Microsoft

Jose Moreno

jose.moreno@microsoft.com, @erjosito

12/6/2017

Azure Networking Lab Guide for  
Hub and Spoke Topologies

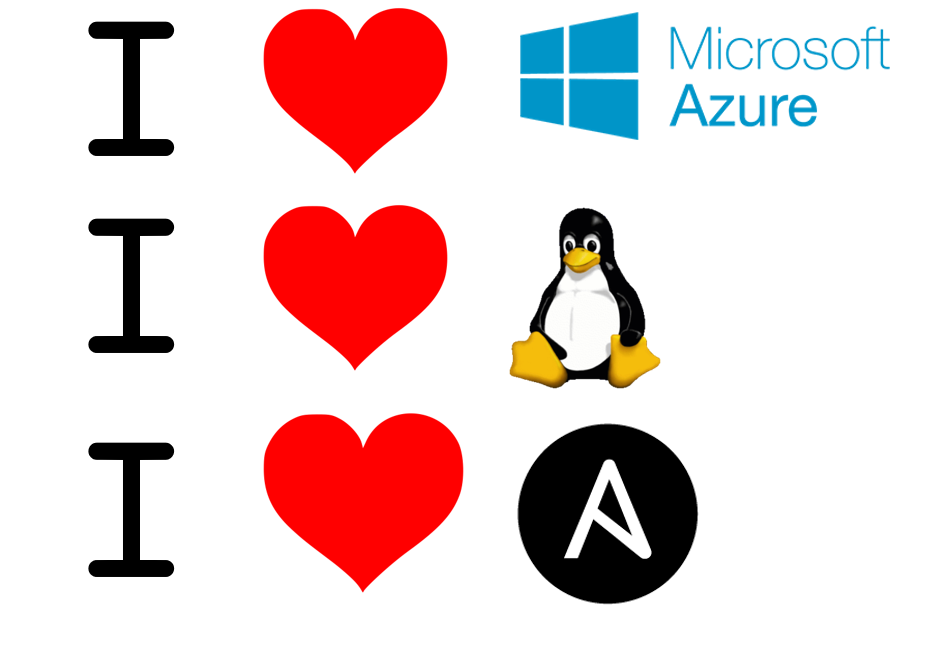


Table of Contents

[Objectives and initial setup 2](#_Toc486579333)

[Introduction to Azure Networking 4](#_Toc486579334)

[Lab 0: Initialize Environment 5](#_Toc486579335)

[Lab 1: Explore Lab environment 8](#_Toc486579336)

[Lab 2: Spoke-to-spoke communication over NVA 13](#_Toc486579337)

[Lab 3: Microsegmentation with NVA 22](#_Toc486579338)

[Lab 4: NVA scalability 26](#_Toc486579339)

[Lab 5: Outgoing Internet Traffic Protected by NVA 32](#_Toc486579340)

[Lab 6: Incoming Internet Traffic Protected by NVA (OPTIONAL) 37](#_Toc486579341)

[Lab 7: Advanced HTTP-based probes (OPTIONAL) 45](#_Toc486579342)

[Lab 8: Spoke-to-Spoke communication over vPN gateway (OPTIONAL) 49](#_Toc486579343)

[Lab 9: VPN connection to the Hub Vnet (OPTIONAL) 55](#_Toc486579344)

[End the lab 64](#_Toc486579345)

[Conclusion 65](#_Toc486579346)

[References 66](#_Toc486579347)

# 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/) 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).
* 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
* Traffic filtering in Azure with firewalls
* Microsegmentation using firewalls
* Scaling out NVAs with load balancing and SNAT
* Advanced probes for Azure Load Balancers
* Linux custom routing

**Important note**:

This lab has been modified to improve the user's experience. Testing with Virtual Network Gateways has been taken all the way to the end, since just the gateway deployment can take up to 45 minutes. The activities in this lab has been divided in 3 sections:

* Section 1: Hub and Spoke networking (around **60 minutes**)
* Section 2: NVA scalability with Azure Load Balancer (around **90 minutes**)
* Section 3: using VPN gateway for spoke-to-spoke connectivity and site-to-site access (around **60 minutes**, not including the time required to provision the gateways)

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 |

As tip, if you want to do the VPN lab, it might be beneficial to run the commands in Lab8 Step1 as you are doing the previous labs, so that you don’t need to wait for 45 minutes (that is more or less the time it takes to provision VPN gateways) when you arrive to Lab8.

# Introduction to Azure Networking

Microsoft Azure has established as one of the leading cloud providers, and part of Azure's offering is Infrastructure as a Service (IaaS), that is, provisioning raw data center infrastructure constructs (virtual machines, networks, storage, etc), so that any application can be installed on top.

An important part of this infrastructure is the network, and Microsoft Azure offers multiple network technologies that can help to achieve the applications' business objectives: from VPN gateways that offer secure network access to load balancers that enable application (and network, as we will see in this lab) scalability.

Some organizations have decided to complement Azure Network offering with Network Virtual Appliances (NVAs) from traditional network vendors. This lab will focus on the integration of these NVAs, and we will take as example an open source firewall, that will be implemented with iptables running on top of an Ubuntu VM with 2 network interfaces. This will allow to highlight some of the challenges of the integration of this sort of VMs, and how to solve them.

At the end of this guide you will find a collection of useful links, but if you don’t know where to start, here is the home page for the documentation for Microsoft Azure Networking: https://docs.microsoft.com/en-us/azure/#pivot=services&panel=network.

The second link you want to be looking at is this document, where Hub and Spoke topologies are discussed: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke.

If you find any issue when running through this lab or any error in this guide, please open a Github issue in this repository, and we will try to fix it. Enjoy!

# Lab 0: Initialize Environment

1. Log into your system. If you are using the Learn On Demand lab environment, the user for the Windows VM is **Admin**, with the password **Passw0rd!**
2. If you don’t have a valid Azure subscription, but have received a voucher code for Azure, go to <https://www.microsoftazurepass.com/Home/HowTo> for instructions about how to redeem it.
3. Open a terminal window. In Windows, for example by hitting the Windows key in your keyboard, typing "cmd" (without the quotes) and hitting the Enter key. You might want to maximize the command Window so that it fills your desktop.
4. 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.

You can copy the following command from this guide with Ctrl-C, and paste it into your terminal window using the **Command** menu (lighting bolt), and select **Paste** | **Paste Clipboard Text**

**az login**

To sign in, use a web browser to open the page https://aka.ms/devicelogin and enter the code XXXXXXXXX to authenticate.

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>. Open an Internet browser, go to this URL, and after introducing the code, you will need to authenticate with credentials that are associated to a valid Azure subscription. After a successful login, you can enter the following two commands back in the terminal window in order to create a new resource group, and to set the default resource group accordingly.

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

You might get an error message in the previous message if you have multiple subscriptions. If that is the case, you can select the subscription where you want to deploy the lab with the command "az account set --subscription <your subscription GUID>. If you did not get any error message, you can safely ignore this paragraph.

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

1. Deploy the master template that will create our initial network configuration. The syntax here depends on the operating system you are using. For example, for **Windows** use this command:

**az group deployment create --name netLabDeployment --template-uri https://raw.githubusercontent.com/erjosito/azure-networking-lab/master/NetworkingLab\_master.json --resource-group vnetTest --parameters "{\"createVPNgw\":{\"value\":\"no\"}, \"adminUsername\":{\"value\":\"lab-user\"}, \"adminPassword\":{\"value\":\"Microsoft123!\"}}"**

Or alternatively use the following command if you are using a **Linux** operative system:

**az group deployment create --name netLabDeployment --template-uri https://raw.githubusercontent.com/erjosito/azure-networking-lab/master/NetworkingLab\_master.json --resource-group vnetTest --parameters '{"createVPNgw":{"value":"no"}, "adminUsername":{"value":"lab-user"}, "adminPassword":{"value":"Microsoft123!"}}'**

1. Since the previous command will take a while (around 15 minutes), open another command window (see Step 3 for detailed instructions) to monitor the deployment progress. Note you might have to login in this second window too:

**az group deployment list -o table**

Name Timestamp State

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

Name Timestamp State

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

myVnet1gwPip 2017-06-29T19:15:28.204648+00:00 Succeeded

myVnet5gwPip 2017-06-29T19:15:28.227920+00:00 Succeeded

myVnet3gwPip 2017-06-29T19:15:28.315235+00:00 Succeeded

myVnet4gwPip 2017-06-29T19:15:31.617920+00:00 Succeeded

myVnet2gwPip 2017-06-29T19:15:32.969018+00:00 Succeeded

myVnet5-vm1-nic 2017-06-29T19:15:38.468886+00:00 Succeeded

myVnet1VpnGw 2017-06-29T19:15:38.565418+00:00 Succeeded

myVnet5VpnGw 2017-06-29T19:15:39.056567+00:00 Succeeded

myVnet3-vm1-nic 2017-06-29T19:15:39.269138+00:00 Succeeded

myVnet4-vm1-nic 2017-06-29T19:15:39.509990+00:00 Succeeded

myVnet2VpnGw 2017-06-29T19:15:40.576390+00:00 Succeeded

myVnet2-vm1-nic 2017-06-29T19:15:41.003741+00:00 Succeeded

myVnet1-vm1-nic 2017-06-29T19:15:41.143406+00:00 Succeeded

myVnet4VpnGw 2017-06-29T19:15:42.608290+00:00 Succeeded

myVnet3VpnGw 2017-06-29T19:15:44.467581+00:00 Succeeded

myVnet3-vm 2017-06-29T19:20:21.195625+00:00 Succeeded

myVnet4-vm 2017-06-29T19:20:21.856738+00:00 Succeeded

myVnet5-vm 2017-06-29T19:20:38.720152+00:00 Succeeded

myVnet-template-3 2017-06-29T19:20:42.269630+00:00 Succeeded

myVnet-template-4 2017-06-29T19:20:42.581238+00:00 Succeeded

myVnet1-vm 2017-06-29T19:20:42.663423+00:00 Succeeded

myVnet-template-5 2017-06-29T19:20:55.790429+00:00 Succeeded

myVnet-template-1 2017-06-29T19:20:59.148149+00:00 Succeeded

myVnet2-vm 2017-06-29T19:21:11.406220+00:00 Succeeded

myVnet-template-2 2017-06-29T19:21:29.787517+00:00 Succeeded

vnets 2017-06-29T19:21:32.490251+00:00 Succeeded

vnet5Gw 2017-06-29T19:21:52.630513+00:00 Succeeded

vnet4Gw 2017-06-29T19:21:53.090784+00:00 Succeeded

linuxnva-1-nic0 2017-06-29T19:21:56.417810+00:00 Succeeded

UDRs 2017-06-29T19:21:57.487004+00:00 Succeeded

linuxnva-2-nic0 2017-06-29T19:21:58.322935+00:00 Succeeded

myVnet1-vm2-nic 2017-06-29T19:21:58.354766+00:00 Succeeded

linuxnva-2-nic1 2017-06-29T19:21:59.328610+00:00 Succeeded

linuxnva-1-nic1 2017-06-29T19:21:59.490821+00:00 Succeeded

nva-slb-int 2017-06-29T19:22:03.146781+00:00 Succeeded

hub2spoke2 2017-06-29T19:22:09.140223+00:00 Succeeded

hub2spoke3 2017-06-29T19:22:10.229240+00:00 Succeeded

hub2spoke1 2017-06-29T19:22:12.980700+00:00 Succeeded

AzureLB 2017-06-29T19:22:15.587843+00:00 Succeeded

nva-slb-ext 2017-06-29T19:22:18.507575+00:00 Succeeded

vnet1subnet1vm2 2017-06-29T19:26:53.109076+00:00 Succeeded

nva 2017-06-29T19:29:24.679832+00:00 Succeeded

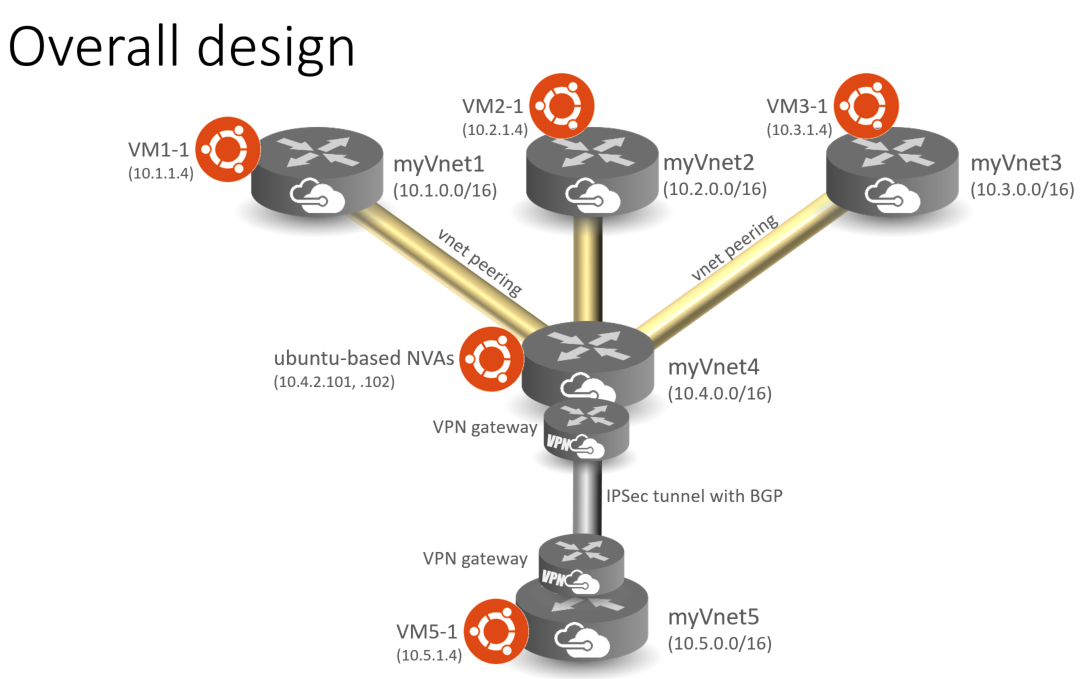
netLabDeployment 2017-06-29T19:29:26.491546+00:00 Succeeded

