **aws_lb**

2. Copy the **Application Load Balancer** example code

3. Remove the **enable_deletion_protection**, **access_logs,** and **tags** sections

4. Rename the resource block from **test** to **lab_alb** and the resource name from **test-lb-tf** to **lab-alb**

5. Modify the security_groups value to use **[module.vpc.loadbal-sg]**

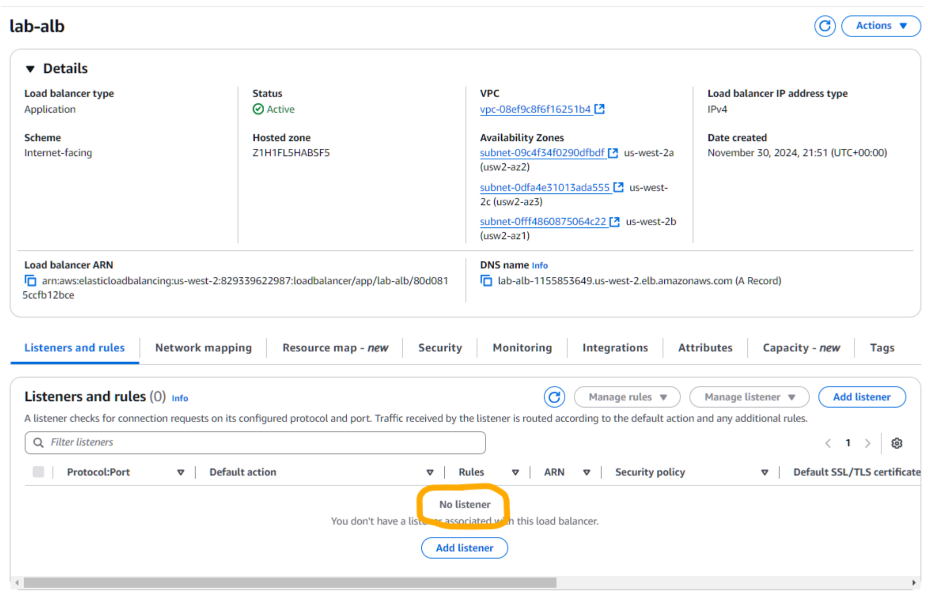
   Note. loadbal-sg is an output from the vpc module. It **is** the id of the security group; therefore we do not have to append **.id** to it as the sample code does. Look at the **lab5/network/outputs.tf** file to verify this.

6. Modify the **subnets** value to reference our public subnets; **[for subnet in module.vpc.public-subnets : subnet.id]**

   Note. Here public-subnets is also an output from the vpc, but in this case it is the name of the subnets and therefore we **do** need the **.id** component

7. **Save/Plan/Apply** to deploy the load balancer.

8. Use the console to verify the creation of the Application Load Balancer. Use the Network mapping tab to verify the load balancer is mapped across all 3 public subnets. The load balancer currently has no listener and therefore is non-functional as yet.


lab-alb

Details

Load balancer type Application	**Status** Active	**VPC** vpc-08ef9c8f6f16251b4	**Load balancer IP address type** IPv4
Scheme Internet-facing	**Hosted zone** Z1H1FL5HABSF5	**Availability Zones** subnet-09c4f34f0290dfbdf us-west-2a (usw2-az2); subnet-0dfa4e31013ada555 us-west-2c (usw2-az3); subnet-0fff4860875064c22 us-west-2b (usw2-az1)	**Date created** November 30, 2024, 21:51 (UTC+00:00)

Load balancer ARN
arn:aws:elasticloadbalancing:us-west-2:829339622987:loadbalancer/app/lab-alb/80d0815ccfb12bce

