Microsoft

Jose Moreno

PDU CSA-P

12/12/2016

Azure Networking Lab Guide for  
Open Source Professionals

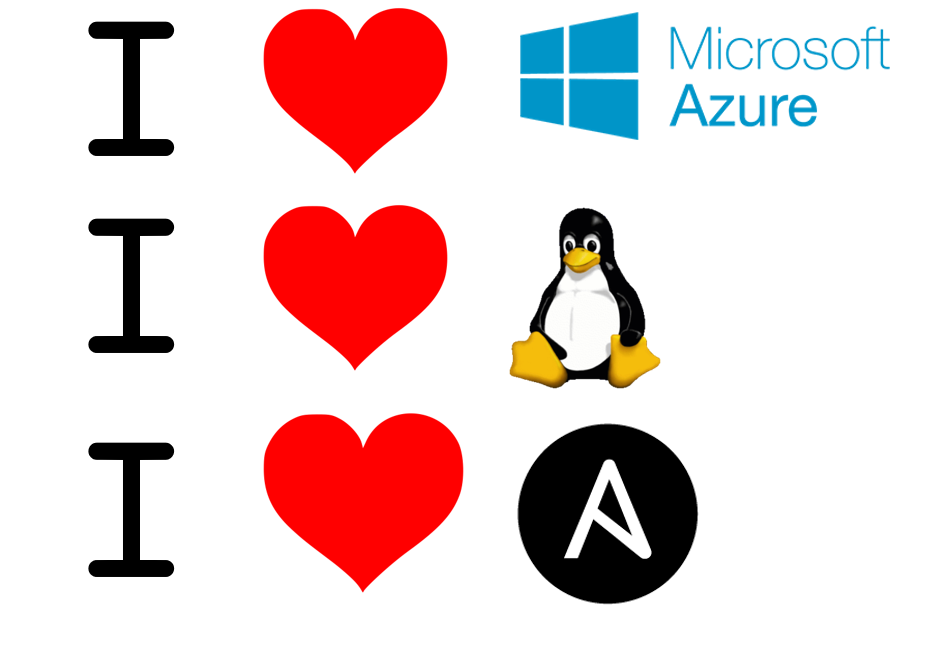


Table of Contents

[Objectives and initial setup 2](#_Toc478373126)

[Introduction to Azure Networking 3](#_Toc478373127)

[Lab 0: Initialize Environment 4](#_Toc478373128)

[Lab 1: Explore Lab environment 5](#_Toc478373129)

[Lab 2: Spoke-to-Spoke communication over vnet gateway 8](#_Toc478373130)

[Lab 3: spoke-to-spoke communication over NVA 13](#_Toc478373131)

[Lab 4: Microsegmentation with NVA 16](#_Toc478373132)

[Lab 5: VPN connection to the Hub Vnet 18](#_Toc478373133)

[Lab 6: NVA scalability 22](#_Toc478373134)

[Lab 7: Outgoing Internet Traffic Protected by NVA 27](#_Toc478373135)

[Lab 8: Incoming Internet Traffic Protected by NVA 29](#_Toc478373136)

[Lab 9: Advanced HTTP-based probes 33](#_Toc478373137)

[Conclusion 34](#_Toc478373138)

[End the lab 35](#_Toc478373139)

[References 35](#_Toc478373140)

# Objectives and initial setup

This document contains a lab guide that helps to deploy a basic environment in Azure that allows to test some of the functionality of the integration between Azure and Ansible.

Before starting with this account, make sure to fulfill all the requisites:

* A valid Azure subscription account. If you don’t have one, you can create your [free azure account](https://azure.microsoft.com/en-us/free/) (https://azure.microsoft.com/en-us/free/) today.
* If you are using Windows 10, you can [install Bash shell on Ubuntu on Windows](http://www.windowscentral.com/how-install-bash-shell-command-line-windows-10) (<http://www.windowscentral.com/how-install-bash-shell-command-line-windows-10>).
* Azure CLI 2.0, follow these instructions to install: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli>

The labs cover:

* Introduction to Azure networking
* Deployment of multi-vnet Hub and Spoke design
* Using NVAs (Network Virtual Appliances) for traffic filtering and microsegmentation
* Scaling out NVAs with load balancing and SNAT

Along this lab some variables will be used, that might (and probably should) look different in your environment. This is the variables you need to decide on before starting with the lab. Notice that the VM names are prefixed by a (not so) random number, since these names will be used to create DNS entries as well, and DNS names need to be unique.

|  |  |
| --- | --- |
| **Description** | **Value used in this lab guide** |
| Azure resource group | vnetTest |
| Username for provisioned VMs and NVAs | lab-user |
| Password for provisioned VMs and NVAs | Microsoft123! |
| Shared key for VPN | Microsoft123 |

# Introduction to Azure Networking

You can find Azure Networking documentation here: <https://docs.microsoft.com/en-us/azure/#pivot=services&panel=network>.

Write something!!

# Lab 0: Initialize Environment

1. Create a new resource group, where we will place all our objects (so that you can easily delete everything after you are done). The last command also sets the default resource group to the newly created one, so that you do not need to download it.

**az login**

The “az login” command will provide you a code, that you need to introduce (over copy and paste) in the web page <http://aka.ms/devicelogin>. After introducing the code, you will need to authenticate with credentials that are associated to a valid Azure subscription.

**az group create --name vnetTest --location westeurope**

**az configure --defaults group=vnetTest**

1. Deploy the master template that will deploy our initial network configuration:

**az group deployment create --name netLabDeployment --template-uri https://raw.githubusercontent.com/erjosito/azure-networking-lab/master/NetworkingLab\_master.json --resource-group vnetTest --parameters '{"adminUsername":{"value":"lab-user"}, "adminPassword":{"value":"Microsoft123!"}}'**

1. As preparation for one of the labs later in this guide, upgrade the deployed Vnet Gateways vnet4gw and vnet5gw from basic to standard. The following commands will kick off a conversion job that will run in the background for some minutes, but after running them, you can safely continue with the next lab in this guide:

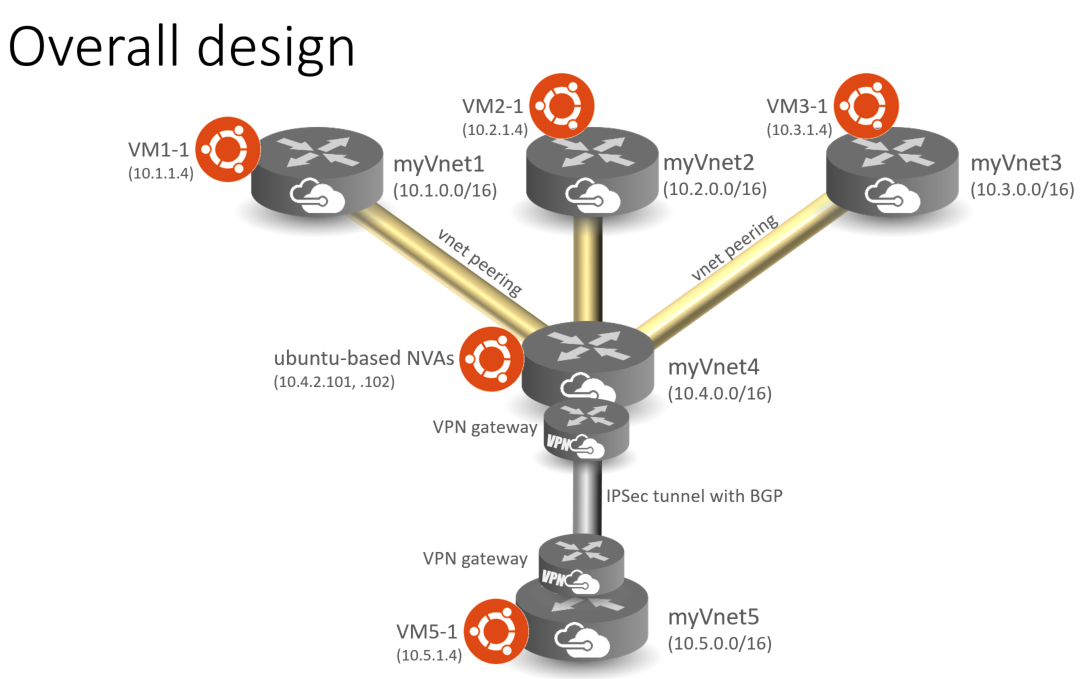
**az network vnet-gateway update --sku Standard -n vnet4gw**