# Lab 1: Explore Lab environment

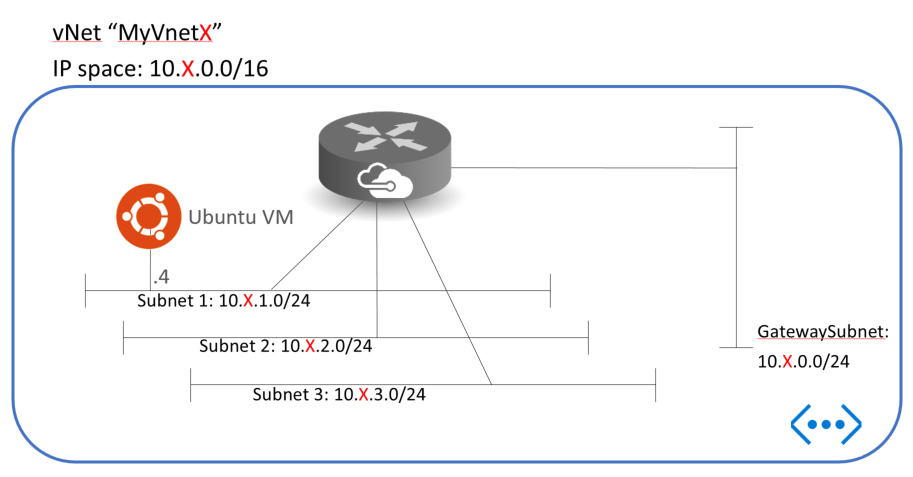
1. You don’t need to wait until all objects in the template have been successfully deployed (although it would be good, to make sure that everything is there). In your second terminal window, start exploring the objects created by the ARM template: vnets, subnets, VMs, interfaces, public IP addresses, etc. Save the output of these commands (copying and pasting to a text file for example).

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

Note that the output of these commands might be different, if the template deployment from lab 0 is not completed yet.



**Figure 1**. Overall vnet diagram



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

**az network vnet list -o table**

Location Name ProvisioningState ResourceGroup ResourceGuid

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

westeurope myVnet1 Succeeded vnetTest 1d20ba9a...

westeurope myVnet2 Succeeded vnetTest 43ca80d0...

westeurope myVnet3 Succeeded vnetTest 4837a481...

westeurope myVnet4 Succeeded vnetTest 72a82a72...

westeurope myVnet5 Succeeded vnetTest 96e5f9c5...

***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

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

linuxnva-1 vnetTest westeurope

linuxnva-2 vnetTest westeurope

myVnet1-vm1 vnetTest westeurope

myVnet1-vm2 vnetTest westeurope

myVnet2-vm1 vnetTest westeurope

myVnet3-vm1 vnetTest westeurope

myVnet4-vm1 vnetTest westeurope

myVnet5-vm1 vnetTest westeurope

**az network nic list -o table**

EnableIpForwarding Location MacAddress Name

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

True westeurope 00-0D-3A-28-F8-F9 linuxnva-1-nic0

True westeurope 00-0D-3A-28-F0-3A linuxnva-1-nic1

True westeurope 00-0D-3A-28-24-73 linuxnva-2-nic0

True westeurope 00-0D-3A-28-2A-28 linuxnva-2-nic1

westeurope 00-0D-3A-2A-48-AF myVnet1-vm1-nic

westeurope 00-0D-3A-28-2C-8C myVnet1-vm2-nic

westeurope 00-0D-3A-2A-4A-DE myVnet2-vm1-nic

westeurope 00-0D-3A-2A-46-DE myVnet3-vm1-nic

westeurope 00-0D-3A-2A-4F-EA myVnet4-vm1-nic

westeurope 00-0D-3A-2A-47-BC myVnet5-vm1-nic

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

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

Name PublicIpAllocationMethod ResourceGroup IpAddress

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

linuxnva-slbpip Dynamic vnetTest 11.11.11.11

myVnet1vm1pip Dynamic vnetTest 1.1.1.1

myVnet1vm2pip Dynamic vnetTest 1.1.1.2

myVnet2vm1pip Dynamic vnetTest 2.2.2.2

myVnet3vm1pip Dynamic vnetTest 3.3.3.3

myVnet4vm1pip Dynamic vnetTest 4.4.4.4

myVnet5vm1pip Dynamic vnetTest 5.5.5.5

nvaPip-1 Dynamic vnetTest 6.6.6.6

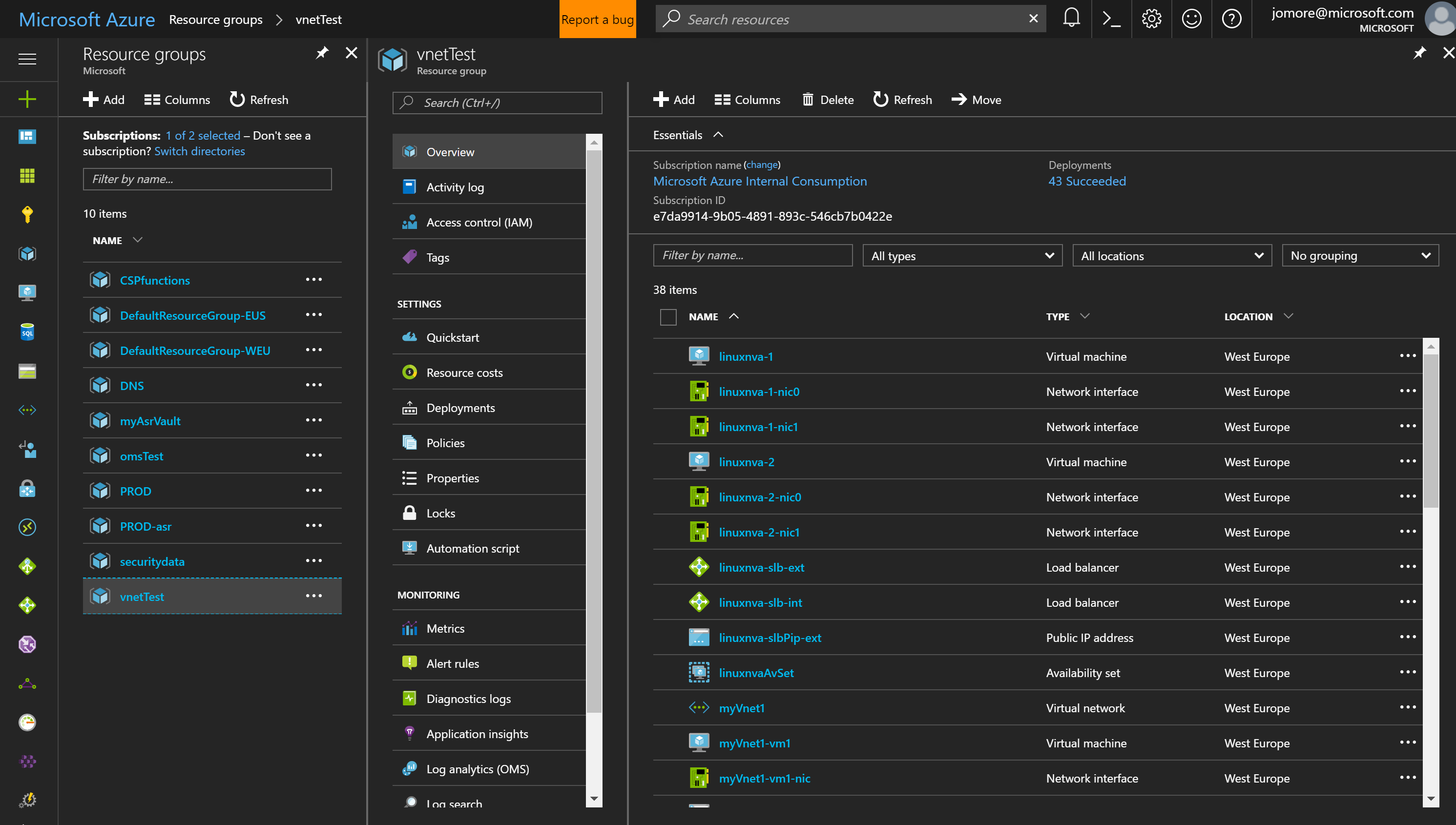
nvaPip-2 Dynamic vnetTest 7.7.7.7

vnet4gwPip Dynamic vnetTest

vnet5gwPip Dynamic vnetTest

***Note****: Some columns of the ouput above have been removed for clarity purposes. Furthermore, the public IP addresses in the table are obviously not the ones you will see in your environment.* ***Note the actual public IP addresses in your environment*** *somewhere (like a Notepad window), since you will be needing them for the rest of the lab.*

1. Connect to the Azure portal (<http://portal.azure.com>) and locate the resource group that we have just created (called "**vnetTest**", if you did not change it). Verify the objects that have been created and explore their properties and states.



**Figure 4**: Azure portal with the resource group created for this lab

**NOTE**: you might want to open two new additional Windows command prompt windows and launch the two commands from Lab 8, Step 1. Each of those commands (can be run in parallel) will take around 45 minutes, so you can leave them running while you proceed with Labs 2 through 7. If you are not planning to run Labs 8-9, you can safely ignore this paragraph.

