Lab5b
Application Load Balancer with Autoscaling

Contents

Objectives

This lab will deploy a Global external application load balancer to distribute web traffic across an auto-scaling group of GCE instances.

Teaching Points

Solution

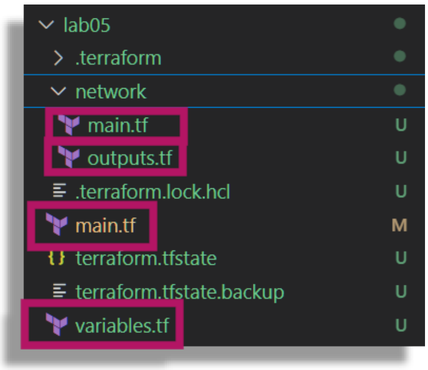
There is a Solution section at the very end of these instructions. Try to use this only as a last resort if you are struggling to complete the step-by-step tasks.

Before you begin

1. Ensure you have completed Lab0 and Lab5a before attempting this lab.
2. In the IDE terminal pane, enter the following commands…

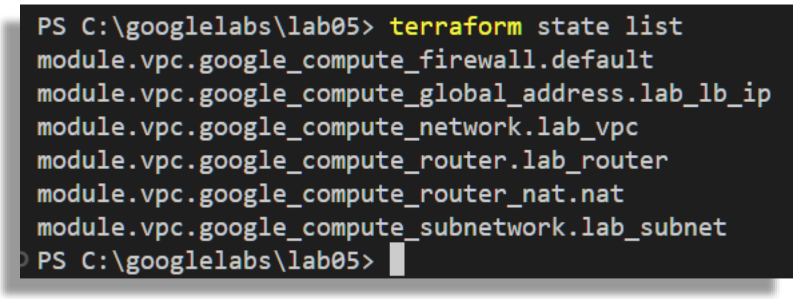
   cd ~/googlelabs/lab05

3. This shifts your current working directory to lab05. Ensure all commands are executed in this directory
4. Close any open files and use the Explorer pane to navigate to and open the googlelabs/lab05 folder.
5. Take note of the contents of this folder. There are 2 configuration files at the root level; maint.tf and variables.tf. There is a subfolder named network with 2 files; main.tf and outputs.tf….



```
lab05
  > .terraform
  v network
      main.tf                    U
      outputs.tf                 U
    .terraform.lock.hcl          U
    main.tf                      M
    terraform.tfstate            U
    terraform.tfstate.backup     U
    variables.tf                 U
```

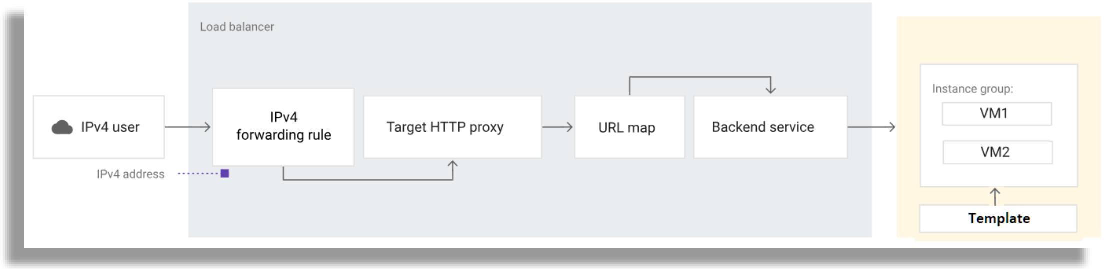
6. In Lab05a you reconfigured this terraform configuration to use a remote backend, and you migrated your statefile to GCS.
7. Run **terraform state list** to review the currently deployed resources..



```
PS C:\googlelabs\lab05> terraform state list
module.vpc.google_compute_firewall.default
module.vpc.google_compute_global_address.lab_lb_ip
module.vpc.google_compute_network.lab_vpc
module.vpc.google_compute_router.lab_router
module.vpc.google_compute_router_nat.nat
module.vpc.google_compute_subnetwork.lab_subnet
PS C:\googlelabs\lab05>
```

The Challenge

Having been provided with a module that deploys all the necessary networking components, you are required to deploy the front-facing layer of an n-tier solution into this network stack. This will be an http web service hosted on linux gce instances. Multiple web instances are to be deployed to a single zone, and there should be a minimum of 2 instances running at any time, automatically increasing to a maximum of 5 should demand dictate. An autoscaling managed instance group should be used to achieve this. Traffic should be distributed across the running instances using a global external application load balancer. Successful completion of the challenge will be the generation of an http url which, when accessed, will display a message indicating from which instance the page is being served from. You may add to but are not to modify any supplied code and the provider version to be used is 6.17.0.