**az network vnet-gateway update --sku Standard -n vnet5gw**

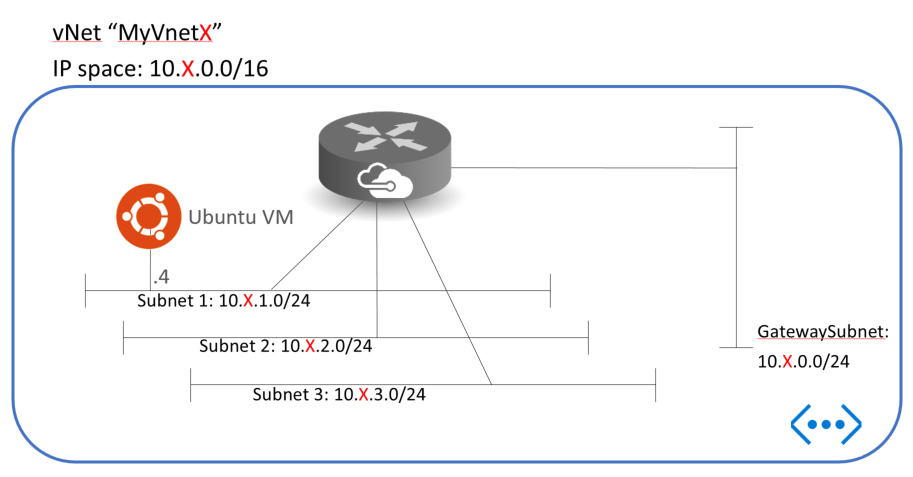
# Lab 1: Explore Lab environment

1. Explore the objects created by the ARM template: vnets, subnets, VMs, interfaces, public IP addresses, etc. Save the output of these commands.

You can see some diagrams about the deployed environment here, so that you can interpret better the command outputs



**Figure 1**. Overall vnet diagram



**Figure 2**. Subnet design of every vnet

**$ az network vnet list -o table**

Location Name ProvisioningState ResourceGroup

---------- ------------- ------------------- ---------------

westeurope myVnet1 Succeeded vnetTest

westeurope myVnet2 Succeeded vnetTest

westeurope myVnet3 Succeeded vnetTest

westeurope myVnet4 Succeeded vnetTest

westeurope myVnet5 Succeeded vnetTest

*Note: Some columns of the ouput above have been removed for clarity purposes.*

**$ az network vnet subnet list --vnet-name myVnet1 -o table**

AddressPrefix Name ProvisioningState ResourceGroup

--------------- -------------- ------------------- ---------------

10.1.0.0/24 GatewaySubnet Succeeded vnetTest

10.1.1.0/24 myVnet1Subnet1 Succeeded vnetTest

10.1.2.0/24 myVnet1Subnet2 Succeeded vnetTest

10.1.3.0/24 myVnet1Subnet3 Succeeded vnetTest

**$ az vm list -o table**

Name ResourceGroup Location

-------------- --------------- ----------

myVnet1vm VNETTEST westeurope

myVnet2vm VNETTEST westeurope

myVnet3vm VNETTEST westeurope

myVnet4vm VNETTEST westeurope

myVnet5vm VNETTEST westeurope

nva-1 VNETTEST westeurope

nva-2 VNETTEST westeurope

**$ az network nic list -o table**

Location Name Primary MacAddress IpForwarding

---------- ----------------- --------- ----------------- -------------

westeurope myVnet1vmnic True 00-0D-3A-21-24-6B

westeurope myVnet2vmnic True 00-0D-3A-24-E2-3A

westeurope myVnet3vmnic True 00-0D-3A-21-3B-CB

westeurope myVnet4vmnic True 00-0D-3A-23-C8-96

westeurope myVnet5vmnic True 00-0D-3A-23-C9-68

westeurope nva-1-nic0 True 00-0D-3A-28-86-EA True

westeurope nva-1-nic1 00-0D-3A-28-80-E5 True

westeurope nva-2-nic0 True 00-0D-3A-28-85-0E True

westeurope nva-2-nic1 00-0D-3A-28-8C-78 True

*Note: Some columns of the ouput above have been removed for clarity purposes.*

**$ az network public-ip list -o table**

Name PublicIpAllocationMethod ResourceGroup IpAddress

----------------- ------------------- ---------------------- -----------

myVnet1vmpip Dynamic vnetTest 52.174.33.80

myVnet2vmpip Dynamic vnetTest 40.68.103.227

myVnet3vmpip Dynamic vnetTest 52.232.76.15

myVnet4vmpip Dynamic vnetTest 52.166.196.212

myVnet5vmpip Dynamic vnetTest 52.166.193.255

nvaPip-1 Dynamic vnetTest 13.81.116.28

nvaPip-2 Dynamic vnetTest 13.81.115.31

vnet4gwPip Dynamic vnetTest 13.81.113.28

vnet5gwPip Dynamic vnetTest 13.81.112.142

*Note: Some columns of the ouput above have been removed for clarity purposes.*

**$ az network vnet-gateway list -o table**

EnableBgp GatewayType Location Name VpnType

----------- ------------- ---------- ------- ----------

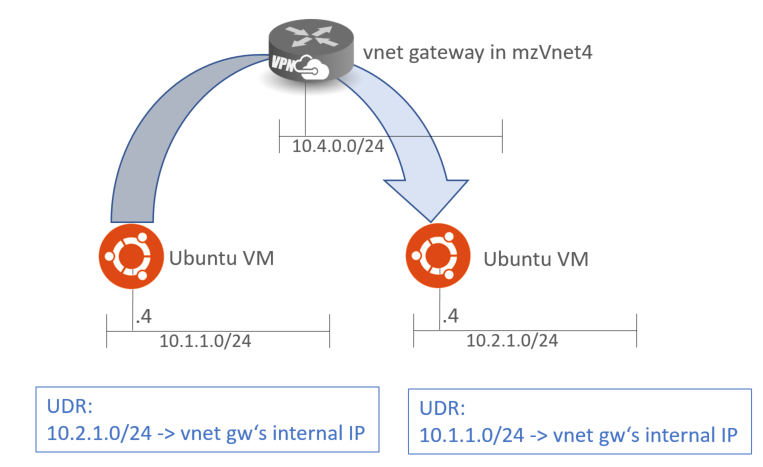
True Vpn westeurope vnet4gw RouteBased

True Vpn westeurope vnet5gw RouteBased

*Note: Some columns of the ouput above have been removed for clarity purposes.*

# Lab 2: Spoke-to-Spoke communication over vnet gateway

Spokes can speak to other spokes by redirecting traffic to a vnet gateway or an NVA in the hub vnet by means of UDRs. The following diagram illustrates what we are trying to achieve in this lab:



**Figure** . Spoke-to-spoke communication over vnet gateway

1. After verifying the public IP address assigned to the first VM in vnet1 (called "myVnet1vm"), connect to it using the credentials that you specified when deploying the template, and verify that you don’t have connectivity to the VM in vnet2:

**$ ssh 52.174.33.80**

The authenticity of host '52.174.33.80 (52.174.33.80)' can't be established.