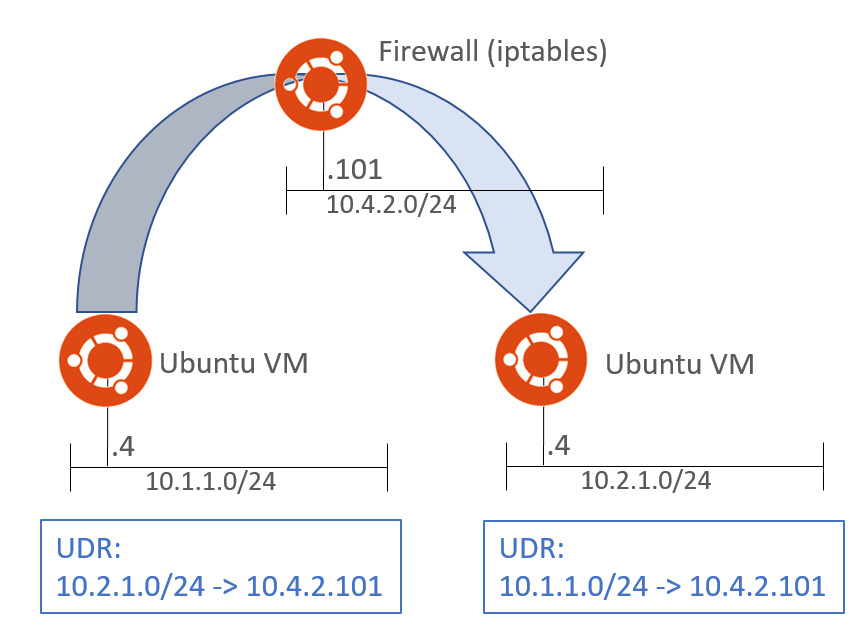
**PART 1**

**Hub and Spoke Networking**

# Lab 2: 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 5**. 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 (look for the ProvisioningState column):

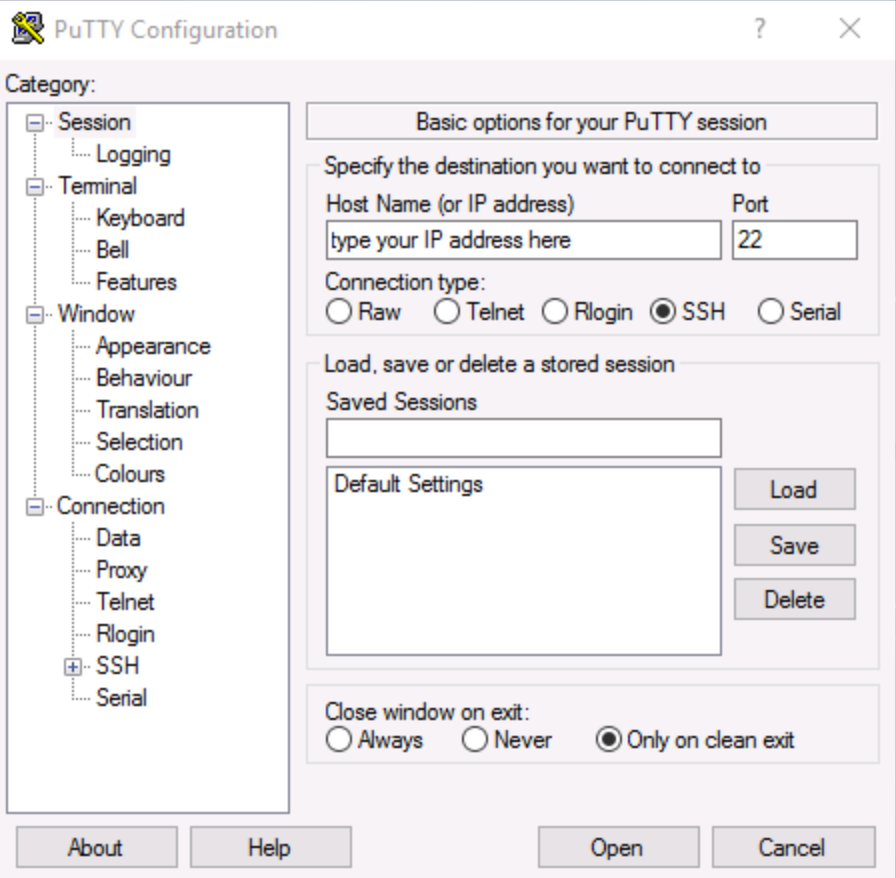
**az vm extension list --vm-name linuxnva-1 -o table**

AutoUpgradeMinorVersion Location Name ProvisioningState

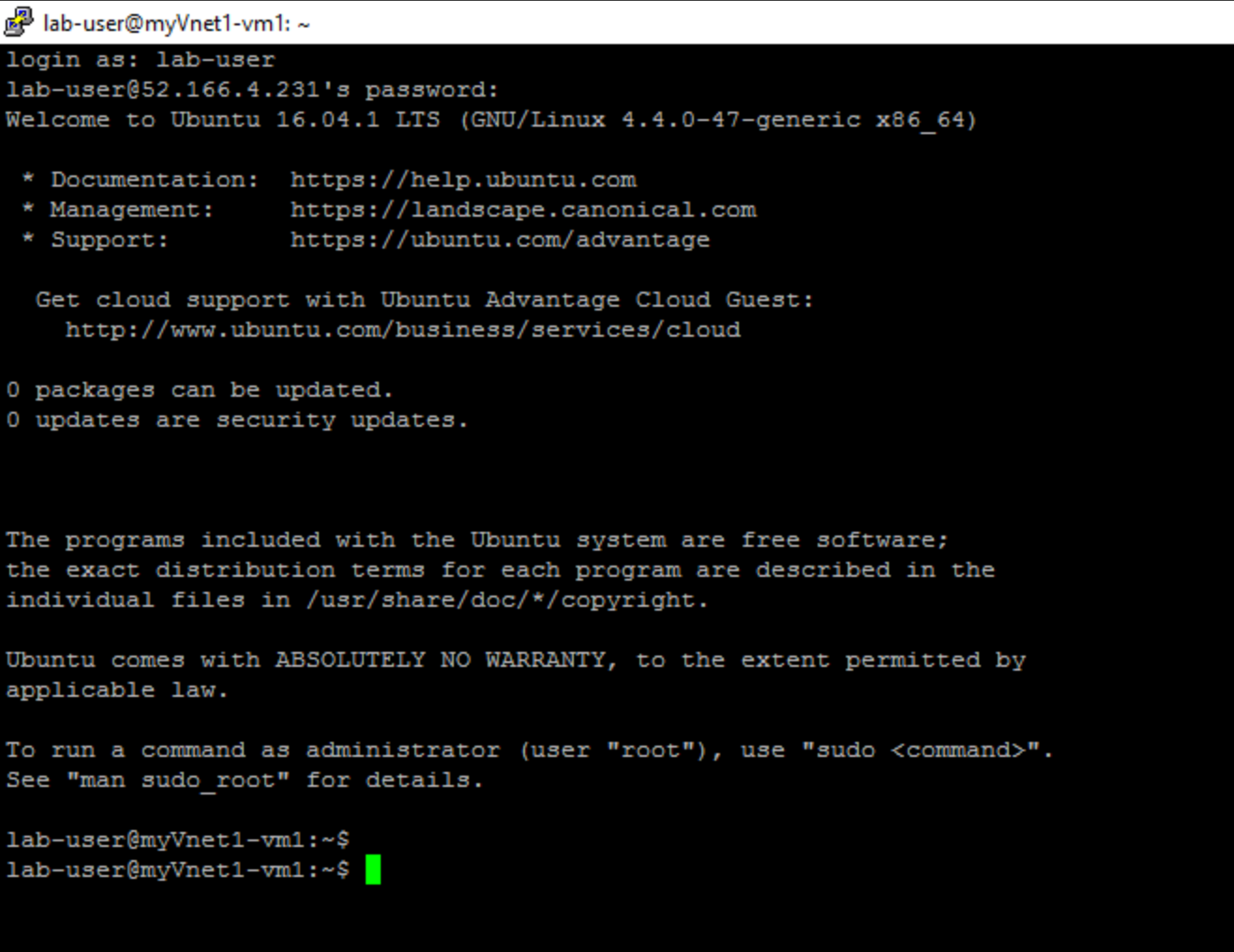
------------------------- ---------- ------------------- -----------------

True westeurope installcustomscript Succeeded

1. After verifying the public IP address assigned to the first VM in vnet1 (called "myVnet1-vm1-pip", go back to the your list of IP addresses), 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. You can open an SSH session from the Linux **bash** shell in Windows (the red, circular icon in your taskbar), or you can use Putty (pre-installed in the Lab VM, you should have a link on your task bar). The following screenshots show you how to open Putty and use it to connect to a remote system over SSH:



***Note****: please note the IP address for your VM will be unique, you can get the IP address assigned to "myVnet1-vm1-pip" with the command "az network public-ip list -o table". Type it in the "Host Name (or IP address)" text box in the dialog window above, and then click on "Open".*



The username and password were specified at creation time (that long command that invoked the ARM template). If you did not change the parameters, the username is "lab-user" and the password "Microsoft123!" (without the quotes).

1. Try to connect to the private IP address of the VM in vnet2 over SSH. We can use the private IP address, because now we are inside of the vnet.

lab-user@myVnet1-vm1:~$ **ssh 10.2.1.4**

ssh: connect to host 10.2.1.4 port 22: Connection timed out

lab-user@myVnet1-vm1:~$

***Note****: you do not need to wait for the "ssh 10.2.1.4" command to time out if you do not want to. Besides, if you are wondering why we are not testing with a simple ping, the reason is because the NVAs are preconfigured to drop ICMP traffic, as we will see in later labs.*

1. Back in the command prompt window, 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 | findstr routeTable**

"routeTable": null

***Note****: if you are using the Azure CLI on a Linux system, replace the "findstr" command in the previous step with "grep"*

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

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

{

"etag": "W/\"...\"",

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

"location": "westeurope",

"name": "vnet1-subnet1",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"routes": [],

"subnets": null,

"tags": null,

"type": "Microsoft.Network/routeTables"

}

**az network route-table create --name vnet2-subnet1**

{

"etag": "W/\"...\"",

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

"location": "westeurope",

"name": "vnet2-subnet1",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"routes": [],

"subnets": null,

"tags": null,

"type": "Microsoft.Network/routeTables"

}

1. Verify that the route tables are successfully created:

**az network route-table list -o table**

Location Name ProvisioningState ResourceGroup

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

westeurope vnet1-subnet1 Succeeded vnetTest

westeurope vnet2-subnet1 Succeeded vnetTest

1. Now attach 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**

*Output omitted*

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

*Output omitted*

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 | findstr routeTable**

"routeTable": {

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

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

"routeTable": {

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

***Note****: if you are using the Azure CLI on a Linux system, replace the "findstr" commands in the previous step with "grep"*

1. Now we can 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**

{

"addressPrefix": "10.2.0.0/16",

"etag": "W/\"b18dd5db-4ff4-4de0-b69b-2de5c7f21985\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1/routes/**vnet2**",

"name": "vnet2",

"nextHopIpAddress": null,

"nextHopType": "VnetLocal",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

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 myVnet1-vm1-nic**

...

{

"addressPrefix": [

"**10.2.0.0/16**"

],

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

"nextHopIpAddress": [],

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

"source": "User",

"state": "Active"

},

***Note****: the previous command takes some seconds to run, since it access the routing programmed into the NIC. If you cannot find the route with the addressPrefix 10.2.0.0/16 (at the bottom of the output), please wait a few seconds and issue the command again, sometimes it takes some time to program the NICs in Azure*

That essentially means, that we need a different method to send spoke-to-spoke traffic, and the native vnet router just will not cut it. For this purpose, we will use the Network Virtual Appliance (our virtual Linux-based firewall) as next-hop. In other words, you need an additional routing device (in this case the NVA, it could be the VPN gateway) other than the standard vNet routing functionality.

1. Now we will install in each route table routes for the other side, but this time pointing to the private IP address of the Network Virtual Appliance in vnet 4.

**az network route-table route update --address-prefix 10.2.0.0/16 --next-hop-ip-address 10.4.2.101 --next-hop-type VirtualAppliance --route-table-name vnet1-subnet1 -n vnet2**

{

"addressPrefix": "10.2.0.0/16",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1/routes/vnet2",

"name": "vnet2",

"nextHopIpAddress": "10.4.2.101",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

**az network route-table route create --address-prefix 10.1.0.0/16 --next-hop-ip-address 10.4.2.101 --next-hop-type VirtualAppliance --route-table-name vnet2-subnet1 -n vnet1**

{

"addressPrefix": "10.1.0.0/16",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet2-subnet1/routes/vnet1",

"name": "vnet1",

"nextHopIpAddress": "10.4.2.101",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

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 10.4.2.101 VirtualAppliance Succeeded

**az network nic show-effective-route-table -n myVnet1-vm1-nic**

...

{

"addressPrefix": [

"10.2.0.0/16"

],

"name": "vnet2",

"nextHopIpAddress": [

**"10.4.2.101"**

],

"nextHopType": "VirtualAppliance",

"source": "User",

"state": "Active"

}

***Note****: the previous command takes some seconds to run, since it access the routing programmed into the NIC. If you cannot find the route with the addressPrefix 10.2.0.0/16 (at the bottom of the output), please wait a few seconds and issue the command again, sometimes it takes some time to program the NICs in Azure*

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

**What we have learnt**

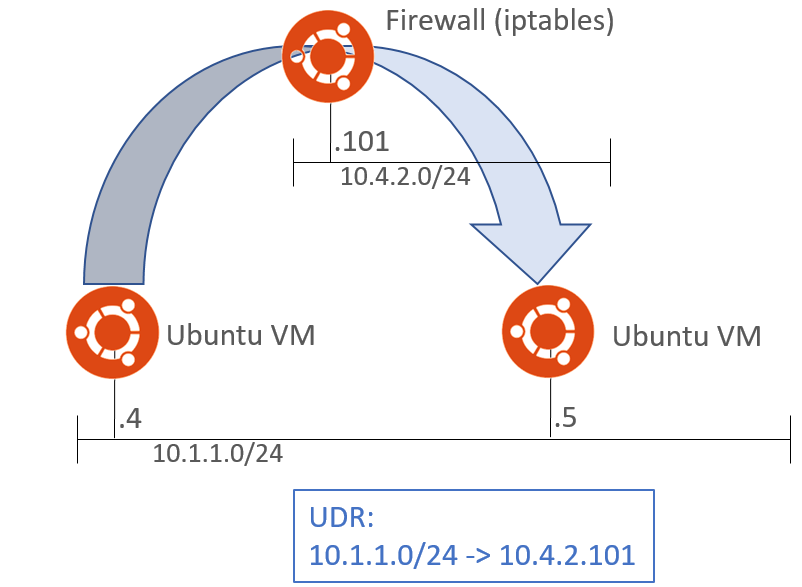
With peered vnets it is not enough to send traffic to the vnet in order to get spoke-to-spoke communication, but you need to steer it to an NVA (or to a VPN/ER Gateway, as we will see in a later lab) via User-Defined Routes (UDR).