Understanding the traffic flow between these components in Google Cloud Load Balancing is crucial for designing and managing your load balancer setup. A high-level explanation of the traffic flow:

Forwarding Rule: The Global Forwarding Rule is the entry point for external traffic. It listens for incoming requests from users or clients. It defines the IP protocol, port range etc and maps incoming traffic to the appropriate target, which is a Target HTTP Proxy or Target HTTPS Proxy.

Target Proxy: The Target Proxy receives incoming traffic from the Global Forwarding Rule. It determines how the incoming requests are mapped to backend services using URL Maps and forwards the requests to the appropriate URL Map.

URL Map: The URL Map defines how incoming requests are matched to backend services based on rules and conditions. It can route requests based on URL path, host, header, query parameters, etc and determines which backend service will handle the request.

Backend Service: The Backend Service defines a group of instances that can handle incoming requests. It may also include health checks to ensure the instances are healthy. The Backend Service handles distributing requests to the instances based on its configuration (e.g., round-robin, session affinity, etc.).

Instance Group / Serverless NEG: The backend instances are organized into an Instance Group or a Serverless Network Endpoint Group (NEG). An Instance Group can contain VM instances, while a Serverless NEG can contain VMs or other resources. These groups are responsible for managing the instances and distributing traffic according to the load balancing algorithm configured in the Backend Service.

Instance / Backend Instance: Instances are the actual virtual machines or resources that handle incoming requests. They run applications and services that users interact with. The load balancer sends traffic to instances based on the rules defined in the Backend Service and the load balancing algorithm. After processing the incoming request, the instance sends its response back to the load balancer.

References

https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_instance_template.html

https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_instance_group_manager

https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_autoscaler

https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_health_check.html

https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_global_forwarding_rule.html

Try it yourself

You have the option of attempting this lab yourself with minimal guidance. Use the step-by-step guides and solution folder if you need assistance.

The high-level lab goals are:

1. Create **template.tf**, and use it to create a **google_compute_instance_template.lab_instance_template** resource with the following settings...

   resource block identifier: lab_instance_template
   name: lab-launch-template
   size: e2-small
   instance: latest version of google cloud Debian linux image
   health checks: permit health checks to instances created from this template
   service account: use the default service account but with narrow scope
   instance metadata: apply the following startup script..

   ```
   "#! /bin/bash\n sudo apt-get update\n sudo apt-get install apache2 -y\n    sudo a2ensite
   default-ssl\n  sudo a2enmod ssl\n  vm_hostname=\"$(curl -H \"Metadata-Flavor:Google\"
   \\\n http://169.254.169.254/computeMetadata/v1/instance/name)\"\n   sudo echo
   \"Page served from: $vm_hostname\" | \\\n   tee /var/www/html/index.html\n   sudo
   systemctl restart apache2"
   ```

2. Create **managed-instance-group.tf,** and use it to create the following resources...

   google_compute_health_check.http_health_check
   google_compute_instance_group_manager.lb_backend_group
   google_compute_autoscaler.lab_autoscaler

3. The managed instance group should use your template, following the challenge requirements, and reside in **us-east1-b**. The **google_compute_health_check.http-health-check** should check for port **80** responses.

4. Create **loadbalancer.tf**, and use it to create the following resources...

   google_compute_backend_service.backend_service
   google_compute_url_map.http_url_map
   google_compute_target_http_proxy.http_proxy
   google_compute_global_forwarding_rule.http_forwarding

5. The **google_compute_backend_service.backend_service** backend should be the managed instance group from step 2.

6. The **google_compute_global_forwarding_rule** should receive http traffic on the **lab-lb-ip** IP address created in the network module.

7. Using **outputs.tf**, output the IP address of the load balancer to your screen for testing.

Task 1. Create Image Template

1. Open the first reference above and familiarize yourself with the resource type **compute_instance_template**

2. Our template will be used in conjunction with a managed instance group so scroll down to the **Using with Instance Group Manager** section.