ECDSA key fingerprint is b5:24:f3:aa:1e:f2:1d:fa:09:0e:b4:91:fa:49:b5:2f.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '52.174.33.80' (ECDSA) to the list of known hosts.

lab-user@52.174.33.80's password:

lab-user@myVnet1vm:~$ ping 10.2.1.4

PING 10.2.1.4 (10.2.1.4) 56(84) bytes of data.

^C

--- 10.2.1.4 ping statistics ---

10 packets transmitted, 0 received, **100% packet loss**, time 8999ms

*Note: please note your IP address will be different to the one used in this example.*

1. Verify that the involved subnets (myVnet1-Subnet1 and myVnet2-Subnet1) do not have any routing table attached:

**$ az network vnet subnet show --vnet-name myVnet1 -n myVnet1Subnet1 | grep routeTable**

"routeTable": null

1. Create a custom route table named "vnet1-subnet1", and another one called "vnet2-subnet1":

**$ az network route-table create --name vnet1-subnet1**

**$ az network route-table create --name vnet1-subnet1**

1. Now attach the custom route tables to both subnets involved in this example (Vnet1Subnet1, Vnet2Subnet2):

**$ az network vnet subnet update -n myVnet1Subnet1 --vnet-name myVnet1 --route-table vnet1-subnet1**

**$ az network vnet subnet update -n myVnet2Subnet1 --vnet-name myVnet2 --route-table vnet2-subnet1**

1. And now you can check that the subnets are associated with the right routing tables:

**$ az network vnet subnet show --vnet-name myVnet1 -n myVnet1Subnet1 | grep routeTable**

"routeTable": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1",

**$ az network vnet subnet show --vnet-name myVnet2 -n myVnet2Subnet1 | grep routeTable**

"routeTable": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet2-subnet1",

1. Now we can try to tell Azure to send traffic from subnet 1 to subnet 2 over the hub vnet. Normally you would do this by sending traffic to the vnet router. Let’s see what happens if we try this with vnet1. In order to do so, we need to add a new route to our custom routing table:

**$ az network route-table route create --address-prefix 10.2.0.0/16 --next-hop-type vnetLocal --route-table-name vnet1-subnet1 -n vnet2**

1. You can verify that the route has been added to the routing table correctly:

**$ az network route-table route list --route-table-name vnet1-subnet1 -o table**

AddressPrefix Name NextHopIpAddress NextHopType Provisioning

--------------- -------------------- ------------------ ---------------- ---------

10.2.0.0/16 vnet2 VnetLocal Succeeded

However, if we verify the routing table that has been programmed in the interface of VMs in the subnet, you can see that the next hop is actually “None”! (in other words, drop the packets):

**$ az network nic show-effective-route-table -n myVnet1vmnic**

...

{

"addressPrefix": [

"**10.2.0.0/16**"

],

"name": "**vnet2**",

"nextHopIpAddress": [],

"nextHopType": "**None**",

"source": "User",

"state": "Active"

},

1. Now we will install in each route table routes for the other side, pointing to the private IP address of the vnet gateway in vnet 4. This private address is usually the fourth one in the subnet. In our case, 10.4.0.4. You can confirm that this address exists by trying to ping it from any VM.

**$ az network route-table route update --address-prefix 10.2.0.0/16 --next-hop-ip-address 10.4.0.4 --next-hop-type VirtualAppliance --route-table-name vnet1-subnet1 -n vnet2**

**$ az network route-table route create --address-prefix 10.1.0.0/16 --next-hop-ip-address 10.4.0.4 --next-hop-type VirtualAppliance --route-table-name vnet2-subnet1 -n vnet1**

1. We can verify what the route tables look like now, and how it has been programmed in one of the NICs associated to the subnet:

**$ az network route-table route list --route-table-name vnet1-subnet1 -o table**

AddressPrefix Name NextHopIpAddress NextHopType ProvisioningState

--------------- ------- ------------------ ---------------- -------------------

10.2.0.0/16 vnet2 10.4.0.4 VirtualAppliance Succeeded

**$ az network nic show-effective-route-table -n myVnet1vmnic**

...

{

"addressPrefix": [

"10.2.0.0/16"

],

"name": "vnet2",

"nextHopIpAddress": [

"10.4.0.4"

],

"nextHopType": "VirtualAppliance",

"source": "User",

"state": "Active"

}

1. And now VM1 should be able to reach VM2:

**lab-user@myVnet1vm:~$ ping 10.2.1.4**

PING 10.2.1.4 (10.2.1.4) 56(84) bytes of data.

64 bytes from 10.2.1.4: icmp\_seq=4 ttl=63 time=7.59 ms

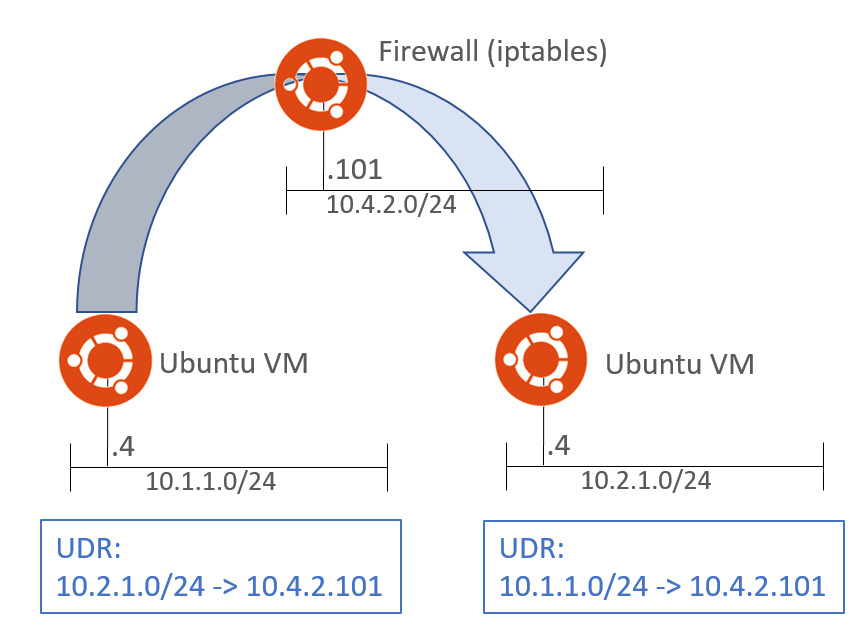
64 bytes from 10.2.1.4: icmp\_seq=5 ttl=63 time=5.79 ms

64 bytes from 10.2.1.4: icmp\_seq=6 ttl=63 time=4.90 ms

# Lab 3: spoke-to-spoke communication over NVA

In some situations you would want some kind of security between the different Vnets. Although this security can be partially provided by Network Security Groups, certain organizations might require some more advanced filtering functionality such as the one that firewalls provide.