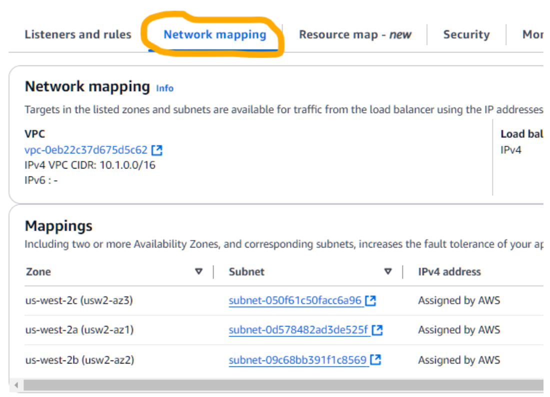
DNS name Info
lab-alb-1155853649.us-west-2.elb.amazonaws.com (A Record)

Listeners and rules (0) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate

No listener
You don't have a listener associated with this load balancer.



Network mapping Info

Targets in the listed zones and subnets are available for traffic from the load balancer using the IP addresses

VPC
vpc-0eb22c37d675d5c62
IPv4 VPC CIDR: 10.1.0.0/16
IPv6 : -

Load bal
IPv4

Mappings

Including two or more Availability Zones, and corresponding subnets, increases the fault tolerance of your ap

Zone	Subnet	IPv4 address
us-west-2c (usw2-az3)	subnet-050f61c50facc6a96	Assigned by AWS
us-west-2a (usw2-az1)	subnet-0d578482ad3de525f	Assigned by AWS
us-west-2b (usw2-az2)	subnet-09c68bb391f1c8569	Assigned by AWS

*This block created a public-facing **Application Load Balancer (ALB)** named **lab-alb** to complement the previously defined infrastructure. The ALB is deployed into the public subnets of the VPC (defined in the module.vpc) and secured with a security group (**loadbal-sg**) to control access. While the ALB is set up, it currently lacks a listener and target group, so it won't yet route traffic to the EC2 instances managed by the Auto Scaling Group. This block serves as the foundation for load balancing traffic in the deployment, requiring additional configuration for full functionality.*

Step2. Create the listener and the target group

An Application Load Balancer needs at least one listener to listen out on for incoming traffic and then direct it to a target group. In our case we want to send http traffic to the load balancer and have it distributed across our instances in the Auto Scaling Group.

Refer to the aws registry for information on resource type **aws_lb_listener**

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lb_listener

1. Copy the resource block **aws_lb_listener** from the **Example Usage, Forward Action** examples. We do not need the **aws_lb** or **aws_lb_target_group** blocks to be copied.

2. Rename the resource block from "front_end" to "**alb_listener"**

3. We will be using HTTP not HTTPS so remove the **ssl_policy** and **certificate_arn** arguments. Change the **port** to **80** and the **protocol** to **HTTP**

4. Change the load_balancer_arn to **aws_lb.lab_alb.arn**

5. Change the target_group_arn to **aws_lb_target_group.alb_targetgroup.arn**. This is the target group we will create next.

6. Refer to the Terraform AWS registry for information on resource type **aws_lb_target_group**

   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lb_target_group

7. Copy the resource block **aws_lb_target_group** from the **Example Usage, Instance Target Group** examples. We do not need the **aws_vpc** code.

8. Change the resource block name to **alb_targetgroup**

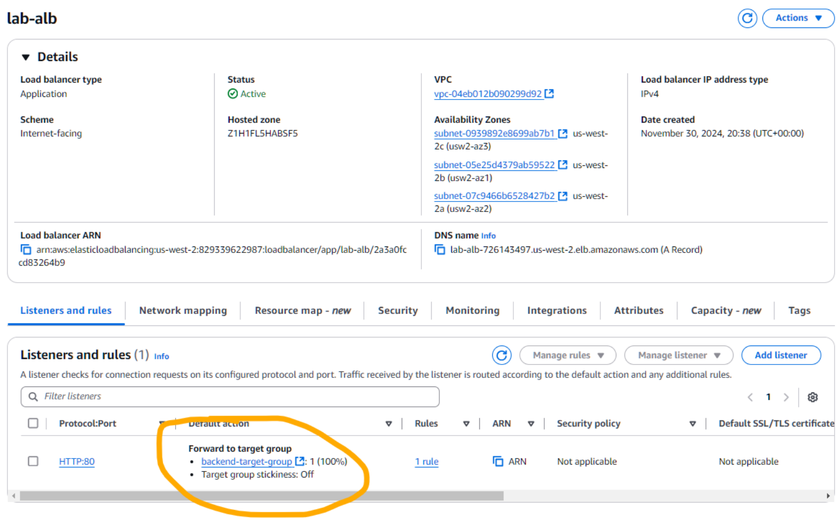
9. Change the resource name to **backend-target-group**