3. Create a new file named **template.tf** and copy the example code for resource **google_compute_instance_template** into it. For convenience the code is shown below..

```
resource "google_compute_instance_template" "instance_template" {
  name_prefix  = "instance-template-"
  machine_type = "e2-medium"
  region       = "us-central1"

  // boot disk
  disk {
    # ...
  }

  // networking
  network_interface {
    # ...
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

4. Change the resource block identifier from "**instance-template**" to "**lab_instance_template**"

5. Replace the name_prefix "**instance-template-**" with **var.instance-prefix** Add this variable to **lab\variables.tf** with a default value of **"lab-template-"**

6. Replace the machine_type "**e2-medium**" with **var.machine-type**. Add this variable to **lab\variables.tf** with a default value of **"e2-small"**

7. Delete the `region` argument, because, reading the argument reference documentation regarding region, we see that if not specified, then the Provider region is used.

8. We need to add disk parameters for our template. Refer to the documentation for understanding and then paste the following code within the `disk` braces { }

   ```
   boot = true
   source_image = "projects/debian-cloud/global/images/family/debian-11"
   mode   = "READ_WRITE"
   type  = "PERSISTENT"
   ```

9. We need to add network interface parameters for our template. Refer to the documentation for understanding and then paste the following code within the `network_interface` braces { } Note the absence of an access_config sub-block, indicating we do not want the template to be given a public IP address.

   ```
   network    = module.vpc.vpc-id
   subnetwork = module.vpc.lab-subnet
   ```

   Note. These values are derived from the outputs of the networking module. They are found in lab05\network\outputs.tf
   Attempting to refer to the vpc directly using **google_compute_network.lab_vpc.id** as we have in the course up until now, would fail here as the root module is not aware of the details of the module resources. The same applies to the subnetwork.

10. A startup script has been provided that will install updates and Apache, configuring index.html to display a message that includes the vm name. Add the following code to the end of your resource block, inside the closing brace...

    ```
    metadata = {
    ```

```
  startup-script = "#! /bin/bash\n    sudo apt-get update\n    sudo apt-get install
apache2 -y\n    sudo a2ensite default-ssl\n    sudo a2enmod ssl\n
vm_hostname=\"$(curl -H \"Metadata-Flavor:Google\" \\\n
http://169.254.169.254/computeMetadata/v1/instance/name)\"\n   sudo echo
\"Page served from: $vm_hostname\" | \\\n   tee /var/www/html/index.html\n
sudo systemctl restart apache2"

}
```

11. This template will be used to create a managed instance group which will have health checks performed against them. Firewall rules must be applied to allow this and therefore we must use the tag already defined in the VPC firewall rules. Add the following block to the end of your resource block, inside the closing brace...

    ```
    tags = ["allow-health-check"]
    ```

12. Save changes to template.tf
13. Run **terraform plan** Address any issues found, referring to the solution section if necessary.
14. Run **terraform apply** and use the Console to verify the deployment of the template.

Task 2. Create Managed Instance Group

1. Create a new file named **managed-instance-group.tf** in the lab05 root directory
2. Open the 2nd, 3rd and 4th references above to familiarize yourself with the architecture surrounding managed instance groups.
3. Focussing on the 2nd reference;

   https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_instance_group_manager

   Copy the **google_compute_health_check** example resource block into **managed-instance-group.tf** For convenience the code is shown below..

```
resource "google_compute_health_check" "autohealing" {
  name                = "autohealing-health-check"
  check_interval_sec  = 5
  timeout_sec         = 5
```

```
  healthy_threshold   = 2
  unhealthy_threshold = 10 # 50 seconds

  http_health_check {
    request_path = "/healthz"
    port         = "8080"
  }
}
```

4. Change the resource block identifier from "autohealing" to "**lab_health_check**"
5. Change the health check name from "autohealing-health-check" to "**lab-http-check**"
6. Reading the Argument reference, you will find that the next 4 lines are optional parameters which, if unspecified, adopt default values. For code brevity, delete these lines.
7. Change the http health check request path from "/healthz" to "**/**"
8. Change the http health check port from "8080" to "**80**"
9. The modified health check block should now read as shown below...