UDRs can be used steer traffic between subnets through a firewall. The UDRs should point to the IP address of a firewall interface in a different subnet. This firewall could be even in a peered vnet, as we demonstrated in this lab, where the firewall was located in the hub vnet.

You can verify the routes installed in the routing table, as well as the routes programmed in the NICs of your VMs. Note that discrepancies between the routing table and the programmed routes can be extremely useful when troubleshooting routing problems.

# Lab 3: 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 6**. Intra-subnet NVA-based filtering, also known as “microsegmentation”

1. In order to be able to test the topology above, we will use the second VM in myVnet1-Subnet1. (vnet1-vm2). We need to instruct all VMs in subnet 1 to send local traffic to the NVAs as well. First, let us verify that both VMs can reach each other. Exit the session from Vnet2-vm1 to come back to Vnet1-vm1, and verify that you can reach its neighbor VM in 10.1.1.5:

lab-user@**myVnet2-vm1**:~$ exit

logout

Connection to 10.2.1.4 closed.

lab-user@**myVnet1-vm1**:~$ **ping 10.1.1.5**

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

64 bytes from 10.1.1.5: icmp\_seq=1 ttl=64 time=0.612 ms

64 bytes from 10.1.1.5: icmp\_seq=2 ttl=64 time=3.62 ms

64 bytes from 10.1.1.5: icmp\_seq=3 ttl=64 time=2.71 ms

64 bytes from 10.1.1.5: icmp\_seq=4 ttl=64 time=0.748 ms

^C

--- 10.1.1.5 ping statistics ---

4 packets transmitted, 4 received, **0% packet loss**, time 3002ms

rtt min/avg/max/mdev = 0.612/1.924/3.628/1.287 ms

lab-user@myVnet1-vm1:~$

1. We want to be able to control traffic flowing between the two VMs, even if they are in the same subnet. For that purpose, we want to send this traffic to our NVA (firewall). This can be easily done by adding an additional User-Defined Route to the corresponding routing table. Go back to your Windows command prompt, and type this command:

**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**

{

"addressPrefix": "10.1.1.0/24",

"etag": "W/\"...\"",

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

"name": "vnet1-subnet1",

"nextHopIpAddress": "10.4.2.101",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

***Note****: this command needs to be issued from your machine, outside of the putty window*

1. If you go back to the Putty window and restart the ping, you will notice that after some seconds ping will stop working. Traffic does not stop to work immediately because of the time it takes Azure to reprogram the User-Defined Routes in every NIC.

lab-user@**myVnet1-vm1**:~$ **ping 10.1.1.5**

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

64 bytes from 10.1.1.5: icmp\_seq=1 ttl=64 time=2.22 ms

64 bytes from 10.1.1.5: icmp\_seq=2 ttl=64 time=0.847 ms

...

64 bytes from 10.1.1.5: icmp\_seq=30 ttl=64 time=0.762 ms

64 bytes from 10.1.1.5: icmp\_seq=31 ttl=64 time=0.689 ms

64 bytes from 10.1.1.5: icmp\_seq=32 ttl=64 time=3.00 ms

^C

--- 10.1.1.5 ping statistics ---

98 packets transmitted, 32 received, 67% packet loss, time 97132ms

rtt min/avg/max/mdev = 0.620/1.160/4.284/0.766 ms

lab-user@myVnet1-vm1:~$

1. To verify that routing is still correct, you can now try SSH instead of ping. The fact that SSH works, but ping does not, demonstrates that the traffic is being dropped by the NVA.

lab-user@my**Vnet1-vm1**:~$ **ssh 10.1.1.5**

**What we have learnt**

UDRs can be used not only to steer traffic between subnets through a firewall, but to steer traffic even between hosts inside of one subnet through a firewall too. This is due to the fact that Azure routing is not performed at the subnet level, as in traditional networks, but at the NIC level. This enables a very high degree of granularity

As a side remark, in order for these microsegmentation designs to work, the firewall needs to be in a separate subnet from the VMs themselves, otherwise the UDR will provoke a routing loop.

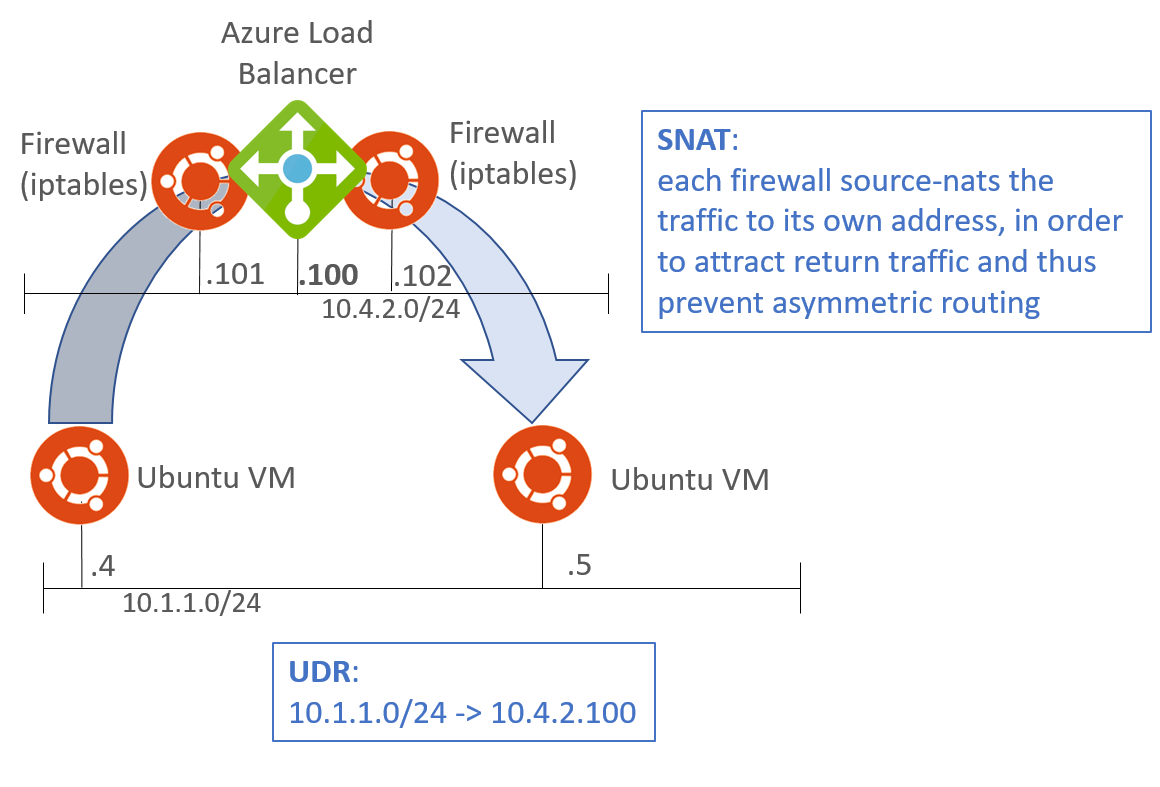
**PART 2**

**NVA High Availability**

# Lab 4: 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 7**. 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, go to your Windows command window to verify that both an internal and an external load balancer have been deployed:

**az network lb list -o table**

Location Name ProvisioningState ResourceGroup

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

westeurope linuxnva-slb-ext Succeeded vnetTest

westeurope linuxnva-slb-int Succeeded vnetTest

1. Now get information about the object names inside of the **internal** Load Balancer. The following command can be used in order to get the names for the objects inside of the load balancer. Specifically we will need the name of the backend farm, highlighted in green:

**az network lb show -n linuxnva-slb-int | findstr name**

"name": "**linuxnva-slbBackend-int**",

"name": "myFrontendConfig",

"name": null,

"name": "ssh",

"name": "linuxnva-slb-int",

"name": "myProbe",

***Note****: if you are running this step from a Linux system, please replace the command "findstr" with "grep"*

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 linuxnva-1-nic0-ipConfig --nic-name linuxnva-1-nic0 --address-pool linuxnva-slbBackend-int --lb-name linuxnva-slb-int**

*Output omitted*

And the same command (observe the differences in red) for our second Linux-based NVA appliance:

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

*Output omitted*

1. Let us verify the LB's rules. In this case, we need to remove the existing one (that was created by default by the ARM template in the very first lab) and replace it with another, where we will enable Direct Server Return:

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

BackendPort FrontendPort LoadDistribution Name Protocol

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

22 1022 Default **ssh** Tcp

**az network lb rule delete --lb-name linuxnva-slb-int -n ssh**

***Note****: the previous command will require some minutes to run*

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

{

"**backend**AddressPool": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/backendAddressPools/linuxnva-slbBackend-int",

"resourceGroup": "vnetTest"

},

"backendPort": 22,

"enableFloatingIp": true,

"etag": "W/\"...\"",

"**frontend**IpConfiguration": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/frontendIPConfigurations/myFrontendConfig",

"resourceGroup": "vnetTest"

},

"frontendPort": 22,

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/loadBalancingRules/sshRule",

"idleTimeoutInMinutes": 4,

"loadDistribution": "Default",

"name": "**sshRule**",

"probe": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/probes/myProbe",

"resourceGroup": "vnetTest"

},

"protocol": "Tcp",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

1. Verify with the following command the fronted IP address that the load balancer has been preconfigured with (with the ARM template in the very first lab):

**az network lb frontend-ip list --lb-name linuxnva-slb-int -o table**

Name PrivateIpAddress PrivateIpAllocationMethod

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

myFrontendConfig **10.4.2.100** Static

***Note****: some columns have been removed from the previous output for simplicity*

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). Remember that we configured that route to point to 10.4.2.101, the IP address of one of the firewalls. 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, on the TCP ports specified by Load Balancer rules. Note that ICMP will still not work, but in this case, this is due to the fact that at this point in time, the Azure Load Balancer is not able to balance ICMP traffic, just TCP or UDP traffic (as configured by the load balancing rules, that require a TCP or a UDP port), so Pings do not even reach the NVAs.

If you go back to the Putty window, you can verify that ping to the neighbor VM in the same subnet still does not work, but SSH does.

lab-user@**myVnet1-vm1**:~$ **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 ---

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

lab-user@**myVnet1-vm1**:~$ **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. This is due to the source NAT that firewalls do, in order to make sure that return traffic from myVnet1-vm2 goes through the NVA as well:

**lab-user@ myVnet1-vm2:~$ who**

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

1. This is expected, since firewalls are configured to source NAT the connections outgoing on that interface. Now open another Putty window, and connect over SSH to the public IP address of the firewall. Remember that you can retrieve the list of public IP address with the command "az network public-ip list -o table". Please go to the same firewall that you just saw in the previous step. That is, if in the "who" command you saw the IP address 10.4.2.101, connect with Putty to the public IP address " nvaPip-1", if you saw 10.4.2.102, connect to " nvaPip-2". In our example from the output above, we saw the "10.4.2.101", so we will connect to the first NVA. Remember that the username is still "lab-user", the password "Microsoft123!" (without the quotes).  
   After connecting to the firewall, you can display the NAT configuration with the following command:

lab-user@linuxnva-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**

***Note****: the Linux machines that we use as firewalls in this lab have the Linux package "iptables" installed to work as firewall. A tutorial of iptables is out of the scope of this lab guide. Suffice to say here that the word "MASQUERADE" means to translate the source IP address of packets and replace it with its own interface address. In other words, source-NAT.*

1. We will simulate a failure of the NVA where the connection is going through (in this case 10.4.2.101, linuxnva-1). First of all, verify that both ports 1138 (used by the internal load balancer of this lab scenario) and 1139 (used by the external load balancer of a lab scenario later in this guide) are open:

lab-user@linuxnva-1:~$ **nc -zv -w 1 127.0.0.1 1138-1139**

Connection to 127.0.0.1 1138 port [tcp/\*] succeeded!