10. Change the vpc_id argument value to **module.vpc.vpc-id**

11. Save\Plan\Apply and verify the deployment of the listener and the target group. Note that whilst the load balancer now shows an attached listener with a target group, the target group is currently empty. We will associate our ASG with the target group next.

    *These two blocks set up the routing mechanism for the Application Load Balancer (ALB) but currently leave the target group without explicitly registered targets. The **aws_lb_target_group** block creates a logical grouping (**backend-target-group**) that listens for HTTP traffic on port 80 within the specified VPC. The **aws_lb_listener** block attaches a listener to the ALB, enabling it to accept HTTP traffic on port 80 and forward it to this target group. While the target group is defined, it will remain empty until targets (the EC2 instances managed by the Auto Scaling*

Group) are explicitly registered or automatically integrated via additional configuration. Together, these blocks provide the framework for load balancing, requiring the target group to be populated for full functionality.



lab-alb

Actions

Details

Load balancer type
Application

Scheme
Internet-facing

Status
Active

Hosted zone
Z1H1FL5HABSF5

VPC
vpc-04eb012b090299d92

Availability Zones
subnet-0939892e8699ab7b1 us-west-2c (usw2-az3)
subnet-05e25d4379ab59522 us-west-2b (usw2-az1)
subnet-07c9466b6528427b2 us-west-2a (usw2-az2)

Load balancer IP address type
IPv4

Date created
November 30, 2024, 20:38 (UTC+00:00)

Load balancer ARN
arn:aws:elasticloadbalancing:us-west-2:829339622987:loadbalancer/app/lab-alb/2a3a0fccd83264b9

DNS name Info
lab-alb-726143497.us-west-2.elb.amazonaws.com (A Record)

Listeners and rules | Network mapping | Resource map - *new* | Security | Monitoring | Integrations | Attributes | Capacity - *new* | Tags

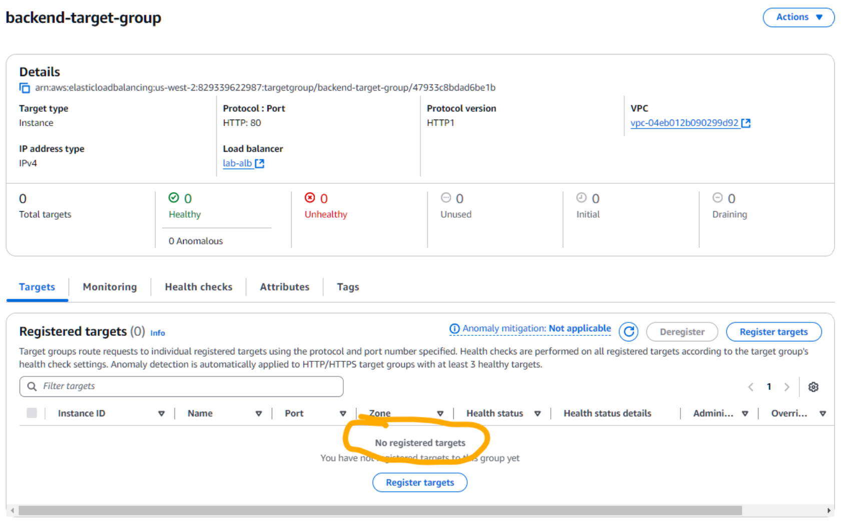
Listeners and rules (1) Info

Manage rules | Manage listener | Add listener

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Filter listeners

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate
HTTP:80	Forward to target group • backend-target-group: 1 (100%) • Target group stickiness: Off	1 rule	ARN	Not applicable	Not applicable



backend-target-group

Actions

Details
arn:aws:elasticloadbalancing:us-west-2:829339622987:targetgroup/backend-target-group/47933c8bdad6be1b