```
resource "google_compute_health_check" "lab_health_check" {
  name            = "lab-http-check"
  http_health_check {
    port            = 80
    request_path    = "/"
  }
}
```

10. Copy the **google_compute_instance_group_manager** resource block example code into the bottom of **managed-instance-group.tf**
11. For convenience the code is shown below..

```
resource "google_compute_instance_group_manager" "appserver" {
  name = "appserver-igm"

  base_instance_name = "app"
  zone               = "us-central1-a"

  version {
    instance_template  =
google_compute_instance_template.appserver.self_link_unique
  }

  all_instances_config {
    metadata = {
      metadata_key = "metadata_value"
    }
    labels = {
      label_key = "label_value"
    }
  }

  target_pools = [google_compute_target_pool.appserver.id]
  target_size  = 2

  named_port {
    name = "customhttp"
    port = 8888
  }

  auto_healing_policies {
    health_check      = google_compute_health_check.autohealing.id
    initial_delay_sec = 300
  }
}
```

12. Change the resource block identifier from "**appserver**" to **"lab_mig"**

13. Change the name from "**appserver-igm**" to **"lab-mig"**

14. Change the base_instance_name from "**app**" to **"vm"**

15. Replace the zone value with variable var.lab-zone, declare this in **variables.tf** with a default value of **"us-east1-b"**

16. Change the instance template resource reference from **google_compute_instance_template.appserver.self_link_unique** to your template **google_compute_instance_template.lab_instance_template.self_link_unique**

17. Remove the **all_instances_config** block

18. Remove the **target_pools** argument (do not remove target_size argument)

19. Change the named_port name from "**customhttp**" to **"http"**

20. Change to port from 8888 to **80**

21. Change the health_check reference from **google_compute_health_check.autohealing.id** to your health check **google_compute_health_check.lab_health_check.id**

22. Change the initial delay to **120** seconds

23. The updated code should now be as shown below...

```
resource "google_compute_instance_group_manager" "lab_mig" {
  name = "lab-mig"

  base_instance_name = "vm"
  zone               = var.lab-zone

  version {
    instance_template  = google_compute_instance_template.lab_instance_template.self_link_unique
  }

  target_size  = 2

  named_port {
    name = "http"
    port = 80
  }

  auto_healing_policies {
    health_check      = google_compute_health_check.lab_health_check.id
    initial_delay_sec = 120
  }
}
```

24. From Ref 3; https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_autoscaler, copy the **google_compute_autoscaler** resource block example from the **Example Usage - Autoscaler Basic** section and paste into the bottom of **managed-instance-group.tf** For convenience the code is shown below..

```
resource "google_compute_autoscaler" "example_autoscaler" {
  name            = "example-autoscaler"
  target          = google_compute_instance_group_manager.lb-backend-group.instance_group
  zone = "us-east1-b"

  autoscaling_policy {
    min_replicas = 2
    max_replicas = 5

    load_balancing_utilization {
      target = 0.5
    }
  }
}
```

25. Change to resource block identifier from "**example_autoscaler**" to **"lab_autoscaler"**

26. Change the name from "**example-autoscaler**" to **"lab-autoscaler"**

27. Change the target reference from **google_compute_instance_group_manager.lb-backend-group.instance_group** to your instance group **google_compute_instance_group_manager.lab_mig.instance_group**

28. Change the zone to variable **var.lab-zone**

29. The updated code should be as shown below...

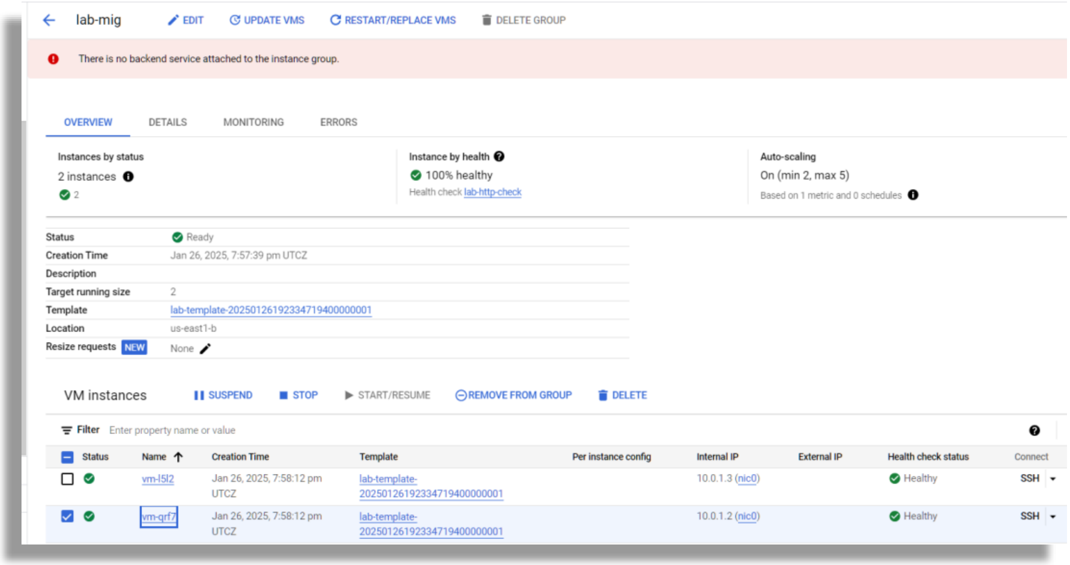
```
resource "google_compute_autoscaler" "lab_autoscaler" {
  name            = "lab-autoscaler"
  target          = google_compute_instance_group_manager.lab_mig.instance_group
  zone            = var.lab-zone