Connection to 127.0.0.1 1139 port [tcp/\*] succeeded!

***Note****: in this example we use the Linux command nc (aka netcat) to open TCP connections to those two ports*

The process answering to TCP requests on those ports is netcat (represented by "nc"), as you can see with netstat:

lab-user@linuxnva-1:~$ **sudo netstat -lntp**

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program

**tcp 0 0 0.0.0.0:1138 0.0.0.0:\* LISTEN 1783/nc**

**tcp 0 0 0.0.0.0:1139 0.0.0.0:\* LISTEN 1782/nc**

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1587/sshd

tcp6 0 0 :::80 :::\* LISTEN 11730/apache2

tcp6 0 0 :::22 :::\* LISTEN 1587/sshd

1. We will shutdown interface eth0 in the firewall where the connection was going through (the address you saw in the "who" command):

lab-user@linuxnva-1:~$ **sudo ifconfig eth0 down**

If you go back to the first Putty window, with the SSH connection to myVnet1-vm1, the SSH session should have become irresponsive, since the flow is now broken (we brought down the firewall's network interface where it was going through).

Please open a new Putty window to the public IP address of myVnet1-vm1, and initiate another SSH connection to (10.1.1.5) (myVnet1-vm2). You will see that you are going now through the other NVA (in this example, nva-2). Note that it takes some time (defined by the probe frequency and number, per default two times 15 seconds) until the load balancer decides to take the NVA out of rotation.

lab-user@**myVnet1-vm1**:~$ **ssh 10.1.1.5**

lab-user@10.1.1.5's password:

...

lab-user@myVnet1-vm2:~$

lab-user@myVnet1-vm2:~$ who

lab-user pts/0 2017-06-29 21:21 (10.4.2.101)

lab-user pts/1 2017-06-29 21:39 (10.4.2.102)

lab-user@myVnet1-vm2:~$

1. Bring eth0 interface back up, in the NVA where you shut it down (in the other Putty window):

lab-user@linuxnva-1:~$ **sudo ifconfig eth0 up**

**What we have learnt**

NVAs can be load balanced with the help of an Azure Load Balancer. UDRs configured in each subnet will essentially point not to the IP address of an NVA, but to a virtual IP address configured in the LB.

Note that at the time of this writing no Layer3 load balancing rules can be configured in the load balancer, but only Layer4 rules. That means, that you need to configure a rule for each TCP or UDP port that requires going through the firewall.

Another problem that needs to be solved is return traffic. With stateful network devices such as firewalls you need to prevent asymmetric routing. In other words, source-to-destination traffic needs to go through the same firewall as destination-to-source traffic (for any given TCP or UDP flow). This can be achieved by source-NATting the traffic at the NVAs, so that the destination will always send the return traffic the right way.

# Lab 5: Outgoing Internet Traffic Protected by NVA

What if we want to send all traffic leaving the vnet towards the public Internet through the NVAs? We need to 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. Go back to your Windows command prompt, and create a routing table for myVnet3Subnet1:

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

{

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet3-su

bnet1",

"location": "westeurope",

"name": "vnet3-subnet1",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"routes": [],

"subnets": null,

"tags": null,

"type": "Microsoft.Network/routeTables"

}

1. Now create a default route in that table pointing to the internal LB VIP (10.4.2.100):

**az network route-table route create --address-prefix 0.0.0.0/0 --next-hop-ip-address 10.4.2.100 --next-hop-type VirtualAppliance --route-table-name vnet3-subnet1 -n default**

{

"addressPrefix": "0.0.0.0/0",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet3-su

bnet1/routes/default",

"name": "default",

"nextHopIpAddress": "10.4.2.100",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

1. Associate the route table to the subnet myVnet3Subnet1:

**az network vnet subnet update -n myVnet3Subnet1 --vnet-name myVnet3  
--route-table vnet3-subnet1**

*Output omitted*

1. We want to verify that Vnet3 has connectivity to our other spoke Vnets (Vnet1 and Vnet2). Add another default route for Vnet1Subnet1 pointing to the internal load balancer's VIP in Vnet3Subnet1, and the reciprocal route in the custom routing table for Vnet1Subnet1, and verify that you have SSH connectivity between the VM in Vnet1 and the VM in Vnet3.

**az network route-table route create --address-prefix 10.1.1.0/24 --next-hop-ip-address 10.4.2.100 --next-hop-type VirtualAppliance --route-table-name vnet3-subnet1 -n vnet1subnet1**

{

"addressPrefix": "10.1.1.0/24",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet3-subnet1/routes/vnet1subnet1",

"name": "vnet1subnet1",

"nextHopIpAddress": "10.4.2.100",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

**az network route-table route create --address-prefix 10.3.1.0/24 --next-hop-ip-address 10.4.2.100 --next-hop-type VirtualAppliance --route-table-name vnet1-subnet1 -n vnet3subnet1**

{

"addressPrefix": "10.3.1.0/24",

"etag": "W/\"6174bb9f-38cc-46c9-94c7-c9edf4752dbc\"",

"id": "/subscriptions/e7da9914-9b05-4891-893c-546cb7b0422e/resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1/routes/vnet3subnet1",

"name": "vnet3subnet1",

"nextHopIpAddress": "10.4.2.100",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

Now let us verify that we have connectivity. As usual, we will use ssh, since the firewalls are blocking ICMP traffic:

lab-user@**myVnet1-vm1:~$ ssh 10.3.1.4**

ssh: connect to host 10.3.1.4 port 22: Connection timed out

lab-user@myVnet1vm:~$ ssh 10.3.1.4

The authenticity of host '10.3.1.4 (10.3.1.4)' can't be established.

ECDSA key fingerprint is SHA256:ofxGjkNl2WYq+GvlEUYNTd5WiAlV4Za2/X3BwcpX8hQ.

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

Warning: Permanently added '10.3.1.4' (ECDSA) to the list of known hosts.

lab-user@10.3.1.4's password:

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

...

lab-user@**myVnet3-vm1**:~$

***Note****: as in the example above, it might happen that your first SSH attempt does not work, if the routes have not been programmed in the VM NICs just yet. If that is the case, please wait a few seconds and try again.*

1. In the Putty window with the connection of your NVA, verify that the NVAs are source-NATting (known as "masquerading" in iptables speech) all traffic outgoing its external interface (eth1):

lab-user@linuxnva-1:~$ **sudo iptables -vL -t nat**

Chain PREROUTING (policy ACCEPT 87329 packets, 3531K bytes)

pkts bytes target prot opt in out source destination

Chain INPUT (policy ACCEPT 48225 packets, 1943K bytes)

pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 2157 packets, 137K bytes)

pkts bytes target prot opt in out source destination

Chain POSTROUTING (policy ACCEPT 29 packets, 1740 bytes)

pkts bytes target prot opt in out source destination

910 61924 MASQUERADE all -- any eth0 anywhere anywhere

**1220 73886 MASQUERADE all -- any eth1 anywhere anywhere**

1. Note that you don’t have Internet access to the VM in myVnet3Subnet1 any more, after changing default routing for that subnet. To verify that, we will use the command curl to connect to the Internet service "ifconfig.co". If successful, the command should return the public IP address of the VM. Run the command to make sure that it does not work yet:

lab-user@my**Vnet3-vm1**:~$ **curl ifconfig.co**

curl: (7) Failed to connect to ifconfig.co port 80: **Connection timed out**

1. In order to provide outgoing Internet access for web traffic (port 80), let's add another rule to the internal load balancer to allow for port 80. For this, please go back to your Windows command prompt:

**az network lb rule create --backend-pool-name linuxnva-slbBackend-int --protocol Tcp --backend-port 80 --frontend-port 80 --frontend-ip-name myFrontendConfig --lb-name linuxnva-slb-int --name httpRule --floating-ip true --probe-name myProbe**

{

"backendAddressPool": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/backendAddressPools/linuxnva-slbBackend-int",

"resourceGroup": "vnetTest"

},

"backendPort": 80,

"enableFloatingIp": true,

"etag": "W/\"...\"",

"frontendIpConfiguration": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/frontendIPConfigurations/myFrontendConfig",

"resourceGroup": "vnetTest"

},

"frontendPort": 80,

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/loadBalancingRules/httpRule",

"idleTimeoutInMinutes": 4,

"loadDistribution": "Default",

"name": "httpRule",

"probe": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/probes/myProbe",

"resourceGroup": "vnetTest"

},

"protocol": "Tcp",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

Now run again our curl command, to verify that it is now working:

**lab-user@myVnet3-vm1:~$ curl ifconfig.co**

52.232.81.172

***Note****: in the previous output you would see your own IP address, which would obviously defer from the one shown in the example above.*

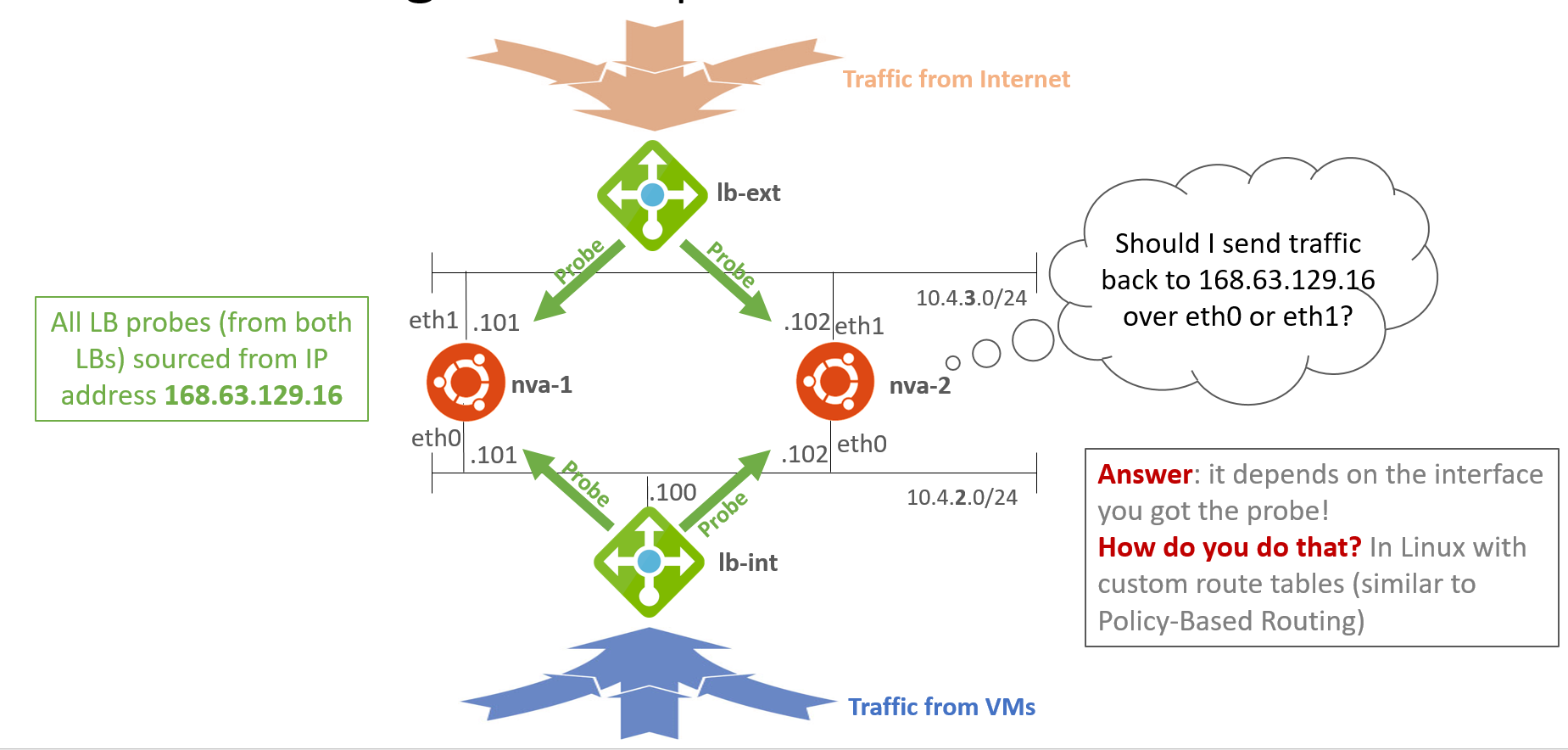
**What we have learnt**

Essentially the mechanism for redirecting traffic going from Azure VMs to the public Internet through an NVA is very similar to the problems we have seen previously in this lab. You need to configure UDRs pointing to the NVA (or to an internal load balancer that sends traffic to the NVA). Source NAT at the firewall will guarantee that the return traffic (destination-to-source) is sent to the same NVA that processed the initial packets (source-to-destination).