Target type
Instance

IP address type
IPv4

Protocol : Port
HTTP: 80

Load balancer
lab-alb

Protocol version
HTTP1

VPC
vpc-04eb012b090299d92

0	0	0	0	0	0
Total targets	Healthy 0 Anomalous	Unhealthy	Unused	Initial	Draining

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (0) Info

Anomaly mitigation: Not applicable | Deregister | Register targets

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Filter targets

Instance ID	Name	Port	Zone	Health status	Health status details	Admini...	Overri...

No registered targets
You have not registered targets to this group yet

Register targets

Task3. Link the Auto Scaling Group and the Application Load Balancer

1. Refer to the aws registry for information on resource type **aws_autoscaling_attachment**

   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/autoscaling_attachment

2. We are dealing with an ALB Target Group so copy the 2nd Example Usage sample block **'Create a new ALB Target Group attachment'**

3. Rename the resource block from **example** to **lab_asg_attachment**

4. Change the autoscaling_group_name to **aws_autoscaling_group.lab_asg.id**

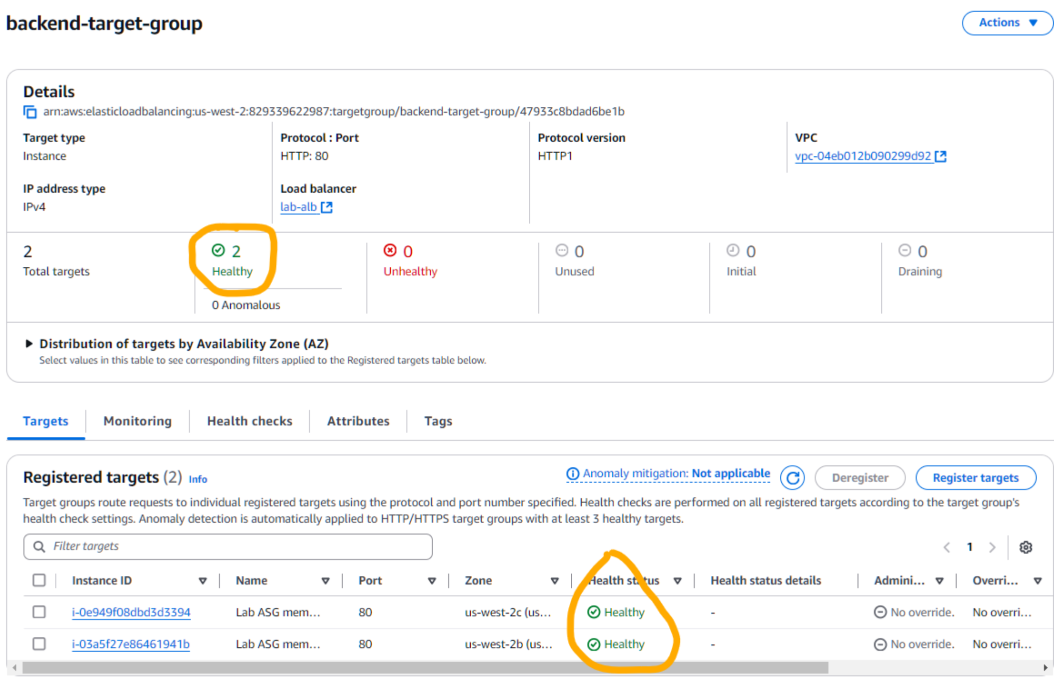
5. Change the lb_target_group_arn to **aws_lb_target_group.alb_targetgroup.arn**