In this lab we will insert a Network Virtual Appliance in the communication flow. Typically these Network Virtual Appliance might be a next-generation firewall of vendors such as Barracuda, Checkpoint, Cisco or Palo Alto, to name a few, but in this lab we will use a Linux machine with 2 interfaces and traffic forwarding enabled. For this exercise, the firewall will be inserted as a “firewall on a stick”, that is one single interface will suffice.



**Figure** . Spoke-to-spoke traffic going through an NVA

1. In the Ubuntu VM acting as firewall iptables have been configured by means of a Custom Script Extension. This extension downloads a script from a public repository (the Github repository for this lab) and runs it on the VM on provisioning time. Verify that the NVAs have successfully registered the extensions with this command:

**$ az vm extension list --vm-name nva-1 -o table**

1. We need to replace the route we installed in Vnet1-Subnet1 and Vnet2-Subnet1 pointing to Vnet4’s vnet gateway, with another one pointing to the NVA. We will use the first NVA, with an IP address of 10.4.2.101.

**$ az network route-table route update --next-hop-ip-address 10.4.2.101 --route-table-name vnet1-subnet1 -n vnet2**

**$ az network route-table route update --next-hop-ip-address 10.4.2.101 --route-table-name vnet2-subnet1 -n vnet1**

1. Find out the public IP address of nva-1, SSH to it and have a look at the iptables rules:

**$ ssh lab-user@13.81.116.28**

The authenticity of host '13.81.116.28 (13.81.116.28)' can't be established.

ECDSA key fingerprint is 17:ac:de:80:b4:48:fc:22:78:18:59:ec:f9:b6:27:ad.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '13.81.116.28' (ECDSA) to the list of known hosts.

lab-user@13.81.116.28's password:

Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86\_64)

lab-user@nva-1:~$ **sudo iptables -L**

Chain INPUT (policy ACCEPT)

target prot opt source destination

ACCEPT udp -- anywhere anywhere udp dpt:bootpc

Chain **FORWARD** (policy ACCEPT)

target prot opt source destination

**DROP icmp** -- anywhere anywhere

ACCEPT all -- anywhere anywhere

ACCEPT all -- anywhere anywhere

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

1. You can verify that VM1 cannot ping VM2, but it can SSH into it:

lab-user@myVnet1vm:~$ **ping 10.2.1.4**

PING 10.2.1.4 (10.2.1.4) 56(84) bytes of data.

^C

--- 10.2.1.4 ping statistics ---

9 packets transmitted, 0 received, **100% packet loss**, time 8033ms

lab-user@myVnet1vm:~$

lab-user@myVnet1vm:~$ ssh 10.2.1.4

The authenticity of host '10.2.1.4 (10.2.1.4)' can't be established.

ECDSA key fingerprint is SHA256:o+kldZQA9cY9bOXQOUUMd3keFXN2TofSGXcJ1VxKuXM.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '10.2.1.4' (ECDSA) to the list of known hosts.

lab-user@10.2.1.4's password:

Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86\_64)

lab-user@myVnet2vm:~$

1. Now you can remove the rule that drops ICMP traffic in the firewall, and verify that ping is now working too. With this you have successfully verified that traffic is actually controlled by the firewall.

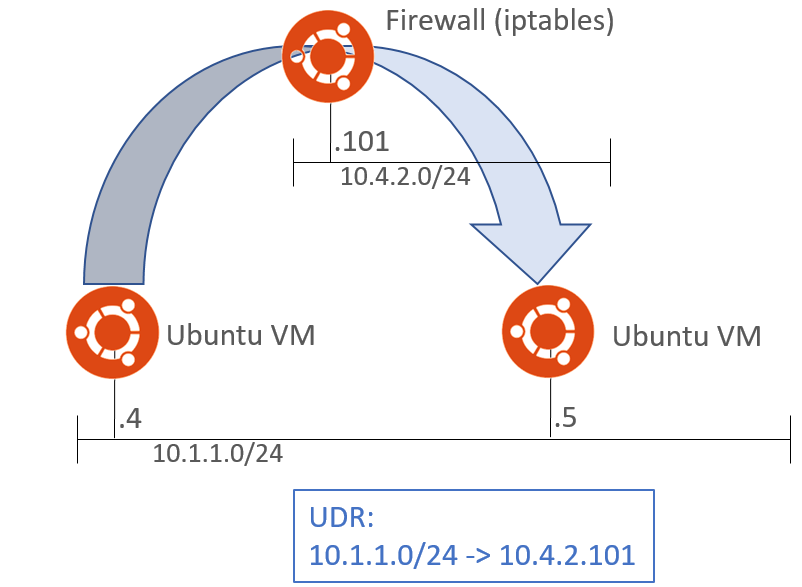
lab-user@nva-1:~$ **sudo iptables -D FORWARD -p icmp -j DROP**

1. Now put back the rule to drop the ICMP traffic, we will need it in further labs:

lab-user@nva-1:~$ **sudo iptables -A FORWARD -p icmp -j DROP**

# Lab 4: Microsegmentation with NVA

Some organizations wish to filter not only traffic between specific network segments, but traffic inside of a subnet as well, in order to reduce the probability of successful attacks spreading inside of an organization. This is what some in the industry know as “microsegmentation”.



**Figure X**. Intra-subnet NVA-based filtering, also known as “microsegmentation”

1. In order to be able to test the topology above, we need to deploy a second VM in myVnet1-Subnet1. You can do so with the following command (find the name of the account name with "az storage account list -o table"):

**az group deployment create --name netLab --template-uri https://raw.githubusercontent.com/erjosito/azure-network-lab/master/linuxVM.json --resource-group vnetTest --parameters '{"adminUsername":{"value":"lab-user"}, "adminPassword":{"value":"Microsoft123!"}, "vmName":{"value":"vnet1vm2"}, "vmType":{"value":"ubuntu"}, "vnetName":{"value":"myVnet1"}, "subnetName":{"value":"myVnet1Subnet1"}, "storageAccountName":{"value":"storagevs5yh554ku772"}}'**

1. Now we need to instruct all VMs in subnet 1 to send local traffic to the NVAs as well. This can be easily done by adding an additional User-Defined Route to the corresponding routing table:

**$ az network route-table route create --address-prefix 10.1.1.0/24 --next-hop-ip-address 10.4.2.101 --next-hop-type VirtualAppliance --route-table-name vnet1-subnet1 -n vnet1-subnet1**

1. You can verify similarly to the previous lab that traffic now is flowing through the firewall, by enabling and disabling the ICMP rule as described in the previous section.