# Lab 6: Incoming Internet Traffic Protected by NVA (OPTIONAL)

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

For this 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 8**. LBs in front and behind the NVAs

Note that in the case of an NVA with a single interface, the answer to the question about on which interface the answer to the probe goes is trivial, but in this lab we represent it to illustrate the most frequent situation where 3rd-pary firewalls have separate internal and external interfaces.

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

1. First things first, let's have a look at the external load balancer from our Windows command prompt:

**az network lb list -o table**

Location Name ProvisioningState ResourceGroup

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

westeurope **nva-slb-ext** Succeeded vnetTest

westeurope nva-slb-int Succeeded vnetTest

**az network lb show -n linuxnva-slb-ext | findstr name**

"name": "linuxnva-slbBackend-ext",

"name": "myFrontendConfig",

"name": null,

"name": "mySLBConfig",

"name": "linuxnva-slb-ext",

"name": "myProbe",

***Note****: if you are running the previous step from a Linux machine, please replace the command "findstr" with "grep"*

1. Now you can add the external interfaces of the NVAs to the backend address pool of the external load balancer:

**az network nic ip-config address-pool add --ip-config-name linuxnva-1-nic1-ipConfig --nic-name linuxnva-1-nic1 --address-pool linuxnva-slbBackend-ext --lb-name linuxnva-slb-ext**

*Output omitted*

**az network nic ip-config address-pool add --ip-config-name linuxnva-2-nic1-ipConfig --nic-name linuxnva-2-nic1 --address-pool linuxnva-slbBackend-ext --lb-name linuxnva-slb-ext**

*Output omitted*

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 and configure both sides on port 22:

**az network lb rule list --lb-name linuxnva-slb-ext -o table**

BackendPort FrontendPort LoadDistribution Name Protocol

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

22 1022 Default ssh Tcp

***Note****: some column have been removed from the previous output for simplicity*

**az network lb rule delete --lb-name linuxnva-slb-ext -n ssh**

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

{

"backendAddressPool": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-ext/backendAddressPools/linuxnva-slbBackend-ext",

"resourceGroup": "vnetTest"

},

"backendPort": 22,

"enableFloatingIp": true,

"etag": "W/\"...\"",

"frontendIpConfiguration": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-ext/frontendIPConfigurations/myFrontendConfig",

"resourceGroup": "vnetTest"

},

"frontendPort": 22,

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-ext/loadBalancingRules/sshRule",

"idleTimeoutInMinutes": 4,

"loadDistribution": "Default",

"name": "sshRule",

"probe": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-ext/probes/myProbe",

"resourceGroup": "vnetTest"

},

"protocol": "Tcp",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

1. The first problem we need to solve is routing at the NVA. VMs get a static route for 168.63.129.16 pointing to their primary interface, in this case, eth0. Verify that that is the case, since the disabling/enabling of eth0 in a previous lab might have deleted that route.

lab-user@linuxnva-2:~$ **route -n**

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface

0.0.0.0 10.4.3.1 0.0.0.0 UG 0 0 0 eth1

0.0.0.0 10.4.2.1 0.0.0.0 UG 100 0 0 eth0

10.0.0.0 10.4.2.1 255.248.0.0 UG 0 0 0 eth0

10.4.2.0 0.0.0.0 255.255.255.0 U 100 0 0 eth0

10.4.3.0 0.0.0.0 255.255.255.0 U 10 0 0 eth1

**168.63.129.16 10.4.2.1 255.255.255.255 UGH 100 0 0 eth0**

169.254.169.254 10.4.2.1 255.255.255.255 UGH 100 0 0 eth0

If the route to 168.63.129.16 is not there, you can add it easily:

**sudo route add -host 168.63.129.16 gw 10.4.2.1 dev eth0**

By the way, if the route to 168.63.129.16 disappeared, you probably need to add another static route telling the firewall where to find the 10.0.0.0/8 networks (where all our Vnets are):

**sudo route add -net 10.0.0.0/8 gw 10.4.2.1 dev eth0**

Now we are sure that the NVA has 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 external load balancer on eth1? Since the static route is pointing down to eth0, the NVA would send the answer there. But this is not going to work, because the answer needs to be sent over the same interface.

1. You can verify this behavior connecting to one of the NVA VMs and capturing traffic on both ports (filtering it to the TCP ports where the probes are configured). In this case we are connecting to linuxnva-1, and verifying the internal interface and TCP port (eth0, TCP port 1138):

lab-user@linuxnva-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

22:50:49.277214 IP 168.63.129.16.50717 > 10.4.2.101.1138: Flags [**S**EW], seq 2412262844, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

22:50:49.277239 IP 10.4.2.101.1138 > 168.63.129.16.50717: Flags [**S.**], seq 3801638535, ack 2412262845, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0

22:50:49.277501 IP 168.63.129.16.50717 > 10.4.2.101.1138: Flags [**.**], ack 1, win 513, length 0

22:50:49.589219 IP 168.63.129.16.50288 > 10.4.2.101.1138: Flags [F.], seq 0, ack 1, win 64240, length 0

22:50:50.198577 IP 168.63.129.16.50288 > 10.4.2.101.1138: Flags [F.], seq 0, ack 1, win 64240, length 0

You can see that the 3-way handshake ("S" is SYN, "." is ACK, so you see the "SYN"/"SYN-ACK"/"ACK" sequence marked in red above) completes successfully on the internal interface, as the TCP flags of the capture indicate. But if we have a look at the external interface, things look different there:

lab-user@nva-1:~$ **sudo tcpdump -i eth1 port 1139**

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

listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes

22:54:15.584402 IP 168.63.129.16.56583 > 10.4.3.101.1139: Flags [**SEW**], seq 314423445, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

22:54:18.584140 IP 168.63.129.16.56583 > 10.4.3.101.1139: Flags [**SEW**], seq 314423445, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

22:54:24.584127 IP 168.63.129.16.56583 > 10.4.3.101.1139: Flags [**S**], seq 314423445, win 8192, options [mss 1440,nop,nop,sackOK], length 0

22:54:30.587651 IP 168.63.129.16.56995 > 10.4.3.101.1139: Flags [**SEW**], seq 2980654025, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

22:54:33.587444 IP 168.63.129.16.56995 > 10.4.3.101.1139: Flags [**SEW**], seq 2980654025, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

As you can see in the TCP flags, the 3-way handshake never completes (only SYN, the SYN-ACK never comes back): the Load Balancer keeps sending packets with the SYN flag on, without getting a single ACK back.

1. The problem here is routing. The NVA is getting the health checks from the load balancer always from the same IP address, but it needs to seem them back on different interfaces. How to route to the same IP address depending on where the packet is coming from?  
   In order to fix routing, we are going to implement policy based routing in both NVAs. The first step is creating a custom route table at the Linux level, by modifying the file rt\_tables and adding the line "201 slbext":

lab-user@linuxnva-1:~$ **sudo vi /etc/iproute2/rt\_tables**

The file now should look like something like this:

lab-user@linuxnva-1:~$ **more /etc/iproute2/rt\_tables**

#

# reserved values

#

255 local

254 main

253 default

0 unspec

#

# local

#

#1 inr.ruhep

201 slbext

1. Now we add a rule that will tell Linux when to use that routing table. That is, when it needs to send the answer to the LB probe from the external interface (10.4.3.101 in the case of linuxnva-1, 10.4.3.102 for nva-2).

lab-user@linuxnva-1:~$ **sudo ip rule add from 10.4.3.101 to 168.63.129.16 lookup slbext**

or

lab-user@linuxnva-2:~$ **sudo ip rule add from 10.4.3.102 to 168.63.129.16 lookup slbext**

1. And finally, we populate the custom routing table with a single route, pointing up to eth1:

lab-user@linuxnva-1:~$ **sudo ip route add 168.63.129.16 via 10.4.3.1 dev eth1 table slbext**

1. Verify that the commands took effect, and that the TCP 3-way handshake is now correctly established on eth1:

lab-user@linuxnva-1:~$ **ip rule list**

0: from all lookup local

**32765: from 10.4.3.101 to 168.63.129.16 lookup slbext**

32766: from all lookup main

32767: from all lookup default

lab-user@nva-1:~$ **ip route show table slbext**

168.63.129.16 via 10.4.3.1 dev eth1

lab-user@linuxnva-1:~$ **sudo tcpdump -i eth1 port 1139**

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

listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes

23:11:45.774301 IP 168.63.129.16.54073 > 10.4.3.101.1139: Flags [**SEW**], seq 3604073494, win 8192, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0

23:11:45.774333 IP 10.4.3.101.1139 > 168.63.129.16.54073: Flags [**S.**], seq 2611260758, ack 3604073495, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0

23:11:45.774488 IP 168.63.129.16.54073 > 10.4.3.101.1139: Flags [**.**], ack 1, win 513, length 0

23:11:46.086572 IP 168.63.129.16.53650 > 10.4.3.101.1139: Flags [F.], seq 0, ack 1, win 64240, length 0

23:11:46.695967 IP 168.63.129.16.53650 > 10.4.3.101.1139: Flags [F.], seq 0, ack 1, win 64240, length 0

1. Don’t forget to run the previous procedure (from step 6) in linuxnva-2 too (or in linuxnva-1, if you did them on linuxnva-2).
2. One missing piece is the NAT configuration at both firewalls: traffic will arrive from the external load balancer addressed to the VIP assigned to the load balancer, since we configured Direct Server Return (also known as floating IP). Now we need to NAT that address to the VM where we want to send this traffic to, in both firewalls:

lab-user@linuxnva-1:~$ **sudo iptables -t nat -A PREROUTING -p tcp -d 1.2.3.4 --dport 22 -j DNAT --to-destination 10.3.1.4:22**

and

lab-user@linuxnva-2:~$ **sudo iptables -t nat -A PREROUTING -p tcp -d 1.2.3.4 --dport 22 -j DNAT --to-destination 10.3.1.4:22**

***Note****: do not forget to replace here the bogus IP address "1.2.3.4" with the actual public IP address assigned in your environment to the public IP address "linuxnva-slbPip-ext". You can get the list of public IP addresses in your environment with the command "az network public-ip list -o table".*

lab-user@linuxnva-2:~$ **sudo iptables -vL -t nat**

Chain PREROUTING (policy ACCEPT 114 packets, 6118 bytes)

pkts bytes target prot opt in out source destination

**0 0 DNAT tcp -- any any anywhere 1.2.3.4 tcp dpt:ssh to:10.3.1.4:22**

Chain INPUT (policy ACCEPT 39 packets, 1967 bytes)

pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 59 packets, 3831 bytes)

pkts bytes target prot opt in out source destination

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)

pkts bytes target prot opt in out source destination

1193 81052 MASQUERADE all -- any eth0 anywhere anywhere

1574 95368 MASQUERADE all -- any eth1 anywhere anywhere

1. Now we should be able to connect to the VM from the public Internet. Open another Putty window, and try to connect to the public IP address of the load balancer (1.2.3.4 in our example).

***Note****: please make sure to replace* ***3.3.3.3*** *with the actual public IP address of your VM*

**What we have learnt**

For traffic incoming from the public Internet, you need to add an extra level of external load balancer. Having multiple load balancers managing traffic to the same set of firewalls can be problematic, specially if the firewalls have multiple interfaces, since health check probes could be inadvertently sent the wrong way, which would break the setup.

One possibility to avoid such complexity (that we did not explore in this particular lab) would be using single-NIC firewalls, also known as "firewall-on-a-stick", as opposed to having separate external and internal interfaces.

# Lab 7: Advanced HTTP-based probes (OPTIONAL)

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.