  autoscaling_policy {
    min_replicas = 2
    max_replicas = 5

    load_balancing_utilization {
      target = 0.5
    }
  }
}
```

30.Save changes to files and then apply them with **terraform apply**

31.Navigate to the Instance groups section of Compute Engine in the console. Select the **lab-mig** instance group to view the newly created instances. Wait for a minute or so and then refresh your browser to verify that the instances show as Healthy, indicating they have passed the health checks. Note the warning that there is no backend service attached to the instance group. This will be our next task.



There is no backend service attached to the instance group.

Instances by status
2 instances
✓ 2

Instance by health
✓ 100% healthy
Health check lab-http-check

Auto-scaling
On (min 2, max 5)
Based on 1 metric and 0 schedules

Status	✓ Ready
Creation Time	Jan 26, 2025, 7:57:39 pm UTCZ
Description	
Target running size	2
Template	lab-template-20250126192334719400000001
Location	us-east1-b
Resize requests NEW	None

☐	Status	Name ↑	Creation Time	Template	Per instance config	Internal IP	External IP	Health check status	Connect
☐	✓	vm-l5l2	Jan 26, 2025, 7:58:12 pm UTCZ	lab-template-20250126192334719400000001		10.0.1.3 (nic0)		✓ Healthy	SSH
☑	✓	vm-qrf7	Jan 26, 2025, 7:58:12 pm UTCZ	lab-template-20250126192334719400000001		10.0.1.2 (nic0)		✓ Healthy	SSH

Task 3. Create Load Balancer

1. Create a new file named **loadbalancer.tf** in the lab05 root directory
2. https://registry.terraform.io/providers/hashicorp/google/6.17.0/docs/resources/compute_global_forwarding_rule.html
3. Navigate down to **Example Usage - Global Forwarding Rule External Managed**

4. Copy the google_compute_global_forwarding_rule resource example into loadbalancer.tf

5. For convenience the code is shown below ...

   ```
   resource "google_compute_global_forwarding_rule" "default" {
    name                  = "global-rule"
    target                = google_compute_target_http_proxy.default.id
    port_range            = "80"
    load_balancing_scheme = "EXTERNAL_MANAGED"
   }
   ```

6. Change resource block identifier from "**default**" to "**http_forwarding**"

7. Change the target reference from **google_compute_target_http_proxy.default.id** to **google_compute_target_http_proxy.http_proxy.id** (we will create this next)

8. Add argument **ip_protocol** with value of **"TCP"**

9. Add argument **ip_address** with value of **module.vpc.lb-ip-id**

   Note: Here we are referencing the public ip address defined in the networking module and exposed in the network/outputs.tf file

10. Your modified code should be as shown below...

    ```
    resource "google_compute_global_forwarding_rule" "http_forwarding" {
     name                  = "global-rule"
     target                = google_compute_target_http_proxy.http_proxy.id
     port_range            = "80"
     load_balancing_scheme = "EXTERNAL_MANAGED"
     ip_protocol           = "TCP"
     ip_address            = module.vpc.lb-ip-id
    }
    ```

11. Copy the **google_compute_target_http_proxy** resource example into **loadbalancer.tf**

12. For convenience the code is shown below ...

    ```
    resource "google_compute_target_http_proxy" "default" {

     name        = "target-proxy"
     description = "a description"
     url_map     = google_compute_url_map.default.id
    }
    ```

13.Change the resource block identifier from "**default**" to "**http_proxy**"

14.Change the name from "**target-proxy**" to "**http-lb-proxy**"

15.Change the **url_map** reference from **google_compute_url_map.default.id** to **google_compute_url_map.http_url_map.id** (we will create this next)

16.Delete the optional description argument. Your code should now be as shown below...