# Lab 5: VPN connection to the Hub Vnet

This lab probably makes sense later, to wait for the completion of the vnet-gateway resize command

1. First, we need to change the BGP Autonomous System Number (ASN) of one of the gateways, since the ARM template deploys them with the same one, but in order to set up a Vnet-to-Vnet connection they need to be different. You can check the ASN with this command:

**$ az network vnet-gateway show -n vnet4gw | grep asn**

"asn": **65515**,

And you can change it this way (please note that the resize operation initiated in the previous lab might not have finished yet, in which case the following command cannot be executed):

**az network vnet-gateway update --asn 65514 -n vnet4gw**

1. Once both Vnet gateways have different ASNs, we can establish a VPN tunnel between them. Note that tunnels are bidirectional, so you will need to establish a tunnel from vnet4gw to vnet5gw, and another one in the opposite direction (note that it is normal for these commands to take some time to run):

**$ az network vpn-connection create -n 4to5 --vnet-gateway1 vnet4gw --enable-bgp --shared-key Microsoft123 --vnet-gateway2 vnet5gw**

**$ az network vpn-connection create -n 5to4 --vnet-gateway1 vnet5gw --enable-bgp --shared-key Microsoft123 --vnet-gateway2 vnet4gw**

Once you have provisioned the connections you can check their state with this command:

**$ az network vpn-connection list -o table**

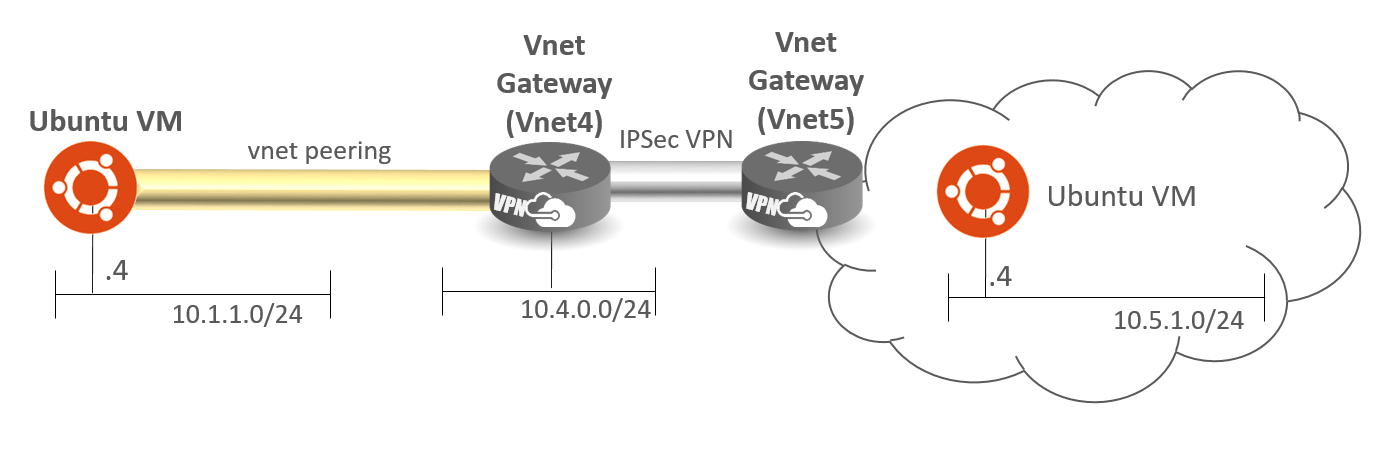
1. If you now try to reach a VM in myVnet5 from any of the VMs in the other Vnets, it should work without any further configuration, following the topology found in the figure below:

**lab-user@myVnet1vm:~$ ping 10.5.1.4**

PING 10.5.1.4 (10.5.1.4) 56(84) bytes of data.

64 bytes from 10.5.1.4: icmp\_seq=1 ttl=62 time=10.9 ms

64 bytes from 10.5.1.4: icmp\_seq=2 ttl=62 time=9.92 ms



**Figure X**: VPN connection through Vnet peering

This is so because of how the Vnet peerings were configured, more specifically the parameters AllowForwardedTraffic and UseRemoteGateways (in the spokes), and AllowGatewayTransite (in the hub):

**az network vnet peering list --vnet-name myVnet1 -o table**

AllowForwardedTraffic Name PeeringState UseRemoteGateways

----------------------- ------------- -------------- -------------------

**True** LinkTomyVnet4 Connected **True**

**$ az network vnet peering list --vnet-name myVnet4 -o table**

AllowGatewayTransit Name PeeringState

--------------------- ------------- --------------

**True** LinkTomyVnet2 Connected

**True**  LinkTomyVnet1 Connected

**True**  LinkTomyVnet3 Connected

1. You can have a look at the effective routing table of an interface, and you will see that a route for Vnet5 has been automatically established, pointing to the vnet Gateway of the hub Vnet (to its public IP address, to be accurate):

**$ az network nic show-effective-route-table -n myVnet1vmnic**

...

{

"addressPrefix": [

"**10.5.0.0/16**"

],

"name": null,

"nextHopIpAddress": [

"**13.81.113.28**"

],

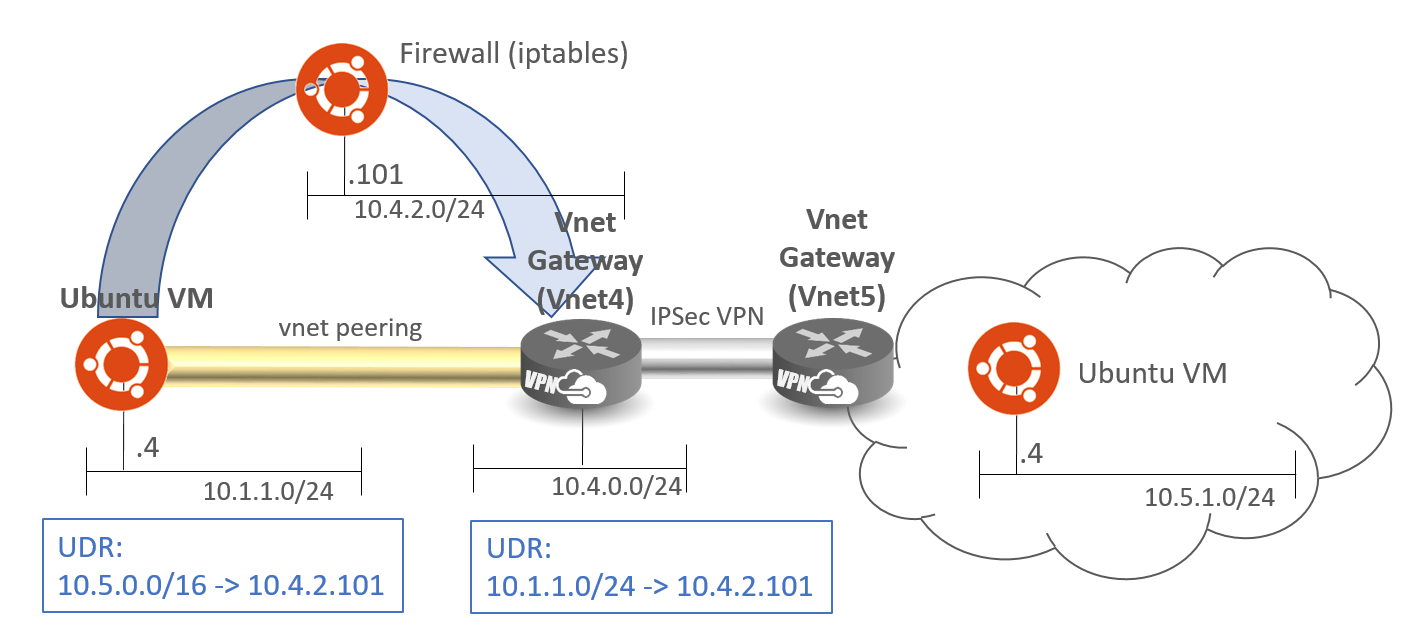
"nextHopType": "**VirtualNetworkGateway**",