1. We need to change the probe from TCP-based to HTTP-based, for example, in the internal LB (you can do it in the external one too). From your Windows command prompt:

**az network lb probe update -n myProbe --lb-name linuxnva-slb-int --protocol Http --path "/" --port 80**

{

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/probes/myProbe",

"intervalInSeconds": 15,

"loadBalancingRules": [

{

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/loadBalancingRules/sshRule",

"resourceGroup": "vnetTest"

},

{

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/loadBalancers/linuxnva-slb-int/loadBalancingRules/httpRule",

"resourceGroup": "vnetTest"

}

],

"name": "myProbe",

"numberOfProbes": 2,

"port": 80,

"protocol": "Http",

"provisioningState": "Succeeded",

"requestPath": "/",

"resourceGroup": "vnetTest"

}

1. Verify the content that NVAs return to the probe. You can query this from any VM, for example, from your Putty window connected to Vnet1-vm1:

lab-user@myVnet1-vm1:~$ **curl -i 10.4.2.101**

HTTP/1.1 **200** OK

Date: Tue, 28 Mar 2017 00:08:47 GMT

Server: Apache/2.4.18 (Ubuntu)

Vary: Accept-Encoding

Content-Length: 236

Content-Type: text/html; charset=UTF-8

<html>

<header>

<title>Network Virtual Appliance</title>

</header>

<body>

<h1>

Welcome to the Open Source Azure Networking Lab

</h1>

<br>

All target hosts seem to be reachable

</body>

</html>

1. Verify the logic of the "/var/www/html/index.php" file in each NVA VM, from the putty window connected to any of the NVAs. As you can see, it returns the HTTP code 200 only if a list of IP addresses or DNS names is reachable. You can query this from any VM, for example, from your Putty window connected to Vnet1-vm1:

lab-user@linuxnva-1:~$ **more /var/www/html/index.php**

<html>

<header>

<title>Network Virtual Appliance</title>

</header>

<body>

<h1>

Welcome to the Open Source Azure Networking Lab

</h1>

<br>

<?php

$hosts = array **("bing.com", "10.1.1.4")**;

$allReachable = true;

foreach ($hosts as $host) {

$result = exec ("ping -c 1 -W 1 " . $host . " 2>&1 | grep received");

$pos = strpos ($result, "1 received");

if ($pos === false) {

$allReachable = false;

break;

}

}

if ($allReachable === false) {

// Ping did not work

**http\_response\_code (299)**;

print ("The target hosts do not seem to be all reachable (" . $host . ")\n");

} else {

// Ping did work

**http\_response\_code (200);**

print ("All target hosts seem to be reachable\n");

}

?>

</body>

</html>

lab-user@nva-1:~$

Now the probe for the internal load balancer will fail even if the internal interface is up, but for some reason the NVA cannot connect to the Internet, therefore enhancing the overall reliability of the solution.

**What we have learnt**

Advanced HTTP probes can be used to verify additional information, so that firewalls are taken out of rotation whenever complex failure scenarios occur, such as the failure of an interface other than the one the probe was sent to, or a certain process not being running in the system (to detect if the firewall daemon is still running).

**PART 3**

**VPN to external site**

# Lab 8: Spoke-to-Spoke communication over vPN gateway (OPTIONAL)

**NOTE**: provisioning of the VPN gateways will take up to 45 minutes to complete

In this lab we will simulate the connection to an on-premises data center, that in our case will be simulated by vnet5. We will create a BGP-based VPN connection between our Hub vnet in Azure (vnet4), and the on-premises DC (simulated with vnet5).

For this lab you will need to have set up virtual network gateways in vnets 4 and 5. You can verify whether gateways exist in those vnets with these commands:

1. No gateway exists in either Vnet, you can create them with these commands. It is recommended to run these commands in separate terminals so that they run in parallel, since they take a long time to complete (up to 45 minutes):

**az network vnet-gateway create --name vnet4Gw --vnet myVnet4 --public-ip-addresses vnet4gwPip --sku standard --asn 65504**

{

"vnetGateway": {

"activeActive": false,

"bgpSettings": {

"asn": **65504**,

"bgpPeeringAddress": "**10.4.0.254**",

"peerWeight": 0

},

"enableBgp": **true**,

"etag": "W/\"...\"",

"gatewayDefaultSite": null,

"gatewayType": "**Vpn**",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet4Gw",

"ipConfigurations": [

{

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet4Gw/ipConfigurations/vnetGatewayConfig0",

"name": "vnetGatewayConfig0",

"privateIpAllocationMethod": "Dynamic",

"provisioningState": "Succeeded",

"publicIpAddress": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/publicIPAddresses/vnet4gwPip",

"resourceGroup": "vnetTest"

},

"resourceGroup": "vnetTest",

"subnet": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet4/subnets/GatewaySubnet",

"resourceGroup": "vnetTest"

}

}

],

"location": "westeurope",

"name": "vnet4Gw",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"resourceGuid": "...",

"sku": {

"capacity": 2,

"name": "**Standard**",

"tier": "**Standard**"

},

"tags": null,

"type": "Microsoft.Network/virtualNetworkGateways",

"vpnClientConfiguration": null,

"vpnType": "RouteBased"

}

}

**az network vnet-gateway create --name vnet5Gw --vnet myVnet5 --public-ip-addresses vnet5gwPip --sku standard --asn 65505**

{

"vnetGateway": {

"activeActive": false,

"bgpSettings": {

"asn": **65505**,

"bgpPeeringAddress": "**10.5.0.254**",

"peerWeight": 0

},

"enableBgp": **true**,

"etag": "W/\"...\"",

"gatewayDefaultSite": null,

"gatewayType": "**Vpn**",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet5Gw",

"ipConfigurations": [

{

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet5Gw/ipConfigurations/vnetGatewayConfig0",

"name": "vnetGatewayConfig0",

"privateIpAllocationMethod": "Dynamic",

"provisioningState": "Succeeded",

"publicIpAddress": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/publicIPAddresses/vnet5gwPip",

"resourceGroup": "vnetTest"

},

"resourceGroup": "vnetTest",

"subnet": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet5/subnets/GatewaySubnet",

"resourceGroup": "vnetTest"

}

}

],

"location": "westeurope",

"name": "vnet5Gw",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"resourceGuid": "...",

"sku": {

"capacity": 2,

"name": "**Standard**",

"tier": "**Standard**"

},

"tags": null,

"type": "Microsoft.Network/virtualNetworkGateways",

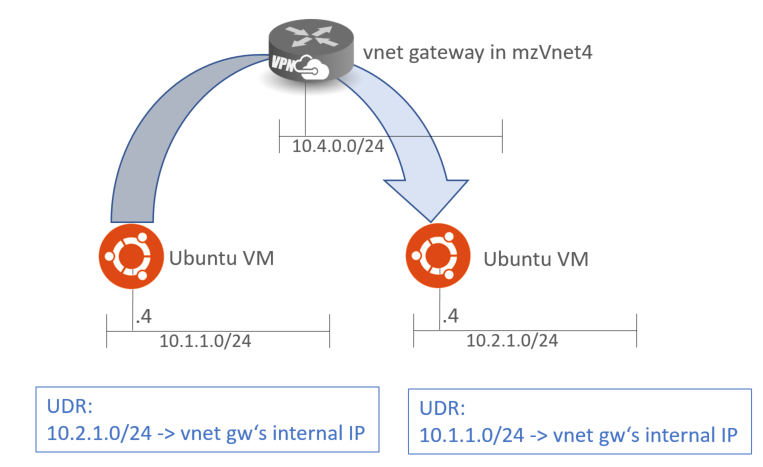
"vpnClientConfiguration": null,

"vpnType": "RouteBased"

}

}

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

1. We need to replace the route we installed in Vnet1-Subnet1 and Vnet2-Subnet1 pointing to Vnet4’s NVA, with another one pointing to the VPN gateway. You will not be able to find out on the GUI or the CLI the IP address assigned to the VPN gateway, but you can guess it. Since the first 3 addresses in every subnet are reserved for the vnet router, the gateway should have got the IP address **10.4.0.4** (remember that we allocated the prefix 10.4.0.0 to the Gateway Subnet in myVnet4). You can verify it pinging this IP address from any VM. For example, from myVnet1-vm1:

lab-user@myVnet1-vm1:~$ **ping 10.4.0.4**

PING 10.4.0.4 (10.4.0.4) 56(84) bytes of data.

64 bytes from 10.4.0.4: icmp\_seq=1 ttl=128 time=2.27 ms

64 bytes from 10.4.0.4: icmp\_seq=2 ttl=128 time=0.794 ms

64 bytes from 10.4.0.4: icmp\_seq=3 ttl=128 time=1.16 ms

64 bytes from 10.4.0.4: icmp\_seq=4 ttl=128 time=0.937 ms

^C

--- 10.4.0.4 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3001ms

rtt min/avg/max/mdev = 0.794/1.291/2.271/0.582 ms

lab-user@myVnet1-vm1:~$

1. Modify the routes in vnets 1 and 2 with these commands, back in your Windows command prompt:

**az network route-table route update --next-hop-ip-address 10.4.0.4 --route-table-name vnet1-subnet1 -n vnet2**

{

"addressPrefix": "10.2.0.0/16",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1/routes/vnet2",

"name": "vnet2",

"nextHopIpAddress": "10.4.0.4",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

**az network route-table route update --next-hop-ip-address 10.4.0.4 --route-table-name vnet2-subnet1 -n vnet1**

{

"addressPrefix": "10.1.0.0/16",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet2-subnet1/routes/vnet1",

"name": "vnet1",

"nextHopIpAddress": "10.4.0.4",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

1. Now you can verify what the route tables look like in an interface from a VM in the vnets and the NICs, and how it has been programmed in the NICs. We will do it with myVnet1, optionally you can verify myVnet2 and a NIC in myVnet2 too:

**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

10.1.1.0/24 vnet1-subnet1 10.4.2.100 VirtualAppliance Succeeded

10.3.1.0/24 vnet3subnet1 10.4.2.100 VirtualAppliance Succeeded

***Note****: some columns have been removed from the output above for clarity purposes.*

**az network nic show-effective-route-table -n myVnet1-vm1-nic**

...

{

"addressPrefix": [

"10.2.0.0/16"

],

"name": "vnet2",

"nextHopIpAddress": [

"10.4.0.4"

],

"nextHopType": "VirtualAppliance",

"source": "User",

"state": "Active"

}

***Note****: the command above will take some seconds to execute, since it needs to access to low-level routing tables programmed in the VM's NIC*

**Optionally**: find out and execute the commands in order to change the next hop for the other routes in the routing table for myVnet1

1. And now VM1 should be able to reach VM2, this time not over the NVA, but over the VPN gateway. Note that ping now is working, since the VPN gateway is not filtering out ICMP as the NVA did:

lab-user@myVnet1-vm1:~$ **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

**What we have learnt**

VPN gateways can also be used for spoke-to-spoke communications, instead of NVAs. You need to "guess" the IP address that a VPN gateway will receive, and you can use that IP address in UDRs as next hop.

# Lab 9: VPN connection to the Hub Vnet (OPTIONAL)

1. Make sure that the VPN gateways have different Autonomous System Numbers (ASN) configured. You can check the ASN with this command, back in your Windows command prompt:

**az network vnet-gateway show -n vnet4gw | findstr asn**

"asn": **65504**,

**az network vnet-gateway show -n vnet5gw | findstr asn**

"asn": **65505**,

***Note****: if you are running this step in a Linux machine, make sure to replace "findstr" with "grep"*

1. Change peerings to use the gateways we created in the previous lab. The "useRemoteGateways" property of the network peering will allow the vnet to use any VPN or ExpressRoute gateway in the destination vnet. Note that this option cannot be set if the destination vnet does not have any VPN or ExpressRoute gateway configured (which is the reason why the initial ARM template did not configure it, since we did not have our VPN gateways yet).

**az network vnet peering update --vnet-name myVnet1 --name LinkTomyVnet4 --set useRemoteGateways=true**

{

"allowForwardedTraffic": **true**,

"allowGatewayTransit": false,

"allowVirtualNetworkAccess": true,

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet1/virtualNetworkPeerings/LinkTomyVnet4",

"name": "LinkTomyVnet4",

"peeringState": "Connected",

"provisioningState": "Succeeded",

"remoteVirtualNetwork": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet4",

"resourceGroup": "vnetTest"

},

"resourceGroup": "vnetTest",

"useRemoteGateways": **true**

}

***Note****: should you receive an error message like "An error occurred", just retry the command.*

**az network vnet peering update --vnet-name myVnet2 --name LinkTomyVnet4 --set useRemoteGateways=true**

{

"allowForwardedTraffic": **true**,

"allowGatewayTransit": false,

"allowVirtualNetworkAccess": true,

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet2/virtualNetworkPeerings/LinkTomyVnet4",

"name": "LinkTomyVnet4",

"peeringState": "Connected",

"provisioningState": "Succeeded",

"remoteVirtualNetwork": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet4",

"resourceGroup": "vnetTest"

},

"resourceGroup": "vnetTest",

"useRemoteGateways": **true**

}

***Note****: should you receive an error message like "An error occurred", just retry the command.*

**az network vnet peering update --vnet-name myVnet3 --name LinkTomyVnet4 --set useRemoteGateways=true**

{

"allowForwardedTraffic": **true**,

"allowGatewayTransit": false,

"allowVirtualNetworkAccess": true,

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet3/virtualNetworkPeerings/LinkTomyVnet4",

"name": "LinkTomyVnet4",

"peeringState": "Connected",

"provisioningState": "Succeeded",

"remoteVirtualNetwork": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet4",

"resourceGroup": "vnetTest"

},

"resourceGroup": "vnetTest",

"useRemoteGateways": **true**

}

***Note****: should you receive an error message like "An error occurred", just retry the command.*

1. Now 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**

{

"connectionStatus": "Unknown",

"connectionType": "Vnet2Vnet",

"egressBytesTransferred": 0,

"enableBgp": true,

"ingressBytesTransferred": 0,

"provisioningState": "Succeeded",

"resourceGuid": "...",

"routingWeight": 10,

"sharedKey": "Microsoft123",

"virtualNetworkGateway1": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet4Gw",

"resourceGroup": "vnetTest"

},

"virtualNetworkGateway2": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet5gw",

"resourceGroup": "vnetTest"

}

}

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

{

"connectionStatus": "Unknown",

"connectionType": "Vnet2Vnet",

"egressBytesTransferred": 0,

"enableBgp": true,

"ingressBytesTransferred": 0,

"provisioningState": "Succeeded",

"resourceGuid": "...",

"routingWeight": 10,

"sharedKey": "Microsoft123",

"virtualNetworkGateway1": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet5Gw",

"resourceGroup": "vnetTest"

},

"virtualNetworkGateway2": {

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet4gw",

"resourceGroup": "vnetTest"

}

}

Once you have provisioned the connections you can list them with this command.:

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

ConnectionType EnableBgp Name ProvisioningState RoutingWeight

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

Vnet2Vnet True 4to5 Succeeded 10

Vnet2Vnet True 5to4 Succeeded 10

1. Get the connection status of the tunnels, and wait until they are connected:

**az network vpn-connection show --name 4to5 | findstr connectionStatus**

"connectionStatus": "**Connecting**",

Wait some seconds, and reissue the command until you get a "Connected" status, as the following ouputs show:

**az network vpn-connection show --name 4to5 | findstr connectionStatus**

"connectionStatus": "**Connected**",

**az network vpn-connection show --name 5to4 | findstr connectionStatus**

"connectionStatus": "**Connected**",

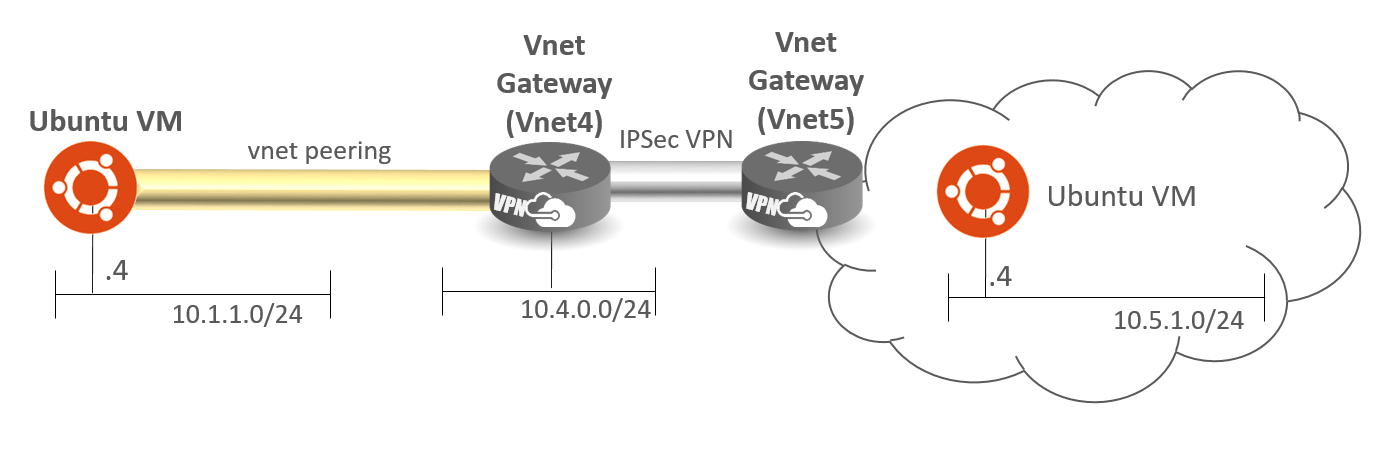
1. If you now try to reach a VM in myVnet5 from any of the VMs in the other Vnets, it should work without any further configuration, following the topology found in the figure below. For example, from our myVnet1-vm1 we will ping 10.5.1.4, which should be the private IP address from myVnet5-vm1:

lab-user@myVnet1-vm1:~$ **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 10**: 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 **AllowGatewayTransit** (in the hub). Back in your Windows command prompt, you can issue this command to check the state and configuration of your vnet peerings:

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

AllowForwardedTraffic Name PeeringState UseRemoteGateways

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

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

***Note****: some of the columns have been removed from the output above for clarity purposes*

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

AllowGatewayTransit Name PeeringState

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

**True** LinkTomyVnet2 Connected

**True**  LinkTomyVnet1 Connected

**True**  LinkTomyVnet3 Connected

***Note****: some of the columns have been removed from the output above for clarity purposes*

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). For example, for our VM1 in Vnet1 look for the route to 10.5.0.0/16, and you will see that the next hop is a public IP address, of the type "VirtualNetworkGateway". This is the public IP address that the VNG uses to establish the VPN tunnel across the public Internet. Obviously, the public IP address you get will be different to the one in this example:

**az network nic show-effective-route-table -n myVnet1-vm1-nic**

...

{

"addressPrefix": [

"**10.5.0.0/16**"

],

"name": null,

"nextHopIpAddress": [

"**13.81.113.28**"

],

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

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

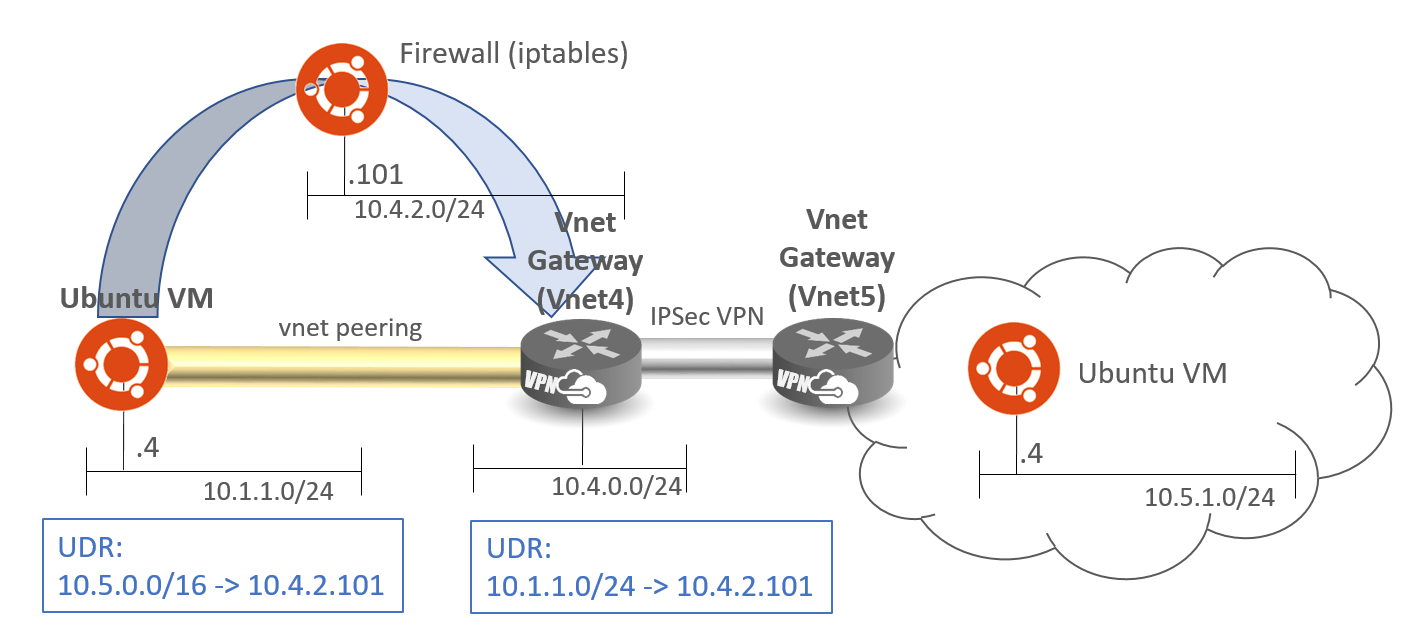
"state": "Active"

},

...

***Note****: the previous command will take some seconds to execute, since it needs to access the routes programmed in the NIC and that takes some time.*

However, you might want to push this traffic through the Network Virtual Appliances too. For example, if you wish to firewall the traffic that leaves your hub and spoke environment. The process that we have seen in previous labs with UDR manipulation is valid for the GatewaySubnet of Vnet4 as well (where the hub VPN gateway is located), as the following figure depicts:



**Figure 11**. 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/\"...\"",

"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/\"...\"",

"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

***Note****: for simplicity we use as next-hop the individual IP address of one of our firewalls, we could have use the load balancer as next hop too for NVA HA.*

And now we associate the new routing table to the gateway subnet:

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

{

"addressPrefix": "10.4.0.0/24",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworks/myVnet4/subnets/GatewaySubnet",

"ipConfigurations": [

{

"etag": null,

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/virtualNetworkGateways/vnet4Gw/ipConfigurations/vnetGatewayConfig0",

"name": null,

"privateIpAddress": null,

"privateIpAllocationMethod": null,

"provisioningState": null,

"publicIpAddress": null,

"resourceGroup": "vnetTest",

"subnet": null

}

],

"name": "GatewaySubnet",

"networkSecurityGroup": null,

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest",

"resourceNavigationLinks": null,

"routeTable": {

"etag": null,

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

"location": null,

"name": null,

"provisioningState": null,

"resourceGroup": "vnetTest",

"routes": null,

"subnets": null,

"tags": null,

"type": null

}

}

1. We need to add an additional route to the spoke vnets, to let them know that they can reach the remote site (in our lab simulated by myVnet5, with an IP prefix of 10.5.0.0/16) over the NVA. Here we do it for myVnet1, you can do it for the other spoke Vnets too optionally (myVnet2 and myVnet3):

**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**

{

"addressPrefix": "10.5.0.0/16",

"etag": "W/\"...\"",

"id": "/subscriptions/.../resourceGroups/vnetTest/providers/Microsoft.Network/routeTables/vnet1-subnet1/routes/vnet5",

"name": "vnet5",

"nextHopIpAddress": "10.4.2.101",

"nextHopType": "VirtualAppliance",

"provisioningState": "Succeeded",

"resourceGroup": "vnetTest"

}

1. Now you can verify that VMs in myVnet1Subnet1 can still connect over SSH to VMs in myVnet5, but not any more over ICMP (since we have a rule for dropping ICMP traffic in the NVA):

lab-user@myVnet1-vm1:~$ **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@myVnet1-vm1:~$ **ssh 10.5.1.4**

...

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

**What we have learnt**

Vnet peerings allow for sharing VPN gateways in the hub to provide connectivity to the spokes through the peering option "Use Remote Gateways".

You can use NVAs to secure the traffic going between the local Vnets and the remote site (at the other side of the Site-To-Site tunnel), manipulating the routing in the subnet gateway with UDRs. This configuration works as well in Hub-And-Spoke Vnet configurations.

# End the lab

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

**az group delete --name vnetTest**

# Conclusion

I hope you have had fun running through this lab, and that you learnt something that you did not know before. We ran through multiple Azure networking topics like IPSec VPN, vnet peering, hub & spoke vnet topologies and advanced NVA integration, but we covered as well other non-Azure topics such as Linux custom routing or advanced probes programming with PHP.

If you have any suggestion to improve this lab, please open an issue in Github in this repository.

## **References**

Useful links:

* Azure network documentation: <https://docs.microsoft.com/en-us/azure/#pivot=services&panel=network>
* Hub and Spoke network topology in Azure: <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke>
* Olivier Martin blog's on Azure networking:
  + Part 1: <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke>
  + Part 2: <https://azure.microsoft.com/en-us/blog/networking-to-and-within-the-azure-cloud-part-2/>
  + Part 3: <https://azure.microsoft.com/en-us/blog/networking-to-and-within-the-azure-cloud-part-3/>
* Vnet documentation: <https://docs.microsoft.com/en-us/azure/virtual-network/>
* Load Balancer documentation: <https://docs.microsoft.com/en-us/azure/load-balancer/>
* VPN Gateway documentation: <https://docs.microsoft.com/en-us/azure/vpn-gateway/>