6. Save changes to **loadbalancer.tf**

7. Paste the following into **lab\outputs.tf** and save...

```
output "lb_endpoint" {
  value = "http://${aws_lb.lab_alb.dns_name}"
}
```

8. Save\Plan\Apply and verify that the ASG is now the load balancer target group. Navigate to the loadbalancer and confirm that the listener and target group are unchanged. Click on the target group and there should be 2 ASG instances showing as 'initializing' or 'Unhealthy'. Refresh until a ‘Healthy’ state is shown (this may take a few minutes) ....



backend-target-group

Details

arn:aws:elasticloadbalancing:us-west-2:829339622987:targetgroup/backend-target-group/47933c8bdad6be1b

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 80	HTTP1	vpc-04eb012b090299d92
IP address type	**Load balancer**		
IPv4	lab-alb		

2	2	0	0	0	0
Total targets	Healthy	Unhealthy	Unused	Initial	Draining
	0 Anomalous				

Distribution of targets by Availability Zone (AZ)
Select values in this table to see corresponding filters applied to the Registered targets table below.

Registered targets (2)

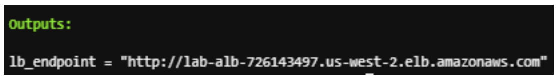
Anomaly mitigation: **Not applicable**

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Instance ID	Name	Port	Zone	Health status	Health status details	Admini...	Overri...
i-0e949f08dbd3d3394	Lab ASG mem...	80	us-west-2c (us...	Healthy	-	No override.	No overri...
i-03a5f27e86461941b	Lab ASG mem...	80	us-west-2b (us...	Healthy	-	No override.	No overri...

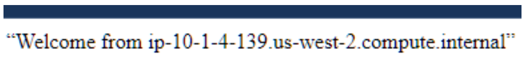
*The **aws_autoscaling_attachment** block bridges the Auto Scaling Group (**lab_asg**) and the target group (**alb_targetgroup**), ensuring that all EC2 instances launched by the ASG are* *automatically registered as targets in the ALB's backend target group. With this association, the ALB can now forward incoming HTTP traffic to the dynamically scaled instances managed by the ASG. This block ensures seamless communication between the scaling mechanism (ASG) and the load balancing mechanism (ALB), finalizing the infrastructure setup for a highly available and scalable application environment.*

9. Test the deployment by clicking on the lb_endpoint url. Refreshing your browser should rotate your traffic across the 2 backend instances.

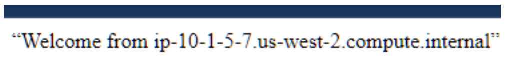


```
Outputs:

lb_endpoint = "http://lab-alb-726143497.us-west-2.elb.amazonaws.com"
```



"Welcome from ip-10-1-4-139.us-west-2.compute.internal"



"Welcome from ip-10-1-5-7.us-west-2.compute.internal"

10. Delete all resources using **terraform destroy**

Summary of VPC Module Integration:

> ***Private Subnets****: The Auto Scaling Group deploys its EC2 instances into private subnets obtained from the VPC module (****module.vpc.private-subnets****), ensuring the backend instances are secure and isolated.*
>
> ***Public Subnets****: The ALB is deployed in public subnets retrieved from the VPC module (****module.vpc.public-subnets****), allowing it to handle incoming internet traffic.*
>
> ***Security Groups****: Both the EC2 instances and the ALB use security groups (****ec2instance-sg and loadbal-sg****) provided by the VPC module to enforce strict access control rules.*
>
> ***VPC ID****: The target group is explicitly associated with the VPC using the* ***vpc-id*** *output from the VPC module, ensuring proper networking integration.*

****Congratulations, you have completed the lab****