"source": "**VirtualNetworkGateway**",

"state": "Active"

},

However, you might want to push this traffic through the Network Virtual Appliances too. The process that we have seen so far is valid for the GatewaySubnet of Vnet4 (where the hub VPN gateway is located), as the following figure depicts:



**Figure X**. VPN traffic combined with Vnet peering and a Network Virtual Appliance

1. For the gateway subnet in myVnet4 we will create a new routing table, add a route for vnet1-subnet1, and associate the route table to the GatewaySubnet:

$ **az network route-table create --name vnet4-gw**

{

"etag": "W/\"c784f479-3e85-42d0-ba7b-d2c420f4d3d3\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet4-gw",

"location": "westeurope",

"name": "vnet4-gw",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"routes": [],

"subnets": null,

"tags": null,

"type": "Microsoft.Network/routeTables"

}

$ **az network route-table route create --address-prefix 10.1.1.0/24 --next-hop-ip-address 10.4.2.101 --next-hop-type VirtualAppliance --route-table-name vnet4-gw -n vnet1-subnet1**

{

"addressPrefix": "10.1.1.0/24",

"etag": "W/\"97e76ca7-9217-4137-80fe-6c40a8488e09\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet4-gw/routes/vnet1-subnet1",

"name": "vnet1-subnet1",

"nextHopIpAddress": "10.4.2.101",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

$ **az network route-table route list --route-table-name vnet4-gw -o table**

AddressPrefix Name NextHopIpAddress NextHopType

--------------- ------------- ------------------ ----------------

10.1.1.0/24 vnet1-subnet1 10.4.2.101 VirtualAppliance

$ **az network vnet subnet update -n GatewaySubnet --vnet-name myVnet4 --route-table vnet4-gw**

**$ az network route-table route create --address-prefix 10.5.0.0/16 --next-hop-ip-address 10.4.2.101 --next-hop-type VirtualAppliance --route-table-name vnet1-subnet1 -n vnet5**

1. Now you can verify that VMs in myVnet1Subnet1 can still connect over SSH to VMs in myVnet5, but not any more over ICMP (as long as the rule for dropping ICMP traffic is configured in the NVA):

lab-user@myVnet1vm:~$ **ping 10.5.1.4**

PING 10.5.1.4 (10.5.1.4) 56(84) bytes of data.

^C

--- 10.5.1.4 ping statistics ---

3 packets transmitted, 0 received, **100% packet loss**, time 1999ms

lab-user@myVnet1vm:~$ **ssh 10.5.1.4**

The authenticity of host '10.5.1.4 (10.5.1.4)' can't be established.

ECDSA key fingerprint is SHA256:x8VGe15aAkaIRjznPaUzO94IkXHQlmh4h2g1Jq1oOdk.

Are you sure you want to continue connecting (yes/no)? zes

Please type 'yes' or 'no': yes

Warning: Permanently added '10.5.1.4' (ECDSA) to the list of known hosts.

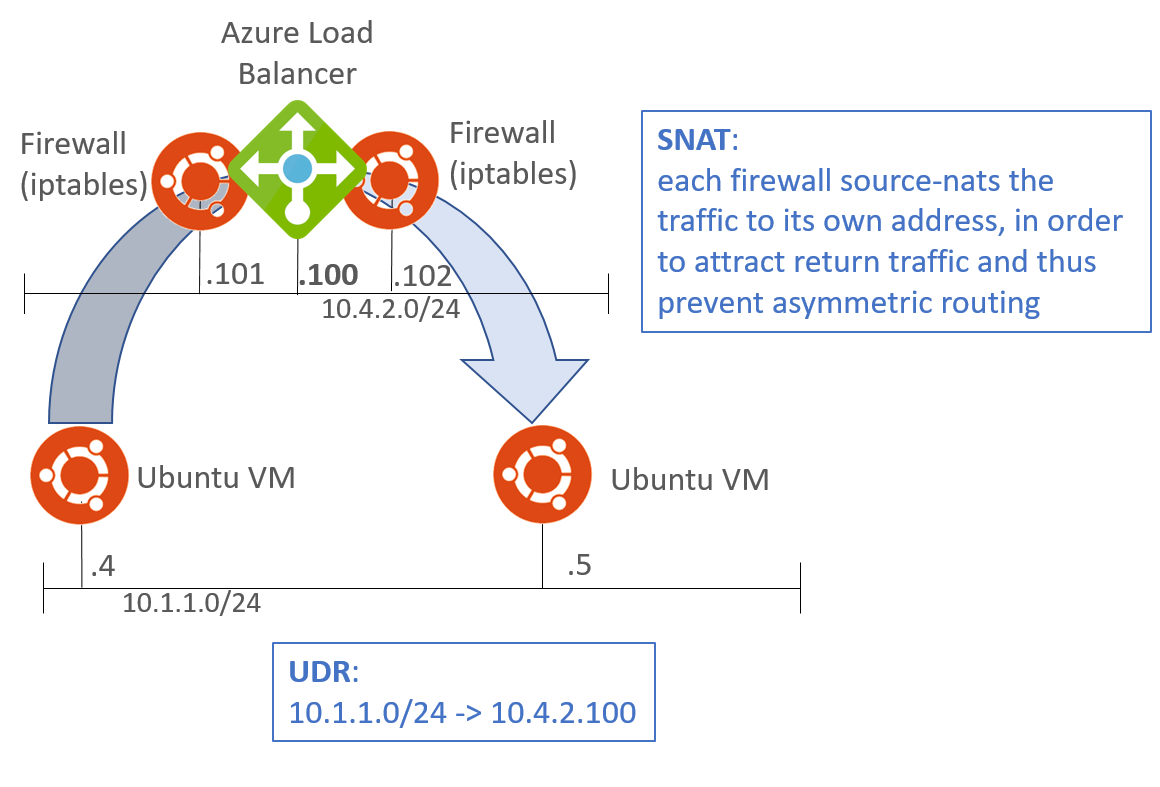
lab-user@10.5.1.4's password:

**Welcome** to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86\_64)

# Lab 6: NVA scalability

If all traffic is going through a single Network Virtual Appliance, chances are that it is not going to scale. Whereas you could scale it up by resizing the VM where it lives, not all VM sizes are supported by NVA vendors. Besides, scale out provides a more linear way of achieving additional performance, potentially even increasing and decreasing the number of NVAs automatically via scale sets.

In this lab we will use two NVAs and will send the traffic over both of them by means of an Azure Load Balancer. Since return traffic must flow through the same NVA (since firewalling is a stateful operation and asymmetric routing would break it), the firewalls will source-NAT traffic to their individual addresses.