```
resource "google_compute_target_http_proxy" "http_proxy" {
  name    = "http-lb-proxy"
  url_map = google_compute_url_map.http_url_map.id

}
```

15. Copy the google_compute_url_map resource example into loadbalancer.tf

16. For convenience the code is shown below ...

```
resource "google_compute_url_map" "default" {
  name            = "url-map-target-proxy"
  description     = "a description"
  default_service = google_compute_backend_service.default.id

  host_rule {
    hosts        = ["mysite.com"]
    path_matcher = "allpaths"
  }

  path_matcher {
    name            = "allpaths"
    default_service = google_compute_backend_service.default.id

    path_rule {
      paths   = ["/*"]
      service = google_compute_backend_service.default.id
    }
  }

}
```

17. Delete all arguments except name and default_service.

18. Change the resource block identifier from "**default**" to "**http_url_map**"

19. Change the default service from **google_compute_backend_service.default.id** to

google_compute_backend_service.backend_service.id (we will create this next)

20. Your modified code should be as shown below...

```
resource "google_compute_url_map" "http_url_map" {
  name            = "url-map-target-proxy"
  default_service = google_compute_backend_service.backend_service.id
}
```

21. Copy the google_compute_backend_service resource example into loadbalancer.tf

22. For convenience the code is shown below ...

```
resource "google_compute_backend_service" "default" {
  name                  = "backend"
  port_name             = "http"
  protocol              = "HTTP"
  timeout_sec           = 10
  load_balancing_scheme = "EXTERNAL_MANAGED"
}
```

23. Change the resource block identifier from "**default**" to "**backend_service**"

24. Add the following lines...

```
 health_checks = [google_compute_health_check.lab_health_check.id]
  backend {
    group = google_compute_instance_group_manager.lab_mig.instance_group
    balancing_mode  = "UTILIZATION"
    capacity_scaler = 1.0
  }
```

25. Your modified code should be as shown below...

```
resource "google_compute_backend_service" "backend_service" {
  name = "backend"
  port_name  = "http"
  protocol = "HTTP"
  timeout_sec = 10
  load_balancing_scheme = "EXTERNAL_MANAGED"
  health_checks = [google_compute_health_check.lab_health_check.id]
  backend {
    group = google_compute_instance_group_manager.lab_mig.instance_group
    balancing_mode  = "UTILIZATION"
    capacity_scaler = 1.0
```

```
     }
   }
```

26. Create a new file in the lab05 root directory named **outputs.tf**

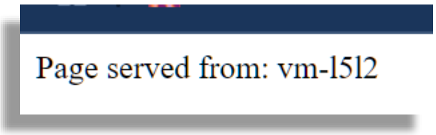
27. Add the following code to **lab05\outputs.tf** to output the load balance IP address to screen...

```
output "lb-ip" {
  description = "ip of lab load balancer"
  value =   module.vpc.lb-ip
}
```

28. Use **terraform apply** to deploy the 4 new resources.

Task 4. Test the Load Balancer

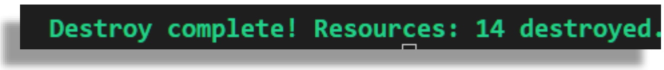
1. Note the IP address of the load balancer. Open a local browser tab and navigate to this address. You will receive error messages for a while whilst the load balancer is deployed globally. Refresh periodically until you are shown the name of the backend vm you are being connected to; this may take 3-4 minutes. Refresh your browser to verify load balancing across the 2 backend instances.



Page served from: vm-l512

Task 5. Lab Clean Up

1. Run terraform destroy to destroy your resources. This may take 5 minutes or more.



```
Destroy complete! Resources: 14 destroyed.
```

## Congratulations, you have completed this lab ###