**Figure X**. Load balancer for NVA scale out

Note that no clustering function is required in the firewalls, each firewall is completely unaware of the others.

1. First, verify that an internal load balancer has been deployed with the frontend IP address 10.4.2.100:

**$ az network lb list -o table**

Location Name ProvisioningState ResourceGroup

---------- ------- ------------------- ---------------

westeurope nva-slb-int Succeeded vnetTest

1. Now get information about the object names inside of the Load Balancer. The following command can be used in order to get the most relevant attributes:

**$ az network lb show -n nva-slb-int | grep name**

"name": "nva-slbBackend-int",

"name": "myFrontendConfig",

"name": null,

"name": "mySLBConfig",

"name": "nva-slb-int",

"name": "myProbe",

1. Now we need to add the internal interfaces of both appliances to the backend address pool of the load balancer:

**$ az network nic ip-config address-pool add --ip-config-name nva-1-nic0-ipConfig --nic-name nva-1-nic0 --address-pool nva-slbBackend-int --lb-name nva-slb-int**

**$ az network nic ip-config address-pool add --ip-config-name nva-2-nic0-ipConfig --nic-name nva-2-nic0 --address-pool nva-slbBackend-int --lb-name nva-slb-int**

1. Let us verify the LB's rules. In this case, we need to remove the existing one and replace it with another, where we will enable Direct Server Return:

**$ az network lb rule list --lb-name nva-slb-int -o table**

BackendPort FrontendPort LoadDistribution Name Protocol

------------- -------------- ------------------ ----------- --------

22 1022 Default mySLBConfig Tcp

**$ az network lb rule delete --lb-name nva-slb-int -n mySLBConfig**

**$ az network lb rule create --backend-pool-name nva-slbBackend-int --protocol Tcp --backend-port 22 --frontend-port 22 --frontend-ip-name myFrontendConfig --lb-name nva-slb-int --name sshRule --floating-ip true --probe-name myProbe**

1. We must change the next-hop for the UDRs that are required for the communication. We need to point them at the virtual IP address of the load balancer (10.4.2.100). We will take the route for microsegmentation, in order to test the connection depicted in the picture above:

**$ az network route-table route update --route-table-name vnet1-subnet1 -n vnet1-subnet1 --next-hop-ip-address 10.4.2.100**

At this point communication between the VMs should be possible, flowing through the NVA (ICMP should be dropped as long as the rule for dropping ICMP traffic is configured at the firewall).

lab-user@myVnet1vm:~$ **ping 10.1.1.5**

PING 10.1.1.5 (10.1.1.5) 56(84) bytes of data.

^C

--- 10.1.1.5 ping statistics ---

2 packets transmitted, 0 received, **100% packet loss**, time 1006ms

lab-user@myVnet1vm:~$ **ssh 10.1.1.5**

lab-user@10.1.1.5's password:

**Welcome** to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86\_64)

1. Observe the source IP address that the destination machine sees:

**lab-user@myvm:~$ who**

lab-user pts/0 2017-03-23 23:41 (**10.4.2.101**)

This is expected, since firewalls are configured to source NAT the connections outgoing on that interface:

lab-user@nva-1:~$ **sudo iptables -L -t nat**

Chain PREROUTING (policy ACCEPT)

target prot opt source destination

Chain INPUT (policy ACCEPT)

target prot opt source destination

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

Chain POSTROUTING (policy ACCEPT)

target prot opt source destination

**MASQUERADE all -- anywhere anywhere**

1. We can simulate a failure of the NVA where the connection is going through (in this case 10.4.2.101, nva-1). We can do it connecting to the other interface (10.4.3.101), and shutting down eth0: change this to kill -9 the nc process

**jose@nva-1:~$ ip a**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether 00:0d:3a:28:86:ea brd ff:ff:ff:ff:ff:ff

inet **10.4.2.101**/24 brd 10.4.2.255 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::20d:3aff:fe28:86ea/64 scope link

valid\_lft forever preferred\_lft forever

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether 00:0d:3a:28:80:e5 brd ff:ff:ff:ff:ff:ff

inet 10.4.3.101/24 brd 10.4.3.255 scope global eth1

valid\_lft forever preferred\_lft forever

inet6 fe80::20d:3aff:fe28:80e5/64 scope link

valid\_lft forever preferred\_lft forever

**jose@nva-1:~$ sudo ifconfig eth0 down**

The SSH session will become irresponsive, since the flow is broken. However, if you initiate another SSH connection to VM2 (10.1.1.5) from VM1 (10.1.1.4), you will see that you are going now through nva-2:

**lab-user@myvm:~$ who**

lab-user pts/0 2017-03-23 23:41 (10.4.2.101)

lab-user pts/1 2017-03-24 00:01 (**10.4.2.102**)

1. Restart the process listening to the load balancer probes on port 1138:

**$**

# Lab 7: Outgoing Internet Traffic Protected by NVA

What if we want to send all traffic leaving the vnet towards the public Internet through the NVAs? Wwe need to do is make sure that Internet traffic to/from all VMs flows through the NVAs via User-Defined Routes, and that NVAs source-NAT the outgoing traffic with their public IP address, so that they get the return traffic too.

For this test we will use the VM in vnet3.

1. Create a routing table for myVnet3Subnet1:

**$ az network route-table create --name vnet3-subnet1**

Create a default route in that table pointing to the internal LB VIP (10.4.2.100):

Associate the route table to the subnet myVnet3Subnet1:

Verify that the NVAs are

**$ sudo iptables -vL -t nat**

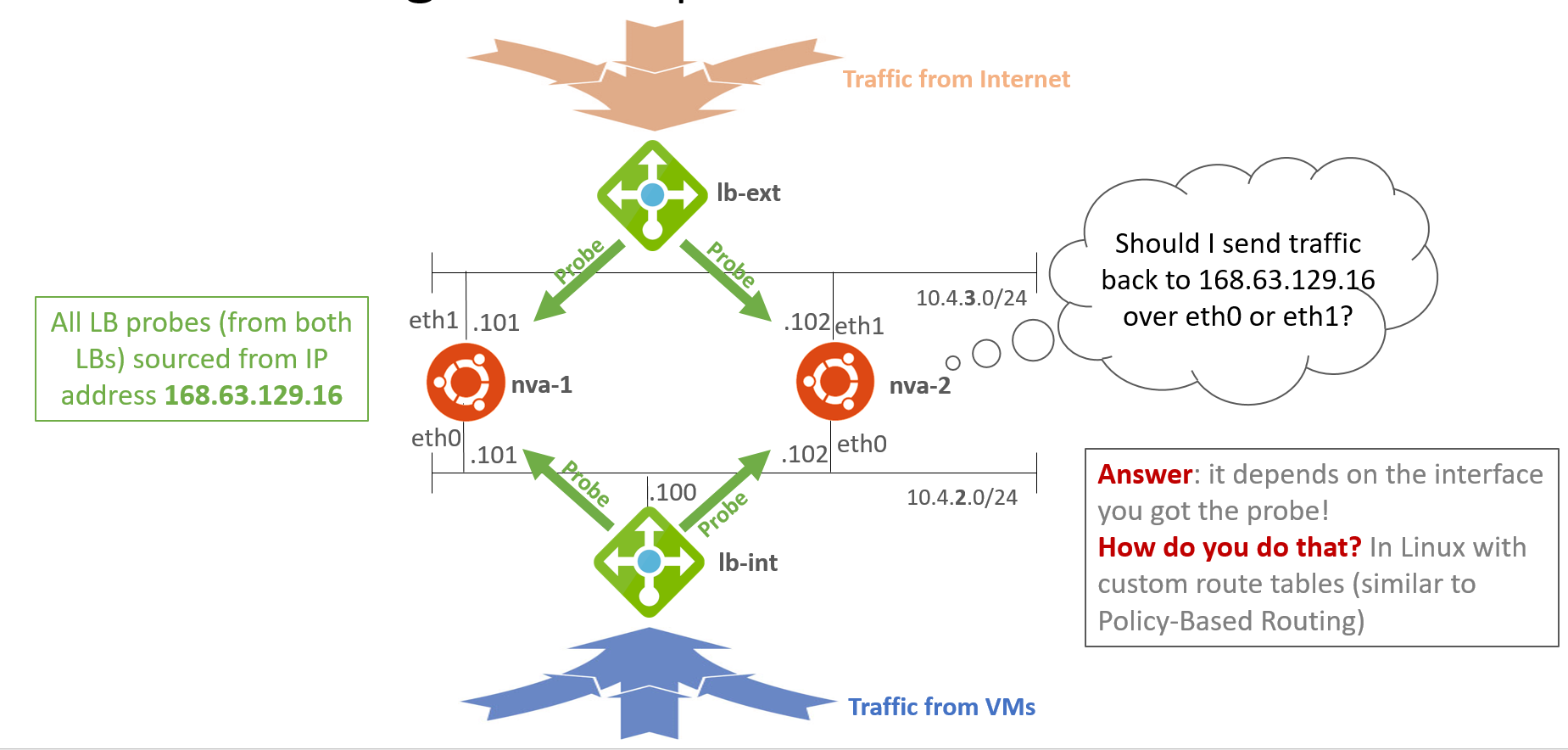
Now verify that you have internet access. Note that you don’t have Internet access to the VM in myVnet3Subnet1 any more, after changing default routing for that subnet. You can connect to one of the NVAs, and from there SSH to the internal IP address of the VM (10.3.1.4):

**$**

# Lab 8: Incoming Internet Traffic Protected by NVA

In this lab we will explore what needs to be done so that certain VMs can be accessed from the public Internet.

We need an external load balancer, with a public IP address, that will take traffic from the Internet, and send it to one of the Network Virtual Appliances, as next figure shows:



**Figure x**. LBs in front and behind the NVAs

As it can be seen in the figure, there are several issues that need to be figured out.

First things first, let's have a look at the external load balancer:

The first problem is routing at the NVA. The NVA had a static route for the IP address where the LB probes come from (168.63.129.16) pointing to 10.4.2.1 (eth1, its internal, vnet-facing interface). So that when a probe from the internal load balancer arrives, its answer will be sent down eth0.

However, what happens when a probe arrives from the internal load balancer on eth1? Since the static route is pointing up eth0, it would send the answer there. But this is not going to work, because the answer needs to be sent over the same interface.

You could verify this behaviour capturing traffic on both ports (filtering it to the ports where the probes are configured):

**jose@nva-1:~$ sudo tcpdump -i eth0 port 1138**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

00:28:46.132933 IP 168.63.129.16.58828 > 10.4.2.101.1138: Flags [**SEW**], seq 185817799, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

00:28:49.132924 IP 168.63.129.16.58828 > 10.4.2.101.1138: Flags [**SEW**], seq 185817799, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

00:28:55.133011 IP 168.63.129.16.58828 > 10.4.2.101.1138: Flags [**S**], seq 185817799, win 8192, options [mss 1440,nop,nop,sackOK], length 0

00:29:01.133386 IP 168.63.129.16.59016 > 10.4.2.101.1138: Flags [**SEW**], seq 4020762119, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

00:29:04.133306 IP 168.63.129.16.59016 > 10.4.2.101.1138: Flags [**SEW**], seq 4020762119, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

00:29:10.133712 IP 168.63.129.16.59016 > 10.4.2.101.1138: Flags [**S**], seq 4020762119, win 8192, options [mss 1440,nop,nop,sackOK], length 0

Implement policy based routing:

**sudo vi /etc/iproute2/rt\_tables**

Figure out a different command, maybe with sed???

**$ sudo ip rule add from 10.4.2.101 to 168.63.129.16 lookup slb**

**$ sudo ip route add 168.63.129.16 via 10.4.2.1 dev eth0 table slb**

After implementing policy-based routing:

**jose@nva-1:~$ sudo tcpdump -i eth0 port 1138**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

00:52:16.192365 IP 10.4.2.101.1138 > 168.63.129.16.59630: Flags [S.], seq 2884973469, ack 351896166, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0

00:52:16.192561 IP 168.63.129.16.59630 > 10.4.2.101.1138: Flags [.], ack 1, win 513, options [nop,nop,sack 1 {0:1}], length 0

00:52:16.195535 IP 168.63.129.16.59630 > 10.4.2.101.1138: Flags [F.], seq 1, ack 1, win 513, length 0

00:52:16.195565 IP 168.63.129.16.59804 > 10.4.2.101.1138: Flags [SEW], seq 4142663484, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

00:52:16.195596 IP 10.4.2.101.1138 > 168.63.129.16.59804: Flags [S.], seq 118673669, ack 4142663485, win 29200, options [mss 1460], length 0

00:52:16.195683 IP 168.63.129.16.59804 > 10.4.2.101.1138: Flags [.], ack 1, win 64240, length 0

00:52:16.507823 IP 168.63.129.16.59630 > 10.4.2.101.1138: Flags [F.], seq 1, ack 1, win 513, length 0

C

Now we can connect to the VM from outside, using port 1022:

**jose@MININT-7CE6LKT:~$ ssh -p 1022 lab-user@52.174.188.207**

The authenticity of host '[52.174.188.207]:1022 ([52.174.188.207]:1022)' can't be established.

ECDSA key fingerprint is 74:1f:d0:f9:fc:6a:0c:bc:d7:ee:d7:96:90:fd:79:b0.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '[52.174.188.207]:1022' (ECDSA) to the list of known hosts.

lab-user@52.174.188.207's password:

Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86\_64)

...

lab-user@myVnet3vm:~$

# Lab 9: Advanced HTTP-based probes

Standard TCP probes only verify that the interface being probed answers to TCP sessions. But what if it is the other interface that has an issue? What good does it make if VMs send all traffic to a Network Virtual Appliance with a perfectly working internal interface (eth0 in our lab), but eth1 is down, and therefore that NVA has no Internet access whatsoever?

HTTP probes can be implemented for that purpose. The probes will call for an HTTP URL that will return different HTTP codes, after verifying that all connectivity for the specific NVA is OK. We will use PHP for this, and a script that pings a series of IP addresses or DNS names, both in the Vnet and the public Internet (to verify internal and external connectivity). See the file "index.php" in this repository for more details.

We need to change the probe from TCP-based to HTTP-based:

# Conclusion

# End the lab

To end the lab, simply delete the resource group that you created in the first place (**ansiblelab** in our example) from the Azure portal or from the Azure CLI:

**az group delete --name vnetTest**

## **References**

Useful links:

* Azure network documentation: