Hyper-V VMs on Azure with Powershell

A "dive" 1" below the water line!

http://github.com/coderUT/cltpsug-July2019

GitHub Repository

- I've created a Repository on GitHub for this Presentation
  - http://github.com/coderUT/cltpsug-July2019

- It contains:
  - A PDF version of this slide deck

  - A set of "usable" Powershell scripts if you want to try to create the same deployments, we talk about tonight
    - Any slide with the header "**Deployment**", the script on that slide is in GitHub

    - There is enough foundation here that if you wanted to re-factor these scripts into a Powershell Module……you can!

    - A side note, you will need to change the names of the Azure Storage Accounts "**cltpsugbootdiags**" and "**cltpsugvmdiags**" in the appropriate scripts
      - Storage Account Names must be globally unique across Azure as they are part of a URI

      - Utilize the Powershell cmdlet to test a name:
        - **Get-AzStorageAccountNameAvailability –Name <val>**

  - A "QandA.txt" file: Any questions I can't answer tonight, I'll try to capture, research and document the results in this file over the next few weeks

Goals

- Hyper-V VMs and Azure is an enormously broad topic for a single 60-90 min presentation
  - I struggled a lot with deciding the appropriate level of depth
  - I tried to make trade-off's between being too simplistic vs getting so detailed no-one understands

- If your new to Azure but an expert with Hyper-V and Windows, I want you leave tonight with:
  - A way to relate your existing knowledge to Hyper-V and Azure

  - A good "jumping off point" to learn more:
    - You can take the VM configuration we create tonight and "layer" more complexity on top
    - Azure can be difficult to learn because is interconnected, dependent and has "default configurations" that in many cases are not documented well-enough (*in my humble opinion*)

  - **A suggestion while learning**: only use the Portal for "verification"
    - Try to do as many tasks as possible with the command line (Powershell, Azure CLI, etc…)
    - The Portal abstracts so much that its hard to learn
    - This is more work on your part, but you will learn more quickly and be more efficient in the long-run

- If you're an expert in Azure and Hyper-V, that's great and you can help me learn instead!
  - If I make any mistakes, please chime in and correct them
  - If there are gaps in my knowledge or I "gloss over" a significant detail, please let me know
  - The goal is to learn as a group, so I strongly encourage participation

“The Storyline”

- Provision 2 Hyper-V VMs (Windows and Linux) in Azure using only Powershell
  - Azure Resource Manager (ARM) Model Overview
    - Prepare our Azure Subscription

  - Azure Resource “analogs” for each major component (VM Settings, Disks, and Networking) of a Hyper-VM
    - A brief survey of
      - The different CPU/memory combinations available
      - The different Disk Mediums available
      - How CPU/memory + Disk Medium affects VM cost given a performance specification
        - The short-answer…you will need to capture statistics on your Workload to pick the right VM/disk specs at the best cost
    - Examine Powershell code snippets for configuration of each Resource
    - Examine the Powershell scripts to effect deployment of our VMs

  - Once our VMs are deployed, we’ll configure diagnostics so we can monitor our VMs using Extensions
    - Azure is constantly evolving but I’m going to limit this to Boot Diagnostics and “classic” Guest OS Diagnostics
    - You can easily layer some of the other diagnostics functionality (e.g., Azure Monitor Logs/Log Analytics) on top of the configurations we create tonight

  - We’ll configure access to our VMs via VPN (we will not make them public endpoints tonight)
    - Its not a lot more work to secure your endpoints once you understand the basics

“The Storyline” (cont’d)

- Cloud Resource Administration and ARM
  - Look at 4 elements of the ARM model and Powershell Snippets
    - Allowing us to exercise control over what Users are doing in our Azure subscriptions

  - **Role Assignments**
    - What Resources can users Access in the Subscription and what Operations can they effect
    - **Example**: Who can restart a VM?

  - **Policy Assignments**
    - Control What Resource Types can be deployed in our Subscription AND where
    - **Example**: You Enterprise Agreement that grants significant discounts on specific VM SKUs in the East US 2 region only
      - How do you make it so your users are forced to deploy only those SKUs and only in East US 2?

  - **Resource Tagging**
    - In order to assign costing to Users/Cost Centers, we need a way to track the resources deployed so we can reconcile them with the bill
      - Now, you can of course just assign separate Cost Centers their own Subscription and be done

  - **Resource Locks**
    - When a Resource is deleted, it is gone….how can we prevent “fat finger’s” in our Subscription?

3 Elements of
A Hyper-V VM

VM Settings

Disks

Networking

Hyper-V VM: Windows Server vs Azure

Windows Server Host

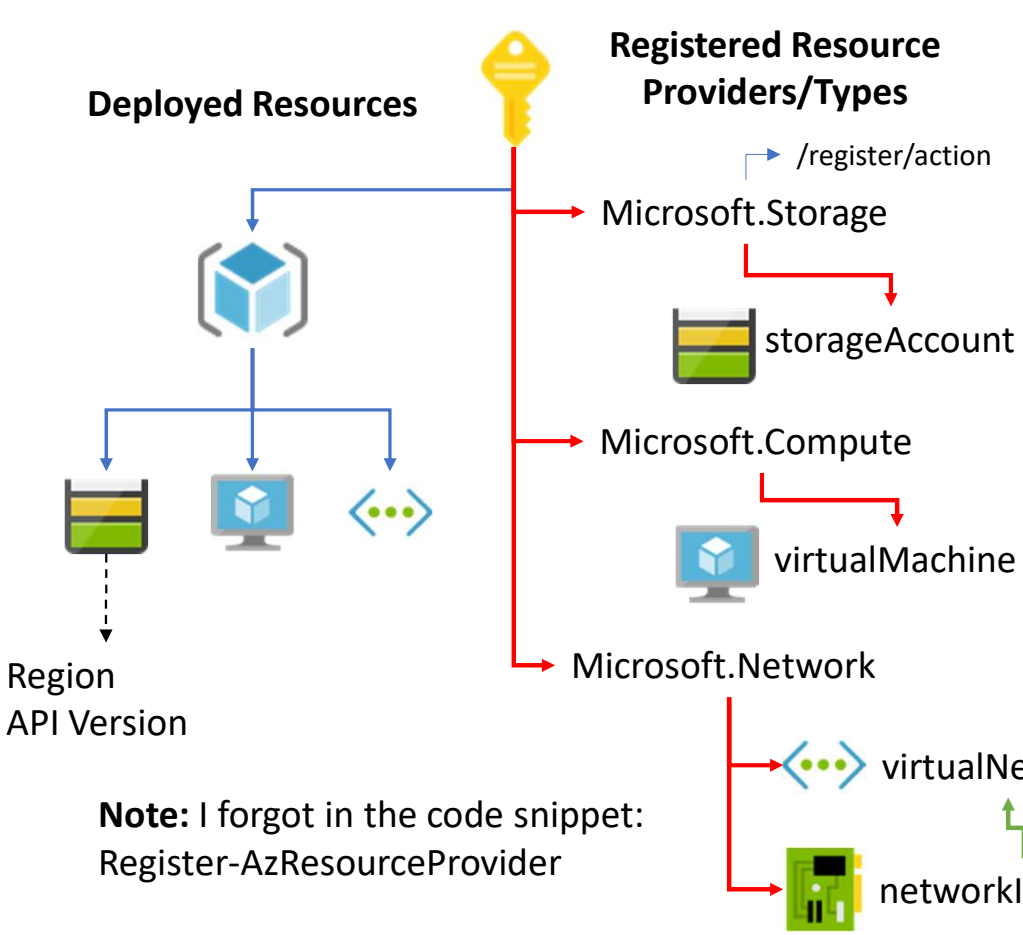
- **VM Settings (vCPUs and Affinity, Memory)**
  - Gen1 or 2
  - Configuration file on Host (.VMCX)
- **Disks (OS Disk required)**
  - VHD (2TB) or VHDx (64TB), Fixed-size or Dynamic
  - Stored on Physical Storage (Host or Network Attached [SAN LUN or SMB])
- **Networking (optional)**
  - Guest OS manages IP configuration
  - Virtual Switch
  - Virtual Network Adapter

Azure Host

- **VM Settings (vCPUs, Memory)**
  - Gen1 only (Gen2 is in preview)
  - Stand-alone Azure Resource Type
- **Disks (OS Disk required)**
  - VHD, Fixed-size only, size limit 4TB (as data disk) (I'm not sure how this is implemented under the covers)
  - Stored in Azure Managed Disk Resource
- **Networking (required)**
  - Azure manages IP configuration
  - Virtual Network (Stand-alone Azure Resource Type)
  - Network Adapter (Stand-alone Azure Resource Type)

```
#Basic Windows Server Hyper-V Cmdlets
New-VM
Start-VM
Stop-VM

Get-VMHardDiskDrive
Add-VMHardDiskDrive
Remove-VMHardDiskDrive

Get-VMNetworkAdapter
Add-VMNetworkAdapter
Remove-VMNetworkAdapter

#Azure Hyper-V Mgmt Analogs
New-AzVM
New-AzVMConfig
Start-AzVM
Stop-AzVM

Get-AzDisk
Set-AzVMOSDisk
Set-AzVMDataDisk

Get-AzNetworkInterface
Add-AzVMNetworkInterface
Remove-AzVMNetworkInterface
```

Azure Resource Types and Resources (ARM Model)



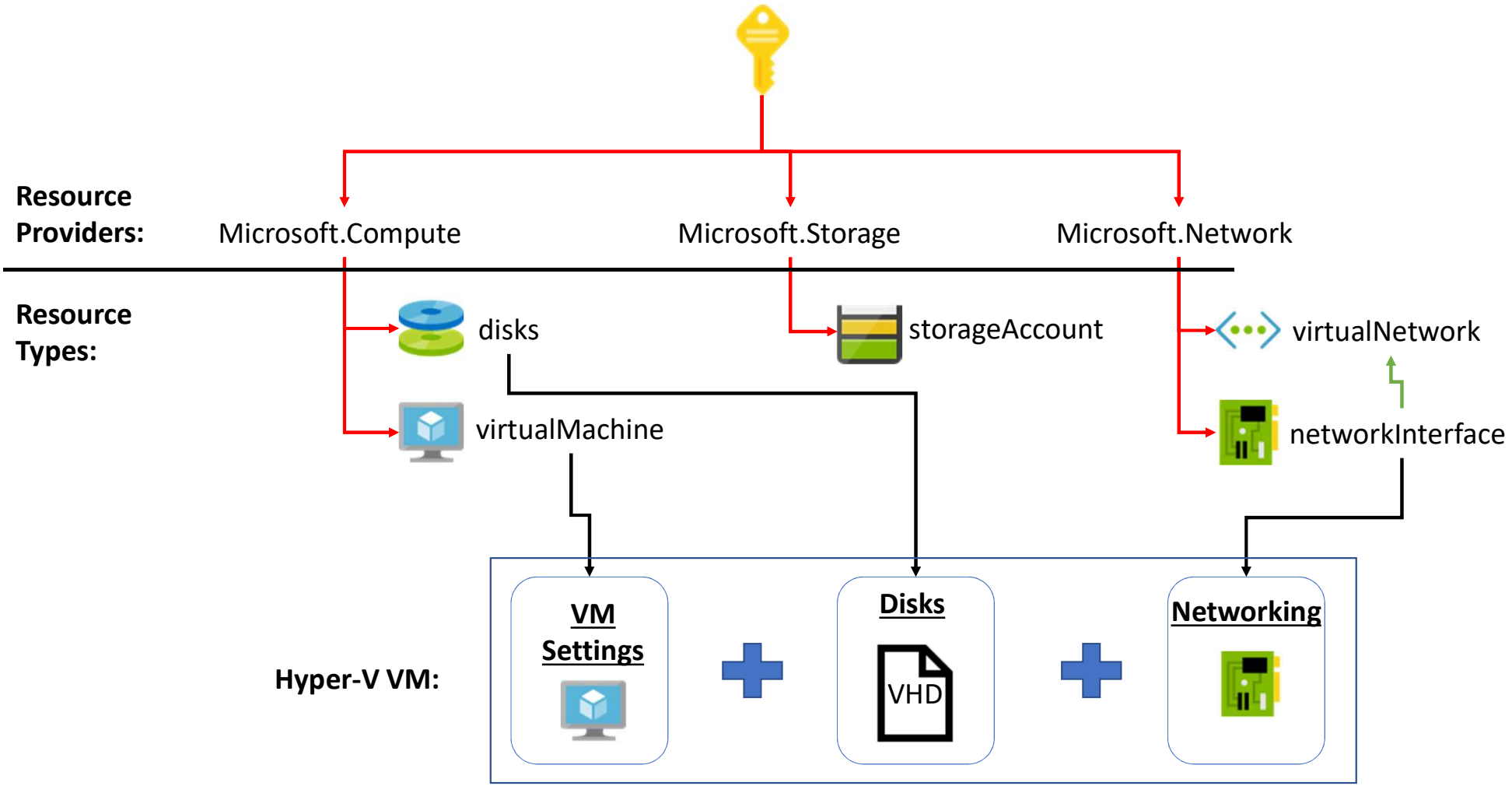

```powershell
$location = "East US"
$resourceProvider = "Microsoft.Compute"

$resourceTypes = Get-AzResourceProvider -ProviderNamespace "Microsoft.Compute"
foreach($resourceType in $resourceTypes)
{
    $resourceTypeAndApi = New-Object System.Collections.Hashtable
    foreach($resourceTypeName in $resourceType.ResourceTypes) {
        if($resourceTypeName.Locations -contains $location) {
            $mostRecentAPI = $resourceTypeName.ApiVersions | sort -Descending | select -First 1
            $resourceTypeAndApi.Add($resourceTypeName.ResourceTypeName, $mostRecentAPI)
        }
    }
    $resourceTypeAndApi.GetEnumerator() `
    | select @{Label="ResourceType";Expression={$_.Key}},@{Label="API Version";Expression={$_.Value}}
}
```

ResourceType	API Version
availabilitySets	2019-07-01
virtualMachines	2019-07-01
virtualMachines/extensions	2019-07-01
virtualMachineScaleSets	2019-07-01
virtualMachineScaleSets/extensions	2019-07-01
virtualMachineScaleSets/virtualMachines	2019-07-01
virtualMachineScaleSets/networkInterfaces	2019-07-01
virtualMachineScaleSets/virtualMachines/networkInterfaces	2019-07-01
virtualMachineScaleSets/publicIPAddresses	2018-10-01
locations/operations	2019-07-01
locations/vmSizes	2019-07-01
locations/runCommands	2019-07-01
locations/usages	2019-07-01
locations/virtualMachines	2019-07-01
locations/publishers	2019-07-01
operations	2019-07-01
restorePointCollections	2019-07-01
restorePointCollections/restorePoints	2019-07-01
proximityPlacementGroups	2019-07-01
virtualMachines/metricDefinitions	2014-04-01
sharedVMImages	2017-10-15-preview
sharedVMImages/versions	2017-10-15-preview
locations/artifactPublishers	2017-10-15-preview
locations/capsoperations	2019-03-01
galleries	2019-03-01
galleries/images	2019-03-01
galleries/images/versions	2019-03-01
disks	2019-03-01
snapshots	2019-03-01
locations/diskoperations	2019-03-01
locations/vsmoperations	2019-07-01
images	2019-07-01
locations/logAnalytics	2019-07-01

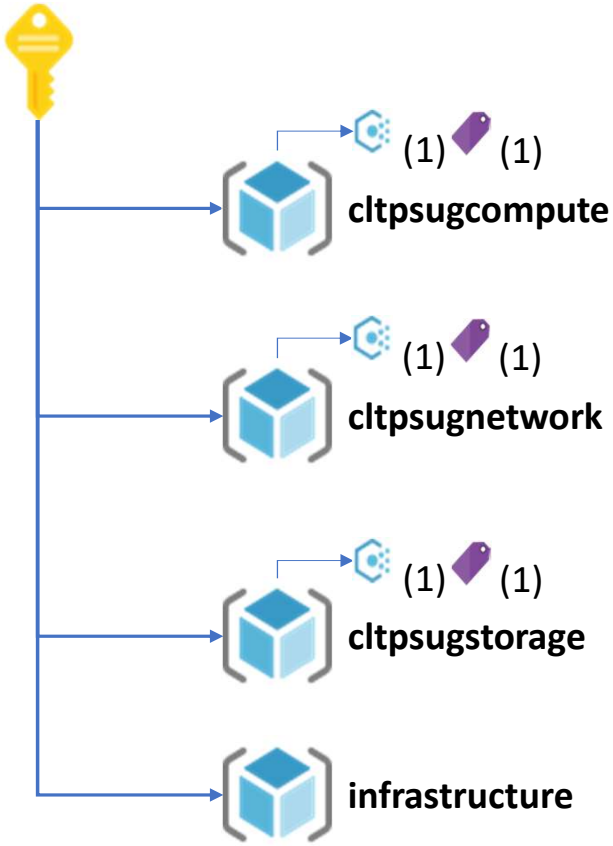
Note: I forgot in the code snippet:
Register-AzResourceProvider

Resource Identifier URI:
/subscriptions/{sub id}/resourceGroups/{rg name}/{provider name}/{resource type}/{resource name}

Azure Resource Types Required to Deploy a Hyper-V VM



Deployment: Resource Groups



Key:

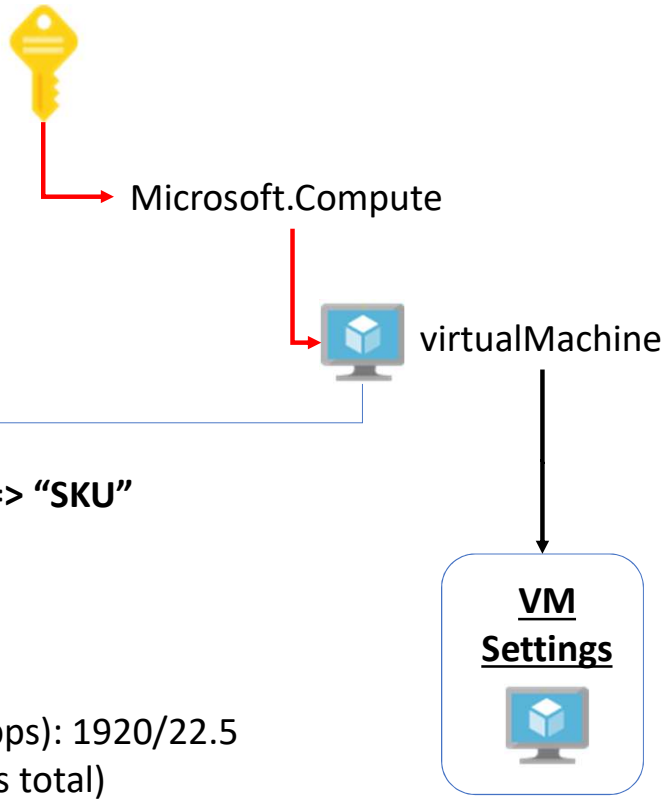
(1) Append Project Tag to Resources

(1) Project='cltpsug'

```powershell
$resourceLocation = "eastus"

#Resource Group Name
$vmResourceGroupName = "cltpsugcompute"
$networkResourceGroupName = "cltpsugnetwork"
$storageResourceGroupName = "cltpsugstorage"
$infrastructureResourceGroupName = "infrastructure";

#Project Tag
$tags = @{'Project'='cltpsug'}

#Create the Resource Groups w/Tags
try { $vmRG = Get-AzResourceGroup -Name $vmResourceGroupName `
    -ErrorAction Stop
    Set-AzResourceGroup -Name $vmResourceGroupName -Tag $tags }
catch { $vmRg = New-AzResourceGroup -Name $vmResourceGroupName `
                    -Location $resourceLocation `
                    -Tag $tags }

try { $networkRg = Get-AzResourceGroup -Name $networkResourceGroupName `
    -ErrorAction Stop
    Set-AzResourceGroup -Name $networkResourceGroupName -Tag $tags }
catch { $networkRg = New-AzResourceGroup -Name $networkResourceGroupName `
                    -Location $resourceLocation `
                    -Tag $tags }

try { $storageRg = Get-AzResourceGroup -Name $storageResourceGroupName `
    -ErrorAction Stop
    Set-AzResourceGroup -Name $storageResourceGroupName -Tag $tags }
catch { $storageRg = New-AzResourceGroup -Name $storageResourceGroupName `
                    -Location $resourceLocation `
                    -Tag $tags }

#Configure the Policy to apply the 'Project' Tag to all child Resources
$policyTagParameter = @{'tagName'='Project'}
$policyDefinition = Get-AzPolicyDefinition -BuiltIn `
 | where {$_.Properties.displayName -eq 'Append tag and `its value from the resource group'}

#Apply Policy to Resource Groups
$policyAssignmentName = "Append Project Tag to Resources"

try { Get-AzPolicyAssignment -Name $policyAssignmentName -Scope $vmRg.ResourceId `
    -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name $policyAssignmentName `
-Scope $vmRg.ResourceId -PolicyDefinition $policyDefinition `
-PolicyParameterObject $policyTagParameter }

try { Get-AzPolicyAssignment -Name $policyAssignmentName -Scope $networkRg.ResourceId `
    -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name $policyAssignmentName `
-Scope $networkRg.ResourceId -PolicyDefinition $policyDefinition `
-PolicyParameterObject $policyTagParameter }

try { Get-AzPolicyAssignment -Name $policyAssignmentName -Scope $storageRg.ResourceId `
    -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name $policyAssignmentName `
-Scope $storageRg.ResourceId -PolicyDefinition $policyDefinition `
-PolicyParameterObject $policyTagParameter }

#Create Resource Group without Tags
try { $infrastructureResourceGroup = Get-AzResourceGroup -Name $infrastructureResourceGroupName `
        -ErrorAction Stop }
catch { $infrastructureResourceGroup = New-AzResourceGroup -Name $infrastructureResourceGroupName `
                                        -Location $resourceLocation }
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateResourceGroups.ps1*

“virtualMachine” Resource Configuration



VM Size Example => “SKU”
vCPU: 2
RAM: 8GB
Swap: 16GB
Data Disks: 4
Disk Perf (IOPS/Mbps): 1920/22.5
NICs: 2 (1000 Mbps total)

OS Profile Example
Hostname: “windowsvm”
Admin Username: “vmadmin”
Admin Password: <prompt>
WinRM: https

Windows Profile
Admin User: vmadmin
Password: <prompt>

```
$resourceLocation = "eastus"
$vmName = "windowsvm"
$vmAdminUserName = "vmadmin"
$adminCredential = Get-Credential -UserName $vmAdminUserName `
-Message "Enter a password for vm admin acct";

#Enumerate all available VM Sizes in a Region
#Get-AzVMSize -Location "eastus"

$vmCfg = New-AzVMConfig -VMName $vmName -VMSize "Standard_B2ms"

#Windows VM
Set-AzVMOperatingSystem -VM $vmCfg -Windows -ComputerName $vmName `
-Credential $adminCredential
```

Linux Profile
Admin User: vmadmin
Password: <prompt>
SSH Key: <public key>

```
$resourceLocation = "eastus"
$vmName = "linuxvm"
$vmAdminUserName = "vmadmin"
$adminCredential = Get-Credential -UserName $vmAdminUserName `
-Message "Enter a password for vm admin acct";
$publicKey = "{enter a public key}"

$vmCfg = New-AzVMConfig -VMName $vmName -VMSize "Standard_B2ms"

#Linux VM
Set-AzVMOperatingSystem -VM $vmCfg -Windows -ComputerName $vmName `
-Credential $adminCredential
Add-AzVMSshPublicKey -VM $vmCfg -KeyData $publicKey `
-Path ("/home/{0}/.ssh/authorized_keys" -f $vmAdminUserName)
```

Comparing SKUs Families for General Usage

Standard_B2ms	Standard_A2_v2	Standard_D2s_v3	Standard_DS2_v2

• Base CPU: 60%
• Max CPU: 200%
• vCPU: 2
• Mem: 8GB
• Swap: 16GB
• Caching: No
• Data Disks: 4
• IOPS/Mbps:
  • 1920/22.5
• NICs: 3
• **Cost/Month**
  • Windows: $66.58
  • Windows (SA): $60.74
  • Linux: $60.74
	• ACU: 100
	• vCPU: 2
	• Mem: 4GB
	• Swap: 20GB
	• Caching: No
	• Data Disks: 4
	• IOPS:
	  • 2000
	• NICs: 2
	  • 500 Mbps
	• **Cost/Month**
	  • Windows: $99.28
	  • Windows (SA): $66.43
	  • Linux: $66.43
		• ACU: 160-190
		• vCPU: 2
		• **vCPU/Core: 2/1**
		• Mem: 8GB
		• Swap: 32GB
		• **Data Disks: 4**
		• Cached(50GB) IOPS/Mbps
		  • 4000/32
		• Non-Cached IOPS/Mbps
		  • 3200/48
		• NICs: 2
		  • 1 Gbps
		• **Cost/Month**
		  • Windows: $137.24
		  • Windows (SA): $70.08
		  • Linux: $70.08
			• ACU: 210-250
			• vCPU: 2
			• Mem: 7GB
			• Swap: 14GB
			• **Data Disks: 8**
			• Cached (43GB) IOPS/Mbps
			  • 8000/64
			• Non-Cached IOPS/Mbps
			  • 6400/96
			• NICs: 2
			  • 1.5 Gbps
			• **Cost/Month**
			  • Windows: $183.96
			  • Windows (SA): $106.58
			  • Linux: $106.58

https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general

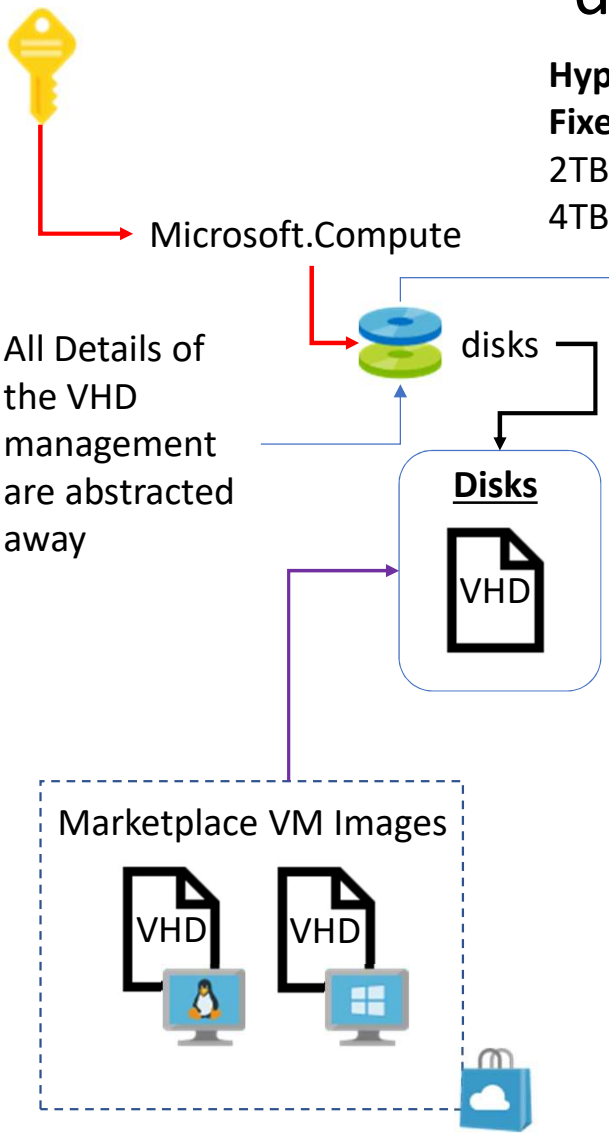
Notes:

- Premium Disk capable only
- East US, 730 hours/month, as of 7/2/2019
- "General Usage"
  - CPU to Memory Ratio
  - Ideal workloads
    - Small-Medium DBs
    - Low-Medium traffic Web Servers

Observations:

- A2 is the "standard" for this set of related SKU families (ACU 100)
- ACU increases
  - IOPS potential increases
  - Better CPUs
  - No core sharing
- B2ms is probably **on average** no better than the A2
  - But if your load is really variable, it could be a cost-friendly option

“disks” – Resource Configuration



Hyper-V VHD format
Fixed Size
2TB Max, OS Disk (Gen1 VM)
4TB Max, Data Disk

Windows:
Server 2016,Core

Cost = Allocated Size + Medium

Linux:
Ubuntu 18.04-LTS

Managed Disks
32GB OS Disk, Standard_HDD Medium

```
#Managed Disks, 32GB Standard Storage, Name: <vmName>OSDisk
$OSDiskCfg = @{
     Name = ("{0}OSDisk" -f $vmCfg.Name);
     StorageAccountType = "Standard_LRS";
     Caching = "None";
     CreateOption = "FromImage";
     DiskSizeInGB = 32}

#Configure for Windows Server 2016 Server Core
$mktplaceImageCfg.Add('PublisherName','MicrosoftWindowsServer');
$mktplaceImageCfg.Add('Offer','WindowsServer');
$mktplaceImageCfg.Add('Skus','2016-Datacenter-Server-Core-smalldisk');
$mktplaceImageCfg.Add('Version', 'latest');
Set-AzVMOSDisk -VM $vmCfg -Windows @OSDiskCfg
Set-AzVMSourceImage -VM $vmCfg @mktplaceImageCfg
```

```
#Managed Disks, 32GB Standard Storage, Name: <vmName>OSDisk
$OSDiskCfg = @{
     Name = ("{0}OSDisk" -f $vmCfg.Name);
     StorageAccountType = "Standard_LRS";
     Caching = "None";
     CreateOption = "FromImage";
     DiskSizeInGB = 32}

#Configure for Ubuntu 18.04-LTS
$mktplaceImageCfg = New-Object -TypeName System.Collections.Hashtable
$mktplaceImageCfg.Add('PublisherName','Canonical');
$mktplaceImageCfg.Add('Offer','UbuntuServer');
$mktplaceImageCfg.Add('Skus','18.04-LTS');
$mktplaceImageCfg.Add('Version', 'latest');
Set-AzVMOSDisk -VM $vmCfg -Linux @OSDiskCfg
Set-AzVMSourceImage -VM $vmCfg @mktplaceImageCfg
```

“disks” – Managed Disk Media Options

Scaling Limits

- Limit of 25,000 Disk resources/subscription

Premium SSD (“*Premium_LRS*” Type)

- Lowest latency and highest consistency
- Disks up to 7500 IOPS and 250 MBps
- Capacity costs only based on fixed sizes
- Target performance based on Specs

Standard SSD (“*StandardSSD_LRS*” Type)

- Higher latency and lower consistency
  - Disks Max at 500 IOPS
- Specs are “UP TO”
- Capacity + Operations Costs

Standard HDD (“*Standard_LRS*” Type)

- Highest latency and lowest consistency
  - Disks Max at 500 IOPS
- Specs are “UP TO” and Operations Costs are 1/4th of Standard SSD

virtualMachine + disks Costs

Premium Disks

- 4 x 1Tib (5000 IOPS:)) $540.69
- Standard_DS3_v2 (ACU 160-190): $213.80
  - **Total: $774.28**
- Standard_D8s_v3 (ACU 210-250): $280.32
  - **Total: $840.71**

Consider an Example

- IOPS Sizing Requirement: 16,000
- 4TB Data Disk Storage
- 128GB OS Disk
  - Premium Disk, $19.71/mo
- Managed Disks

Standard Disks

- 32 x 128 Gib (500 IOPS)
- Standard_D16_v3 (ACU 160-190): $560.64
  - **Total: $~768.63 + Operations**
- Standard_DS4_v2 (ACU 210-250): $427.05
  - **Total: $~635.24 + Operations**

Observations

- Have to trade off guaranteed minimum IOPS that exceeds our spec using Premium disks
- Savings isn't that large

Pricing as of 7/2/2019, East US, Azure Pricing Calculator

“disks” – OS VM Images

Image Sources (Regional)

- Publisher (e.g, Canonical)
- Offer (UbuntuServer)
- Sku (18.04-LTS)

```powershell
$resourceLocation = "eastus";
#List Publishers
Get-AzVMImagePublisher -Location $resourceLocation;

#List Offers (e.g., UbuntuServer)
$ubuntuMktplaceImageCfg = @{ PublisherName = "Canonical" }
Get-AzVMImageOffer -Location $resourceLocation @ubuntuMktplaceImageCfg

#List Skus
$ubuntuMktplaceImageCfg.Add('Offer','UbuntuServer')
Get-AzVMImageSku -Location $resourceLocation @ubuntuMktplaceImageCfg

#List Versions
$ubuntuMktplaceImageCfg.Add('Skus', '18.04-LTS')
Get-AzVMImage -Location $resourceLocation @ubuntuMktplaceImageCfg
```

Capture Generalized Windows VM

```powershell
#Assume the VM has been generalized
#This would require running sysprep on the VM
#Best to take a copy of the VM an generalize that
$resourceLocation = "eastus"
$vmName = "windowsvm"
$imageName = "windowsvm_image"
$vmResourceGroupName = "cltpsugcompute"

#Make sure the VM is stopped
Stop-AzVM -ResourceGroupName $vmResourceGroupName -Name $vmName `
    -Force

#Mark the VM as Generalized and create Image
Set-AzVM -ResourceGroupName $vmResourceGroupName `
    -Name $vmName -Generalized

$vm = Get-AzVM -ResourceGroupName $vmResourceGroupName `
    -Name $vmName

$vmImageCfg = New-AzImageConfig -Location $resourceLocation `
-SourceVirtualMachineId $vm.Id

New-AzImage -Image $vmImageCfg -ImageName $imageName `
-ResourceGroupName $vmResourceGroupName
```

Azure Marketplace

Globally Available:
Windows Server, Ubuntu, RedHat, SuSe

Images: Generalized Hyper-V VMs

Windows: sysprep
Linux: deprovisioned (waagent)

Custom Images (Location)

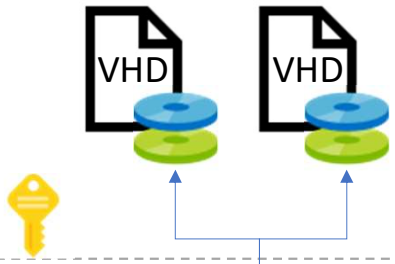
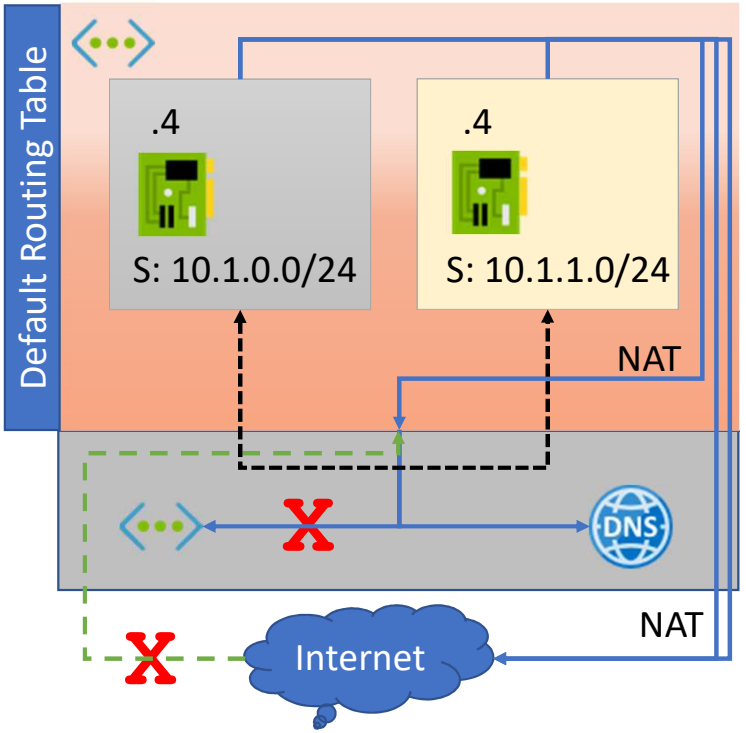


Image Sources

- On-Prem VM
- Azure VM
- Snapshot

“networking” – Resource Configuration



Virtual Network 10.1.0.0/16

Azure Backbone

Prefixes	Next Hop
10.1.0.0/16	Virtual Network
0.0.0.0/0	Internet
10.0.0.0/8	None
172.16.0.0/12	None
192.168.0.0/16	None

Virtual Network (Azure Resource)

- IPv4 private network address space that acts as an “Isolation Boundary”
  - Divided into non-overlapping Subnets
- Private IP addresses assigned by DHCP (default) to all IP configurations for a Network Interface
- Azure-provided DNS
  - Name assigned automatically for the private IP address of the primary IP configuration
  - Unique DNS suffixes for each Subnet (automatically defined)
- No inbound connections from outside the VNet can be initiated

Network Interface (Azure Resource)

- Must be bound to Virtual Network/Subnet when created
- At least 1 IPv4 Addressing configuration (“Primary”) must be defined
  - Private IPv4 address from within the Subnet and DNS name
- Can only be attached to a single VM
- VM must have at least one Network Interface attached at all times.

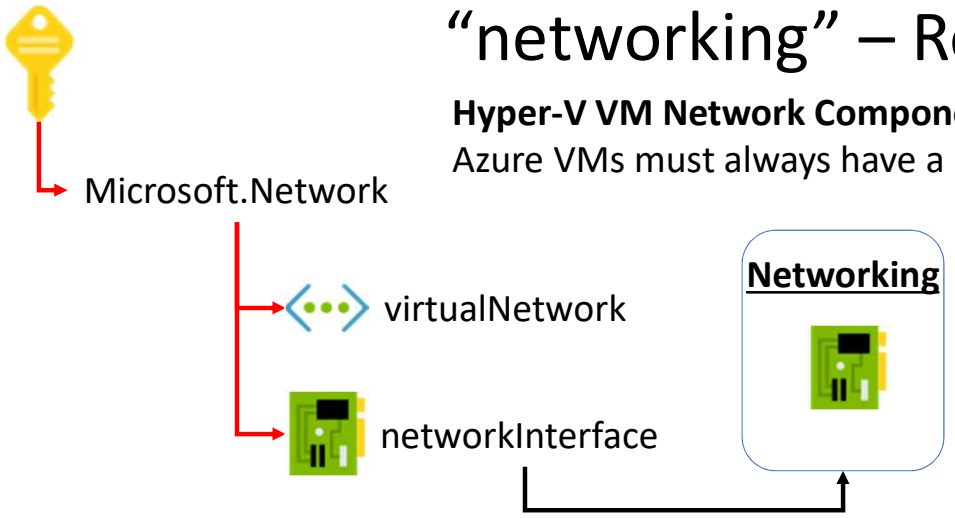
Default Routing

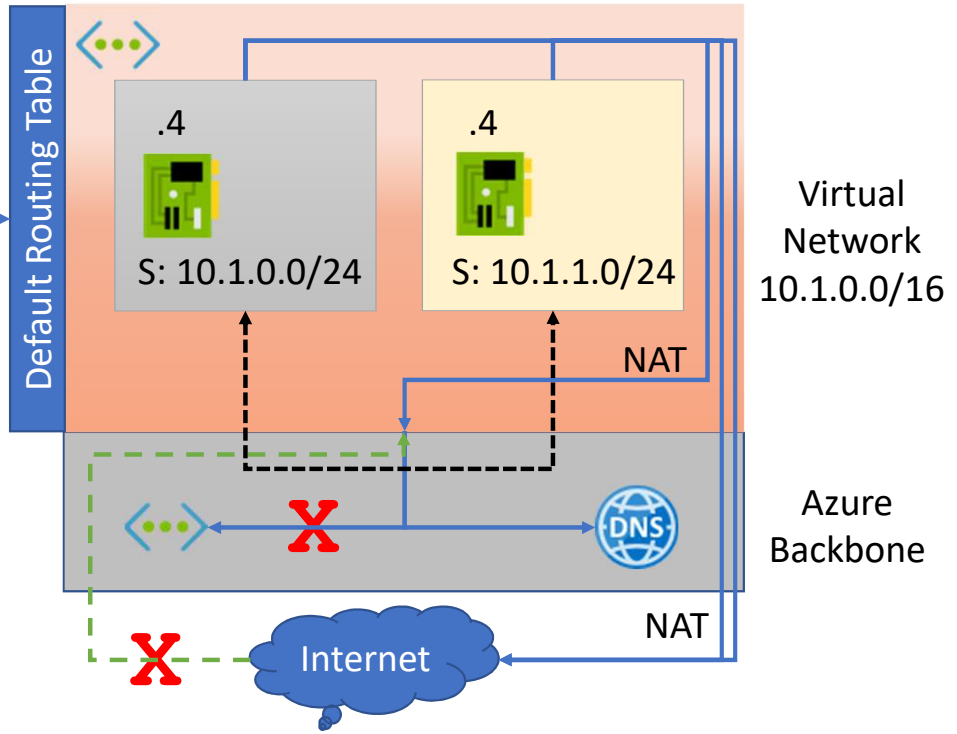
- Traffic can flow between Subnets
- All other traffic is routed to “Internet” using NAT
  - Traffic destined for Azure Services/DNS remains on Azure Backbone
- Its possible to add a “Route Table” to a Subnet to override default routes
  - To control how traffic flows to Subnets or force traffic through a NVA (Network Virtual Appliance)

“networking” – Resource Configuration

Hyper-V VM Network Component

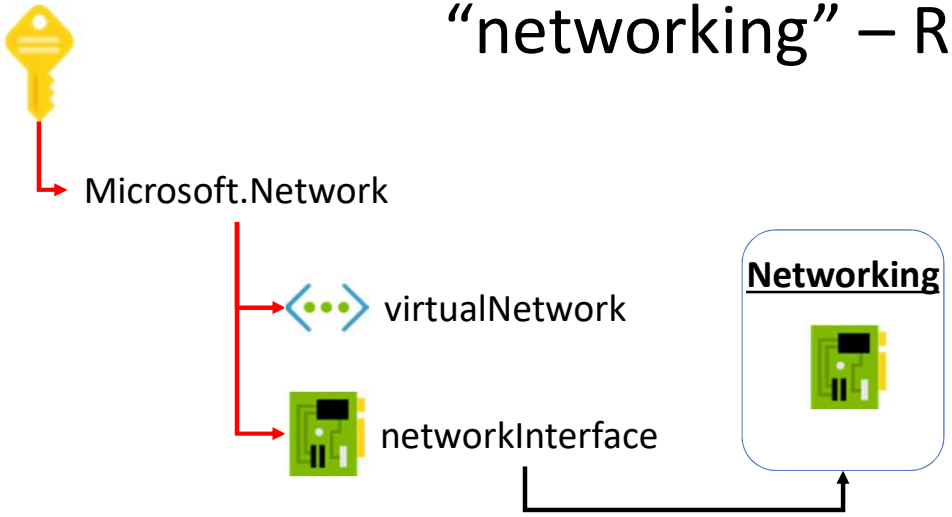
Azure VMs must always have a Network Interface



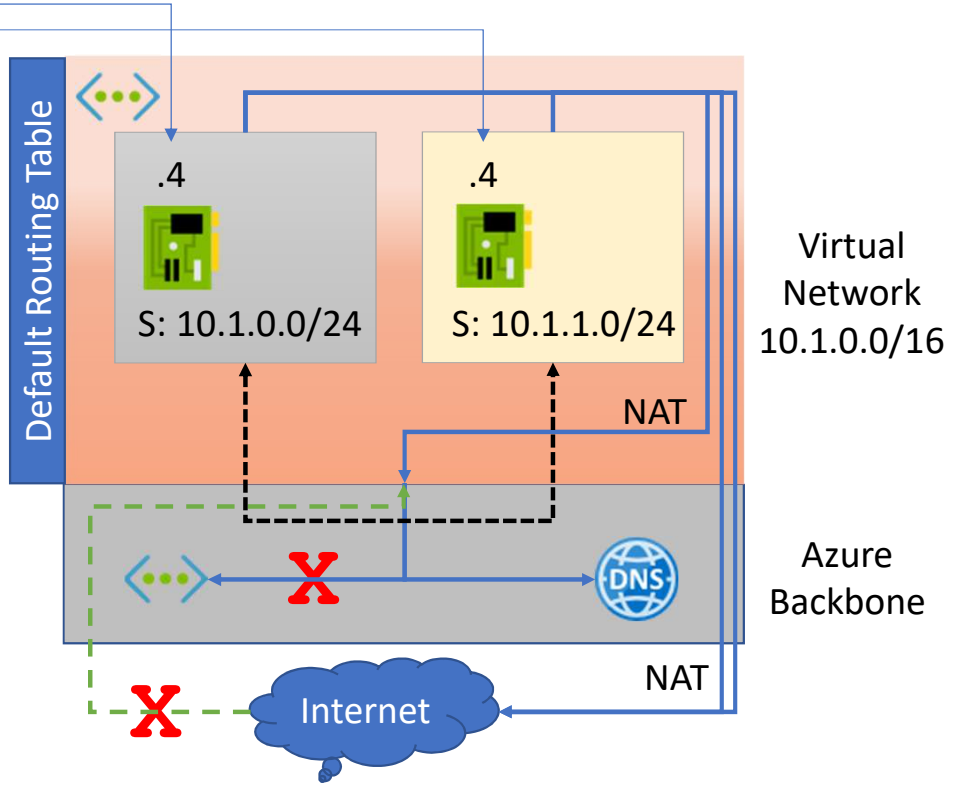



```powershell
$resourceLocation = "eastus"
$resourceGroupName = "cltpsugnetwork"
$virtualNetworkAddrSpace = "10.1.0.0/16"
$subnet1AddrSpace = "10.1.0.0/24"
$subnet2AddrSpace = "10.1.1.0/24"

$virtualNetworkCfg = @{
    Name = "vnet";
    ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
    AddressPrefix = $virtualNetworkAddrSpace;
}

$subnet1Cfg = @{Name = "subnet1"; AddressPrefix = $subnet1AddrSpace;}
$subnet2Cfg = @{Name = "subnet2"; AddressPrefix = $subnet2AddrSpace;}

#Get/Configure the Virtual Network
try { $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] `
    -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch { $vnet = New-AzVirtualNetwork @virtualNetworkCfg }

#Get/Configure the Subnets
try { $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet `
                        -Name $subnet1Cfg['Name'] -ErrorAction Stop }
catch {
    Add-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet @subnet1Cfg
    $vnet = ($vnet | Set-AzVirtualNetwork)
    $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name'] }

try { $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet `
                        -Name $subnet2Cfg['Name'] -ErrorAction Stop}
catch {
    Add-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet @subnet2Cfg
    $vnet = ($vnet | Set-AzVirtualNetwork)
    $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet2Cfg['Name'] }
```

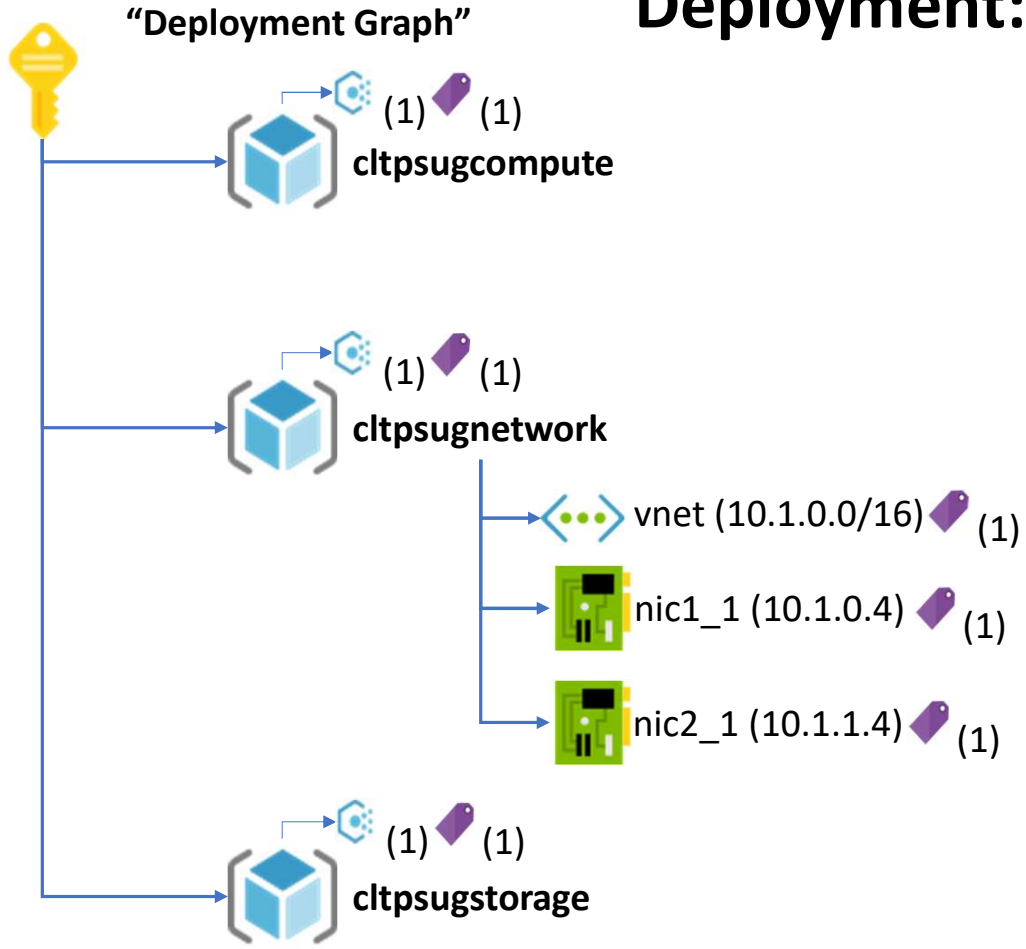
“networking” – Resource Configuration




```powershell
#Get/Configure the Network Interfaces
$subnet1Nic1Cfg = @{
    Name = "nic1_1"; ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
}
$subnet2Nic1Cfg = @{
    Name = "nic2_1"; ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation
}

$subnet1Nic1IPCfg = @{Name = "ipcfg1"; PrivateIpAddress = "10.1.0.4";}
$subnet2Nic1IPCfg1 = @{Name = "ipcfg1"; PrivateIpAddress = "10.1.1.4";}

#Create the 10.1.0.4 Network Interface (Subnet 1)
try { $nic1_1 = Get-AzNetworkInterface -Name $subnet1Nic1Cfg['Name'] `
        -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch {
    $nic1_1ipcfg1 = New-AzNetworkInterfaceIpConfig @subnet1Nic1IPCfg -Subnet $subnet1
    $nic1_1 = New-AzNetworkInterface -IpConfiguration $nic1_1ipcfg1 @subnet1Nic1Cfg
    $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] -ResourceGroupName $resourceGroupName
    $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name']
}

#Create the 10.1.1.4 Network Interface (Subnet 2)
try { $nic2_1 = Get-AzNetworkInterface -Name $subnet2Nic1Cfg['Name'] `
        -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch {
    $nic2_1ipcfg1 = New-AzNetworkInterfaceIpConfig @subnet2Nic1IPCfg1 -Subnet $subnet2
    $nic2_1 = New-AzNetworkInterface -IpConfiguration $nic2_1ipcfg1 @subnet2Nic1Cfg
    $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] -ResourceGroupName $resourceGroupName
    $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name']
}
```



Deployment: Networking



Key:

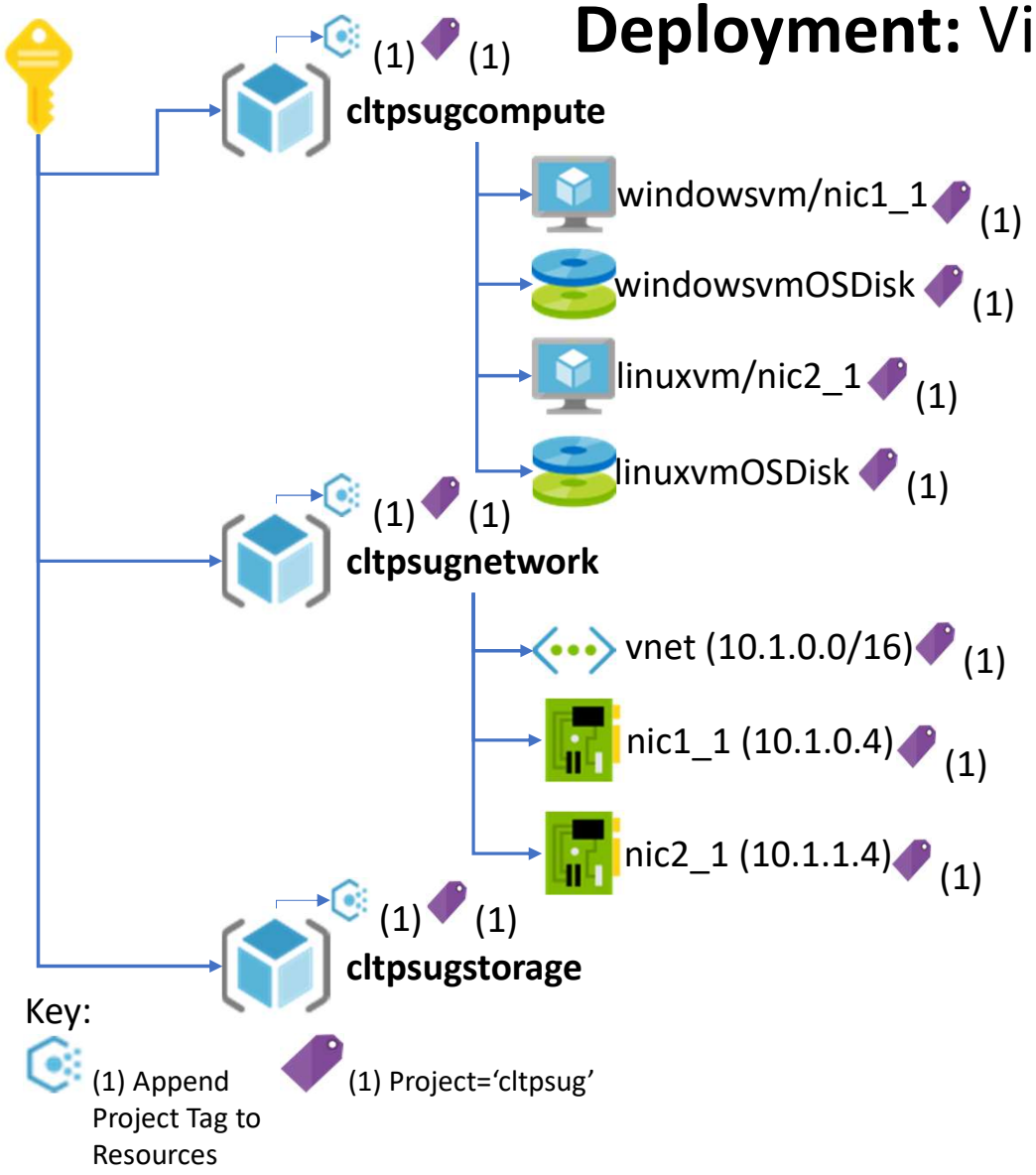
(1) Append Project Tag to Resources

(1) Project='cltpsug'

```powershell
$resourceLocation = "eastus"
$resourceGroupName = "cltpsugnetwork"
$virtualNetworkAddrSpace = "10.1.0.0/16"
$subnet1AddrSpace = "10.1.0.0/24"
$subnet2AddrSpace = "10.1.1.0/24"

$virtualNetworkCfg = @{
    Name = "vnet";
    ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
    AddressPrefix = $virtualNetworkAddrSpace;
}

$subnet1Cfg = @{Name = "subnet1"; AddressPrefix = $subnet1AddrSpace;}
$subnet2Cfg = @{Name = "subnet2"; AddressPrefix = $subnet2AddrSpace;}

#Get/Configure the Virtual Network
try { $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] `
    -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch { $vnet = New-AzVirtualNetwork @virtualNetworkCfg }

#Get/Configure the Subnets
try { $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet `
                          -Name $subnet1Cfg['Name'] -ErrorAction Stop }
catch {
    Add-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet @subnet1Cfg
    $vnet = ($vnet | Set-AzVirtualNetwork)
    $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name'] }

try { $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet `
                          -Name $subnet2Cfg['Name'] -ErrorAction Stop}
catch {
    Add-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet @subnet2Cfg
    $vnet = ($vnet | Set-AzVirtualNetwork)
    $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet2Cfg['Name'] }


#Get/Configure the Network Interfaces
$subnet1Nic1Cfg = @{
    Name = "nic1_1"; ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
}
$subnet2Nic1Cfg = @{
    Name = "nic2_1"; ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation
}

$subnet1Nic1IPCfg = @{Name = "ipcfg1"; PrivateIpAddress = "10.1.0.4";}
$subnet2Nic1IPCfg1 = @{Name = "ipcfg1"; PrivateIpAddress = "10.1.1.4";}

#Create the 10.1.0.4 Network Interface (Subnet 1)
try { $nic1_1 = Get-AzNetworkInterface -Name $subnet1Nic1Cfg['Name'] `
        -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch {
    $nic1_1ipcfg1 = New-AzNetworkInterfaceIpConfig @subnet1Nic1IPCfg -Subnet $subnet1
    $nic1_1 = New-AzNetworkInterface -IpConfiguration $nic1_1ipcfg1 @subnet1Nic1Cfg
    $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] -ResourceGroupName $resourceGroupName
    $subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name']
}

#Create the 10.1.1.4 Network Interface (Subnet 2)
try { $nic2_1 = Get-AzNetworkInterface -Name $subnet2Nic1Cfg['Name'] `
        -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch {
    $nic2_1ipcfg1 = New-AzNetworkInterfaceIpConfig @subnet2Nic1IPCfg1 -Subnet $subnet2
    $nic2_1 = New-AzNetworkInterface -IpConfiguration $nic2_1ipcfg1 @subnet2Nic1Cfg
    $vnet = Get-AzVirtualNetwork -Name $virtualNetworkCfg['Name'] -ResourceGroupName $resourceGroupName
    $subnet2 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name']
}
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateNetworkingConfiguration.ps1*

Deployment: Virtual Machines



windowsvm

```powershell
$resourceLocation = "eastus"
$vmName = "windowsvm"
$vmAdminUserName = "vmadmin"
$adminCredential = Get-Credential -UserName $vmAdminUserName `
-Message "Enter a password for vm admin acct";

$vmCfg = New-AzVMConfig -VMName $vmName -VMSize "Standard_B2ms"

#Windows VM
Set-AzVMOperatingSystem -VM $vmCfg -Windows -ComputerName $vmName `
-Credential $adminCredential

#Managed Disks, 32GB Standard Storage, Name: <vmName>OSDisk
$OSDiskCfg = @{ Name = ("{0}OSDisk" -f $vmCfg.Name); StorageAccountType = "Standard_LRS"; `
    Caching = "None"; CreateOption = "FromImage"; DiskSizeInGB = 32}
#Configure for Windows Server 2016 Server Core
$mktplaceImageCfg = @{ PublisherName = "MicrosoftWindowsServer"; Offer = "WindowsServer"; `
    Skus = "2016-Datacenter-Server-Core-smalldisk"; Version = "latest" }
Set-AzVMOSDisk -VM $vmCfg -Windows @OSDiskCfg
Set-AzVMSourceImage -VM $vmCfg @mktplaceImageCfg
#Disable Boot Diags so we can set them in a different part of the script
Set-AzVMBootDiagnostic -VM $vmCfg -Disable

#Resource Groups
$vmResourceGroupName = "cltpsugcompute"
$networkingResourceGroupName = "cltpsugnetwork"
#Network Interface
$nicName = "nic1_1";

#Network Interface nic1_1: (10.1.0.4)
$nic = Get-AzNetworkInterface -Name $nicName -ResourceGroupName $networkingResourceGroupName
Add-AzVMNetworkInterface -VM $vmCfg -NetworkInterface $nic

#Create the VM
New-AzVM -VM $vmCfg -ResourceGroupName $vmResourceGroupName -Location $resourceLocation
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateWindowsVM.ps1*

linuxvm

```powershell
$resourceLocation = "eastus"
$vmName = "linuxvm"
$vmAdminUserName = "vmadmin"
$adminCredential = Get-Credential -UserName $vmAdminUserName `
-Message "Enter a password for vm admin acct";
#Read in an SSH Public Key or set to empty string
$publicKey = ""

$vmCfg = New-AzVMConfig -VMName $vmName -VMSize "Standard_B2ms"

#Linux VM
Set-AzVMOperatingSystem -VM $vmCfg -Linux -ComputerName $vmName `
-Credential $adminCredential
if($publicKey.Length -gt 0) {
    Write-Output "Configuring Admin SSH Public Key"
    Add-AzVMSshPublicKey -VM $vmCfg -KeyData $publicKey `
    -Path ("/home/{0}/.ssh/authorized_keys" -f $vmAdminUserName)
}

#Managed Disks, 32GB Standard Storage, Name: <vmName>OSDisk
$OSDiskCfg = @{ Name = ("{0}OSDisk" -f $vmCfg.Name); StorageAccountType = "Standard_LRS"; `
    Caching = "None"; CreateOption = "FromImage"; DiskSizeInGB = 32}
#Configure for Ubuntu 18.04-LTS
$mktplaceImageCfg = @{ PublisherName = "Canonical"; Offer = "UbuntuServer"; `
    Skus = "18.04-LTS"; Version = "latest" }
Set-AzVMOSDisk -VM $vmCfg -Linux @OSDiskCfg
Set-AzVMSourceImage -VM $vmCfg @mktplaceImageCfg
#Disable Boot Diags so we can set them in a different part of the script
Set-AzVMBootDiagnostic -VM $vmCfg -Disable

#Resource Groups
$vmResourceGroupName = "cltpsugcompute"
$networkingResourceGroupName = "cltpsugnetwork"
#Network Interface
$nicName = "nic2_1";

#Network Interface nic2_1: (10.1.1.4)
$nic = Get-AzNetworkInterface -Name $nicName -ResourceGroupName $networkingResourceGroupName
Add-AzVMNetworkInterface -VM $vmCfg -NetworkInterface $nic

#Create the VM
New-AzVM -VM $vmCfg -ResourceGroupName $vmResourceGroupName -Location $resourceLocation
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateLinuxVM.ps1*

VM Boot Diagnostics

Boot Diagnostics

- Boot Screen Captures
- Serial Console Logs
- Requires a Storage Account
  - Storage Account Firewall must allow “All networks”
  - Use a “cheap” configuration and share among multiple VMs

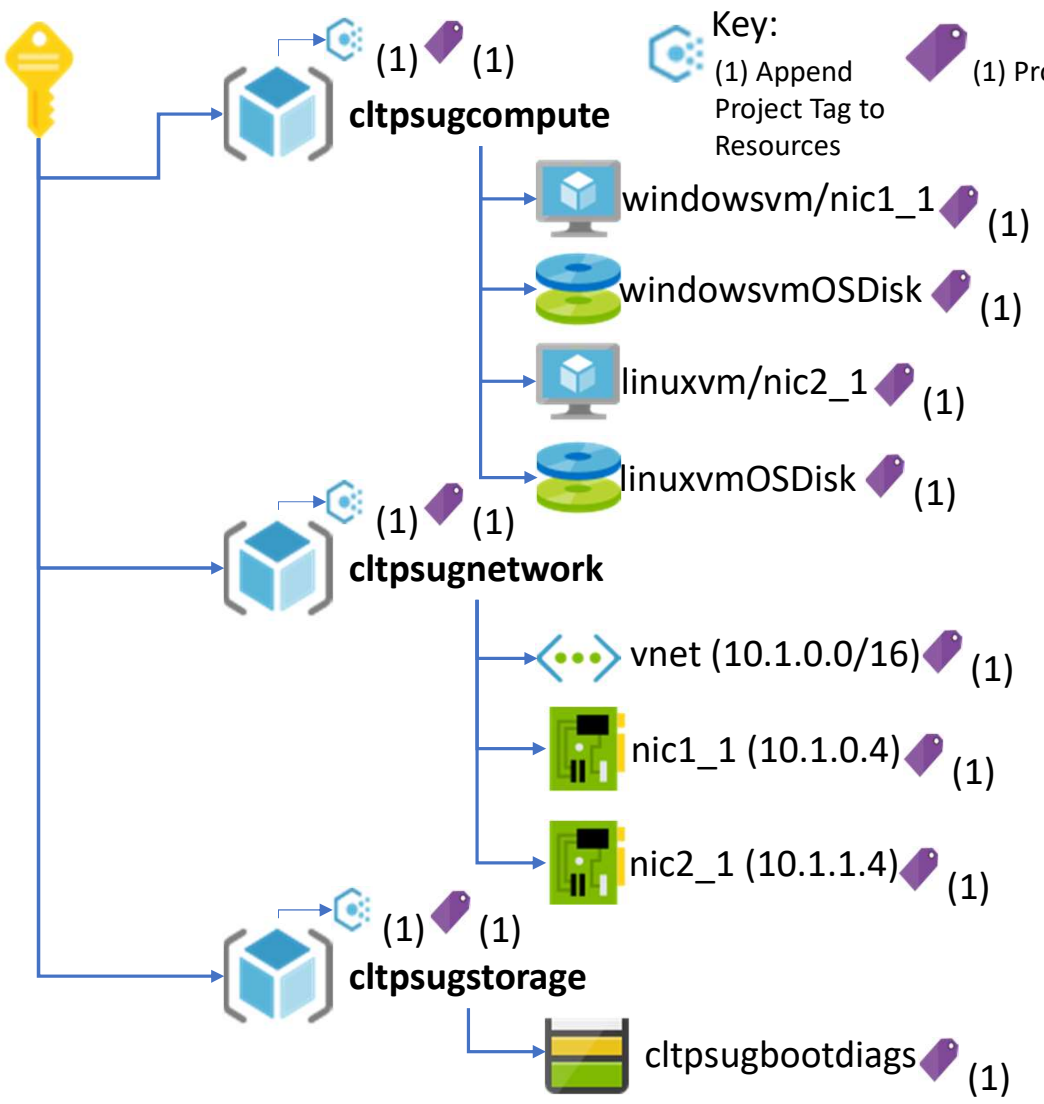
Serial Console

- Windows
  - Start a Command Prompt on channel 1
  - Login with the Admin username and password configured during VM creation
- Linux
  - Remote Serial Console is configured automatically on ttys0 for Marketplace images in /etc/inittab
    - /sbin/agetty –L 115200 console vt102
- Login for Full Console Access

Command Prompt

- Run any command
- Use Powershell for Windows

Deployment: VM Boot Diagnostics



windows

```powershell
$resourceLocation = "eastus"
#Resource Groups
$vmResourceGroupName = "cltpsugcompute"
$storageResourceGroupName = "cltpsugstorage"

$bootdiagsStorageAccountName = "cltpsugbootdiags"
#Create a Diagnostics Storage Account if necessary
try { Get-AzStorageAccount -Name $bootdiagsStorageAccountName `
    -ResourceGroupName $storageResourceGroupName -ErrorAction Stop}
catch {

    New-AzStorageAccount -Name $bootdiagsStorageAccountName `
    -ResourceGroupName $storageResourceGroupName `
    -Location $resourceLocation -SkuName Standard_LRS `
    -Kind StorageV2
}

$windowsVMName = "windowsvm"
$windowsVM = Get-AzVM -Name $windowsVMName `
 -ResourceGroupName $vmResourceGroupName
Set-AzVMBootDiagnostic -VM $windowsVM -Enable `
 -StorageAccountName $bootdiagsStorageAccountName `
 -ResourceGroupName $storageResourceGroupName
Update-AzVM -VM $windowsVM -ResourceGroupName $vmResourceGroupName
```

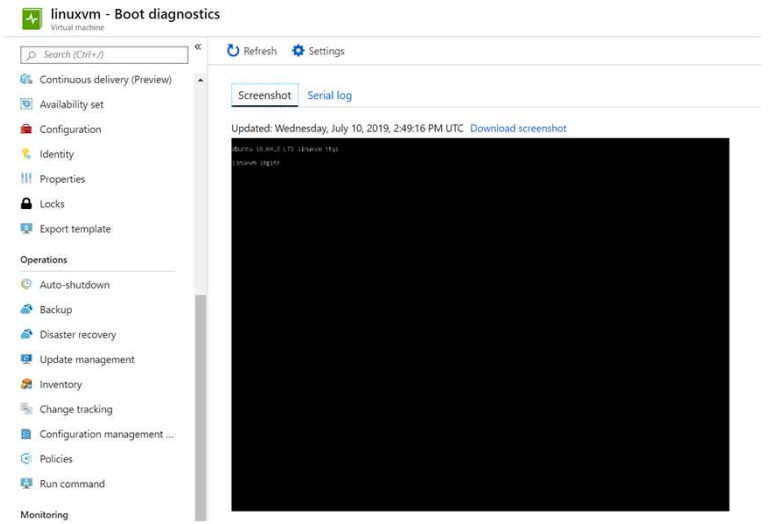
URI: *github.com/coderUT/cltpsug-july2019/Code/ConfigureWindowsVMBootDiags.ps1*

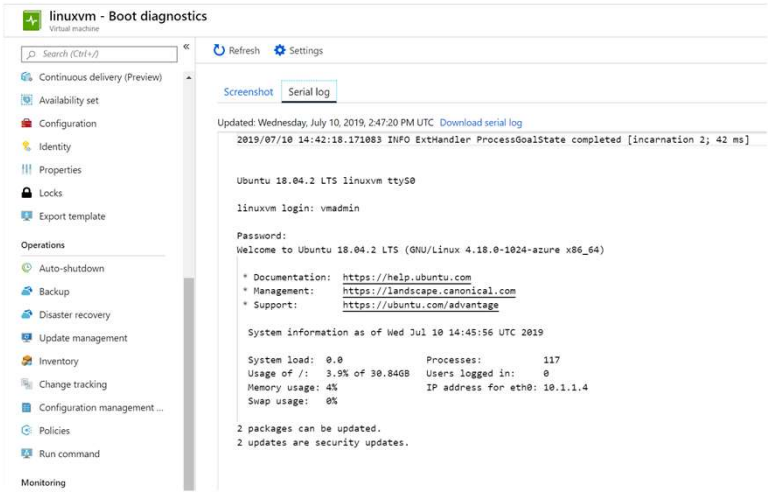
linux

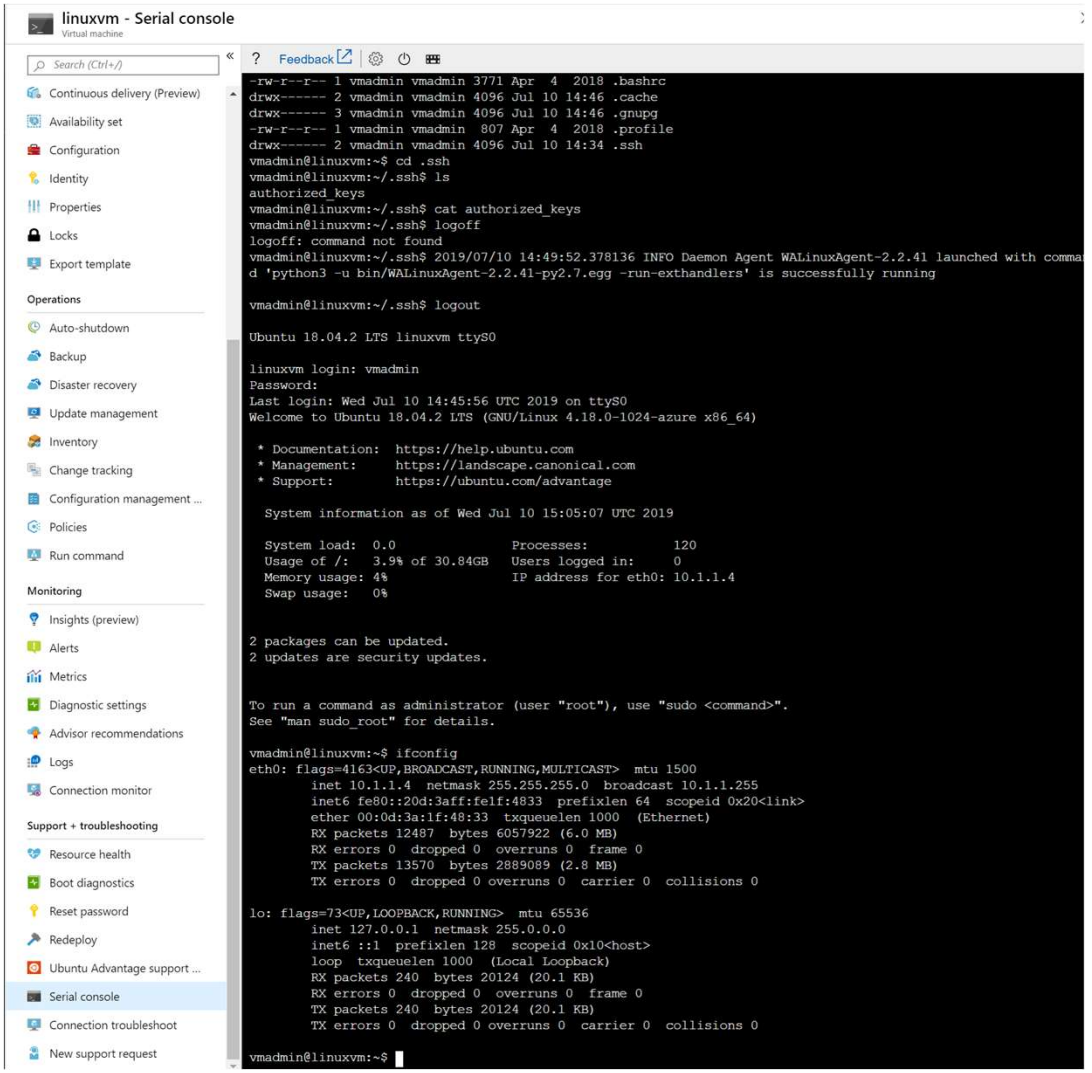
```powershell
$resourceLocation = "eastus"
#Resource Groups
$vmResourceGroupName = "cltpsugcompute"
$storageResourceGroupName = "cltpsugstorage"

$bootdiagsStorageAccountName = "cltpsugbootdiags"
#Create a Diagnostics Storage Account if necessary
try { Get-AzStorageAccount -Name $bootdiagsStorageAccountName `
    -ResourceGroupName $storageResourceGroupName -ErrorAction Stop}
catch {

    New-AzStorageAccount -Name $bootdiagsStorageAccountName `
    -ResourceGroupName $storageResourceGroupName `
    -Location $resourceLocation -SkuName Standard_LRS `
    -Kind StorageV2
}

$linuxVMName = "linuxvm"
$linuxVM = Get-AzVM -Name $linuxVMName `
 -ResourceGroupName $vmResourceGroupName
Set-AzVMBootDiagnostic -VM $linuxVM -Enable `
 -StorageAccountName $bootdiagsStorageAccountName `
 -ResourceGroupName $storageResourceGroupName
Update-AzVM -VM $linuxVM -ResourceGroupName $vmResourceGroupName
```

URI: *github.com/coderUT/cltpsug-july2019/Code/ConfigureLinuxVMBootDiags.ps1*

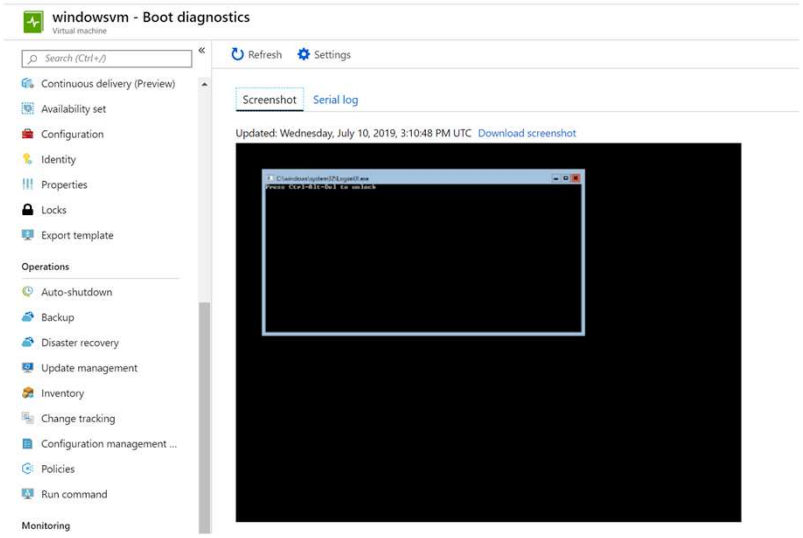
Post-Deployment: VM Boot Diagnostics, Linux

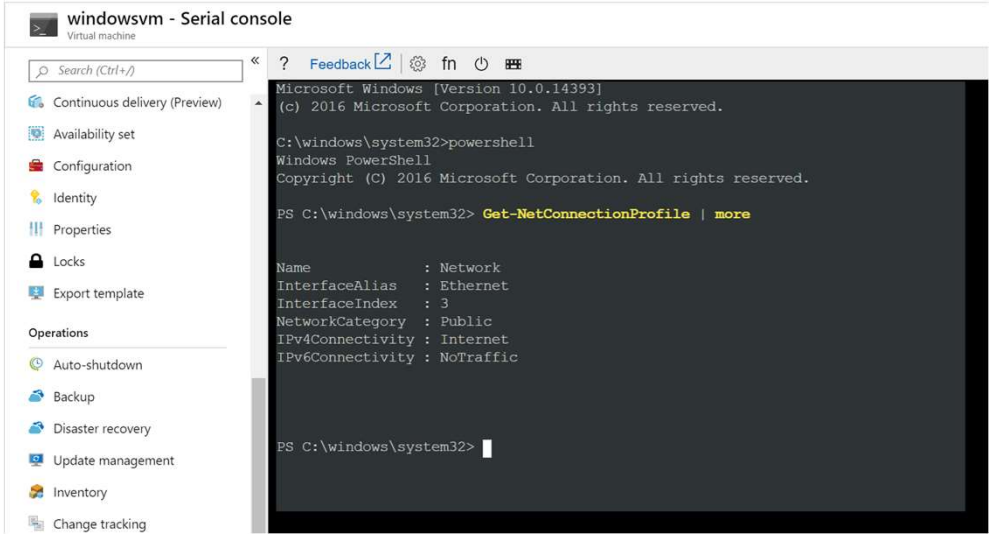


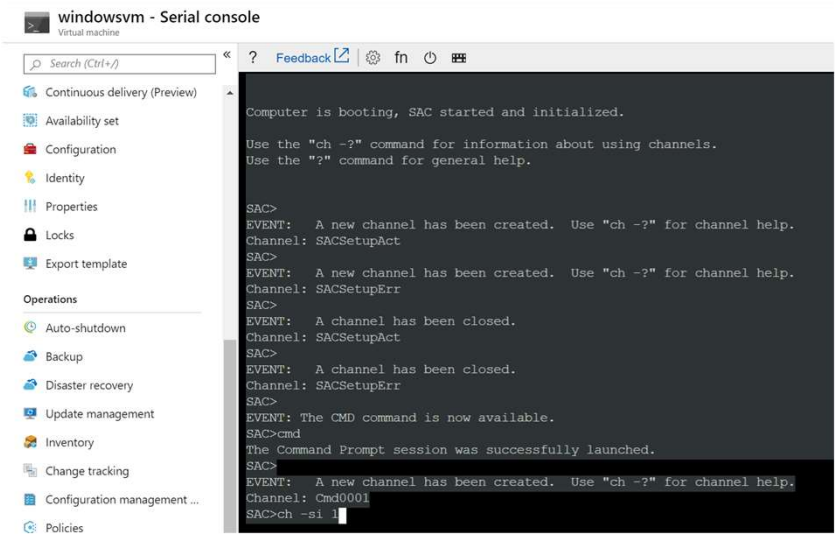


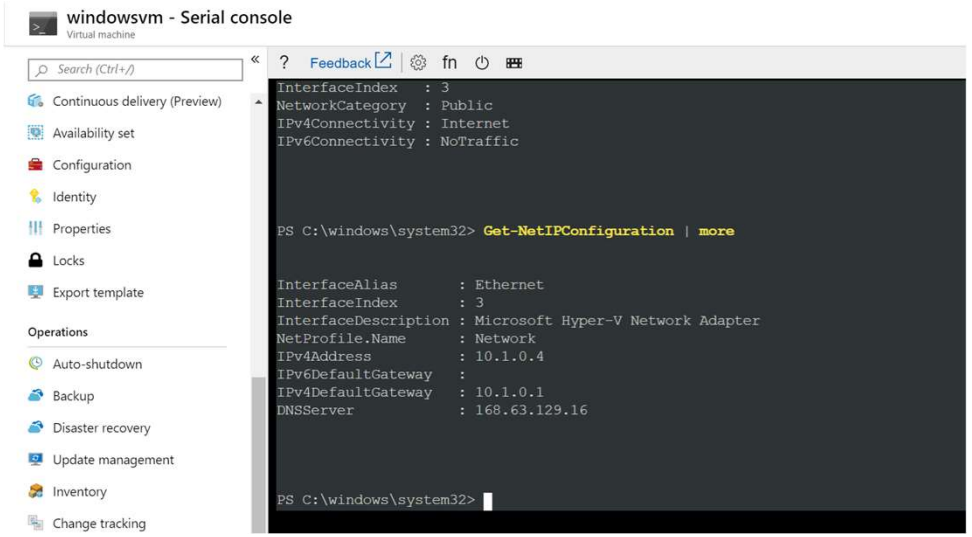


Post-Deployment: VM Boot Diagnostics, Windows

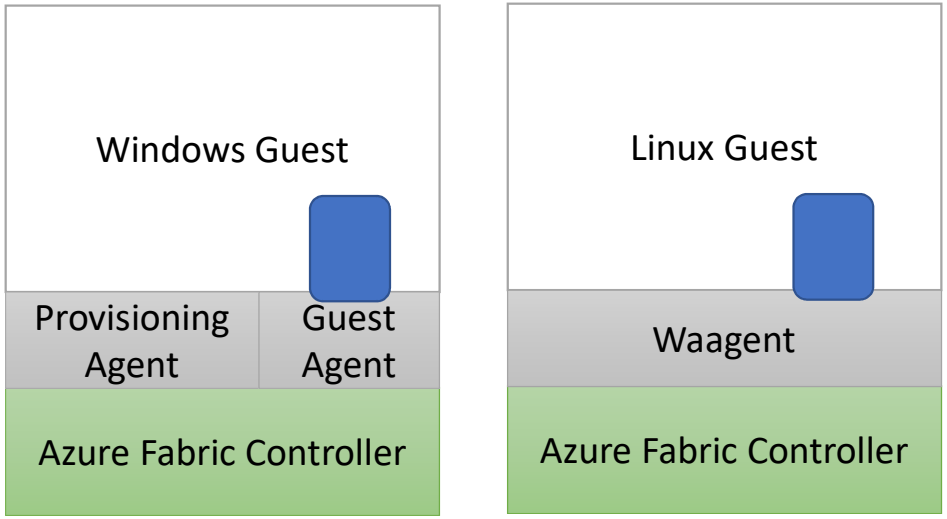








VMAgent and VMExtensions



Key:

VMAgent

- VM Guest to Azure interactions
- Enables VM Extensions to be managed via Azure Platform
- Required for VM to boot
- Use to “generalize” a Linux VM

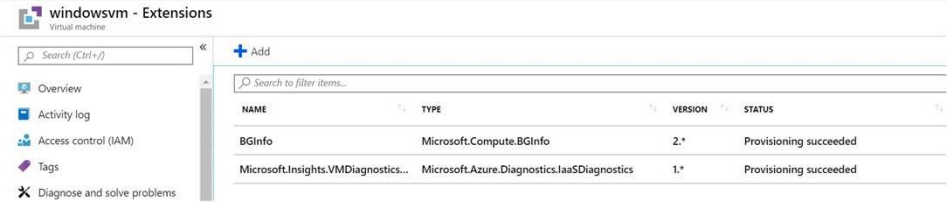
VMExtensions

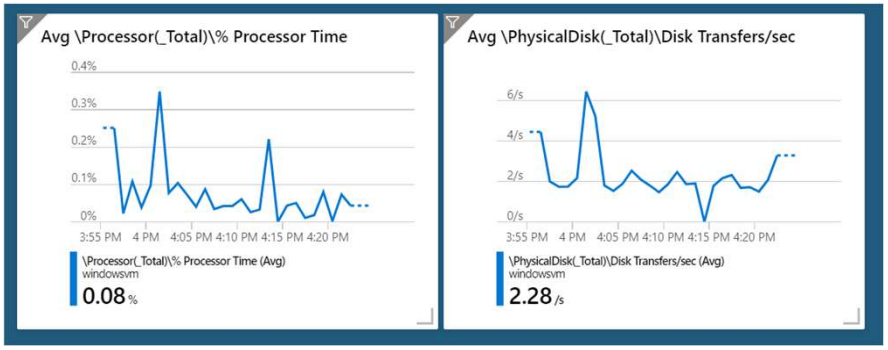
- Requires “Guest Agent” on Windows
  - waagent is already required on Linux
- Small applications useful for monitoring, configuration and automation
  - Diagnostics
  - Powershell DSC
  - Custom Script Extension
  - Azure Disk Encryption
  - BGInfo Extension
  - Azure Monitor Logs/Log Analytics (Microsoft Monitoring Agent)

Deployment: VMDiagnostics Extension

```xml
<?xml version="1.0" encoding="utf-8"?>
<DiagnosticsConfiguration xmlns="http://schemas.microsoft.com/ServiceHosting/2010/10/DiagnosticsConfiguration">
  <PublicConfig>
    <WadCfg>
      <DiagnosticMonitorConfiguration>
        <PerformanceCounters scheduledTransferPeriod="PT5M">
          <PerformanceCounterConfiguration counterSpecifier="\Processor(_Total)\% Processor Time"
                                           sampleRate="PT30S"
                                           unit="Percent"/>
          <PerformanceCounterConfiguration counterSpecifier="\PhysicalDisk(_Total)\Disk Transfers/sec"
                                           sampleRate="PT30S"
                                           unit="CountPerSecond"/>
        </PerformanceCounters>
        <WindowsEventLog scheduledTransferPeriod="PT5M">
          <DataSource name="System!*[System[(Level &lt;= 4)]]"/>
        </WindowsEventLog>
        <Metrics resourceId="">
          <MetricAggregation scheduledTransferPeriod="PT1H"/>
          <MetricAggregation scheduledTransferPeriod="PT1M"/>
        </Metrics>
      </DiagnosticMonitorConfiguration>
    </WadCfg>
  </PublicConfig>
  <IsEnabled>true</IsEnabled>
</DiagnosticsConfiguration>
```

URI: *github.com/coderUT/cltpsug-july2019/Code/WadCfg.template.xml*





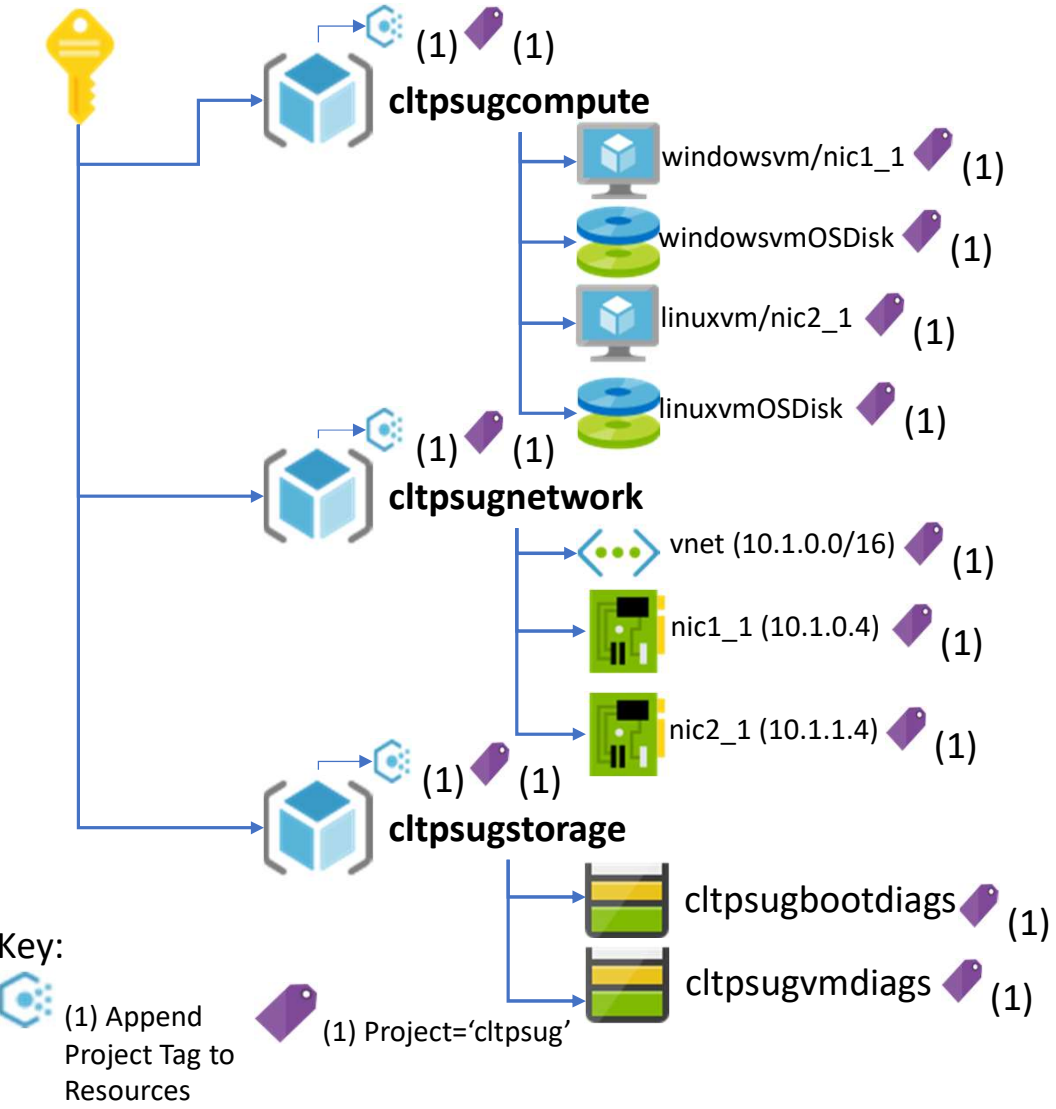
Data Storage

- Azure Storage Account
  - Again, I would use a “cheap” configuration
- Uses Table or Blob Storage
- Retention and Azure Monitor Logs/Metrics
  - Retention in the Storage Account is pretty much forever
  - If you have it ingested to Azure Monitor Logs, then there is a cost for Ingestion and a cost/month for Retention

Configuration

- Diagnostics Monitor Configuration element
  - How much local VM disk space is used
  - How often data is transferred to the Storage Account
  - This can be overridden for individual Performance Counters or Logs
  - Additional Data Destinations
- Windows
  - Specific Performance Counters
  - Windows Event Log
    - All or just specific Events from specific Logs
  - Metrics
    - Performance Counter data in a single Table for optimized querying
    - These “classic” metrics are being retired
      - I had trouble getting the new Azure Monitor Sink to work

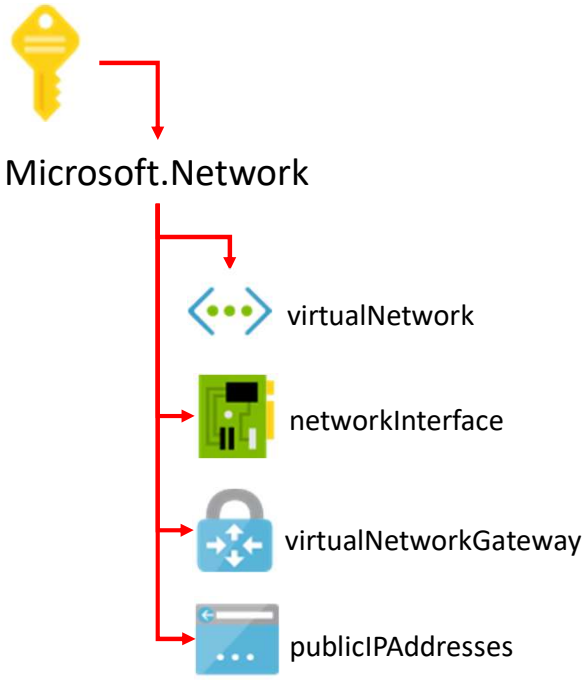
Deployment: VMDiagnostics Extension

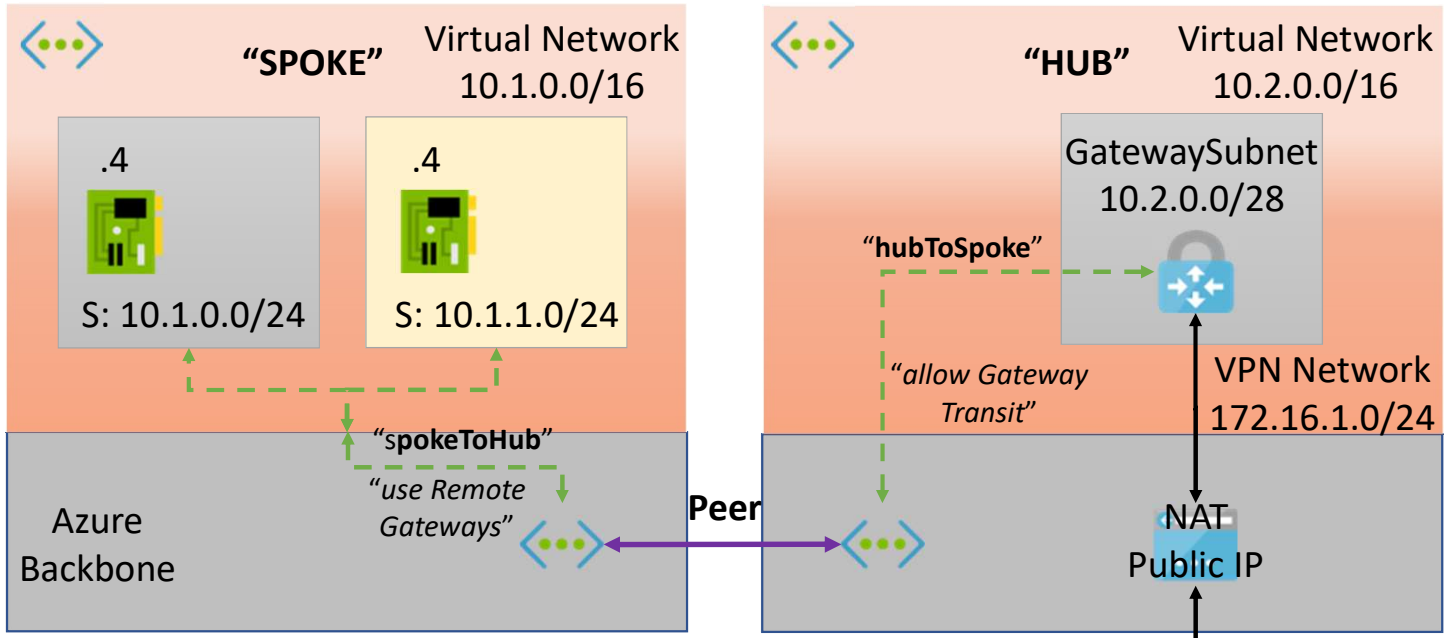



```powershell
$resourceLocation = "eastus"

#Resource Groups
$vmResourceGroupName = "cltpsugcompute"
$storageResourceGroupName = "cltpsugstorage";

$vmdiagsStorageAccountName = "cltpsugvmdiags";
#Create a VM Diagnostics Storage Account if necesssary
try { Get-AzStorageAccount -Name $vmdiagsStorageAccountName `
      -ResourceGroupName $storageResourceGroupName -ErrorAction Stop}
catch {
    New-AzStorageAccount -Name $vmdiagsStorageAccountName `
                         -ResourceGroupName $storageResourceGroupName `
                         -Location $resourceLocation `
                         -SkuName Standard_LRS -Kind StorageV2
}

$vmName = "windowsvm"
#Access VM Runtime Object
try {
$vmRunTime = Get-AzVM -Name $vmName -ResourceGroupName $vmResourceGroupName `
-Status -ErrorAction Stop
$vmCfg = Get-AzVM -Name $vmName -ResourceGroupName $vmResourceGroupName
}
catch {
    Write-Output ("Error accessing VM {0} status" -f $vmName)
    Exit
}

#Status code is "PowerState/deallocated" if stopped and deallocated
#VM needs to be running to install VM extensions
if(($vmRunTime.Statuses | where {$_.Code -eq "PowerState/running"}).Count -eq 1) {
    #Check the Guest Agent Status
    if(($vmRunTime.VMAgent.Statuses | where {($_.Code -eq "ProvisioningState/succeeded") -and `
            ($_.DisplayStatus -eq "Ready")}).Count -eq 1) {

        #Update The WadCfg Template
        $diagnosticsTemplateCfgPath = ("{0}\WadCfg.template.xml" -f (Get-Location).Path)
        $diagnosticsCfgPath = ("{0}\WadCfg.xml" -f (Get-Location).Path)

        $wadxml = [xml](Get-Content $diagnosticsTemplateCfgPath)
        $wadxml.DiagnosticsConfiguration.PublicConfig.WadCfg.DiagnosticMonitorConfiguration.Metrics.`
        SetAttribute("resourceId",$vmCfg.Id)
        $wadxml.Save($diagnosticsCfgPath)

        Set-AzVMDiagnosticsExtension -VMName $vmName -ResourceGroupName $vmResourceGroupName `
                                     -autoUpgradeMinorVersion $true `
                                     -DiagnosticsConfigurationPath $diagnosticsCfgPath `
                                     -StorageAccountName $vmdiagsStorageAccountName
    } else {
        Write-Output ("VM {0}, VMAgent must be provisioned and ready to install VM Extensions" -f $vmName)
    }
} else {
    Write-Output ("VM {0} needs to be running to install VM Extensions" -f $vmName)
}
```

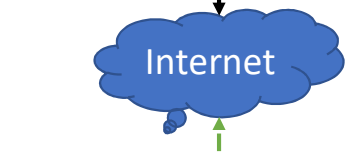
URI: *github.com/coderUT/cltpsug-july2019/Code/InstallVMDiagnosticsExtension.ps1*

VPN and Hub/Spoke Network Topology





Prefixes	Next Hop
172.16.1.0/24	Virtual Network Gateway
10.1.0.0/16	Virtual Network
10.2.0.0/16	Virtual Network Peering
0.0.0.0/0	Internet



SSTP/443

VPN Client
172.16.1.4

Prefixes	Next Hop
172.16.1.0/24	Virtual Network Gateway
10.2.0.0/16	Virtual Network
10.1.0.0/16	Virtual Network Peering
0.0.0.0/0	Internet

VPN Gateway Configuration

VPN Gateway Configuration

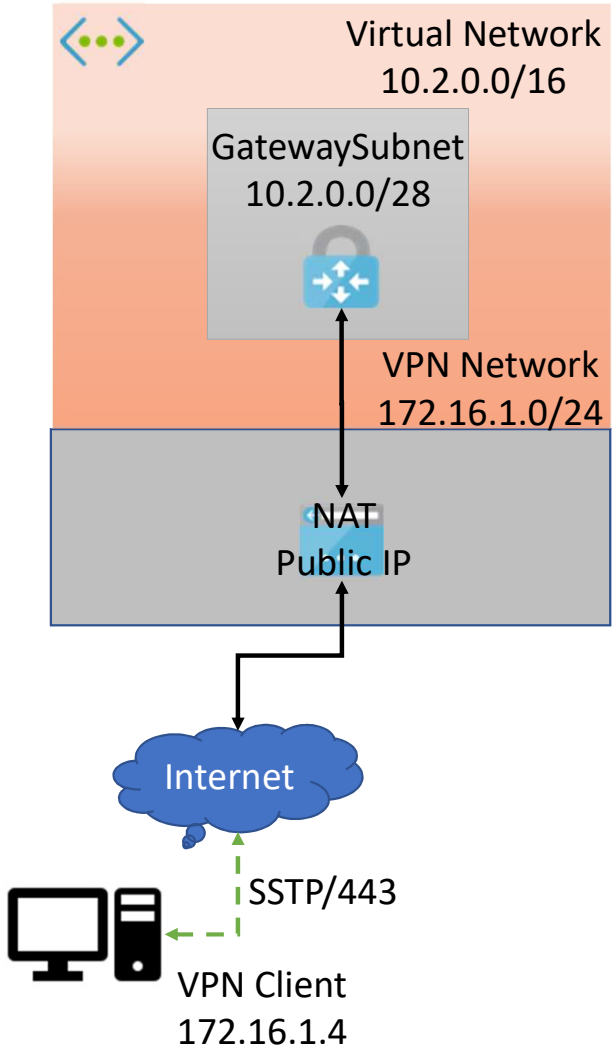
- Only 1 Virtual Network Gateway/Virtual Network
- Must be in its own Subnet named “GatewaySubnet”
  - Subnet can be as small as /29
  - Underlying implementation deploys multiple VMs as Network Virtual Appliances
- Different SKUs basically place limits on Bandwidth, Tunnel Count, Protocols
- Require a Public IP Resource to be associated
- Capable of Point to Site, Site to Site and Virtual Network to Virtual Network VPNs

Point to Site VPN

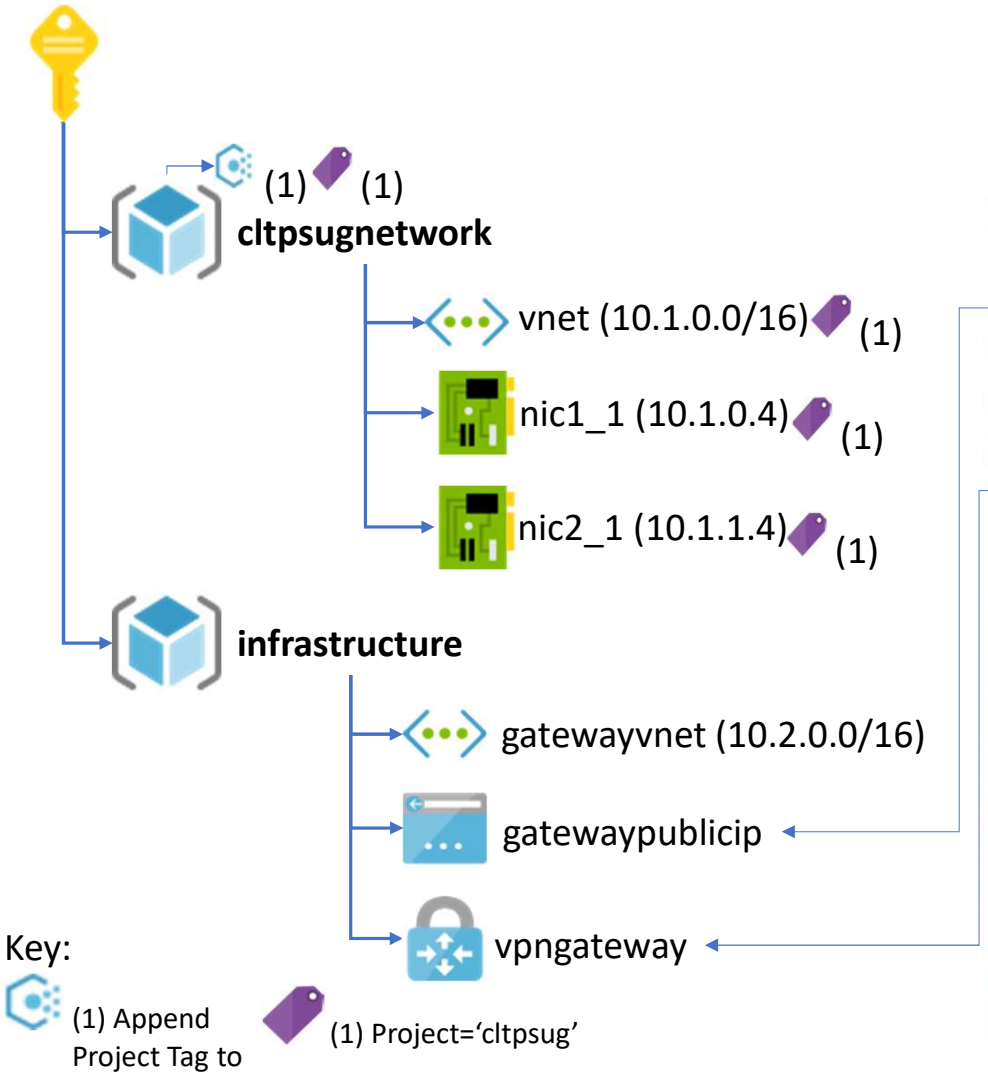
- Gateway must be “Routed”
- IKEv2, SSTP and OpenVPN are available
- Client Authentication can be by certificate or RADIUS
- Once the Gateway is configured, you can down load a pre-configured client to install

Public IP Resource Configuration

- Separate Azure Resources (dynamic or statically allocated)
- Completely disconnected from the Resource they are associated with by NAT
- Allow Inbound connections to be directly established to the Resource its associated with



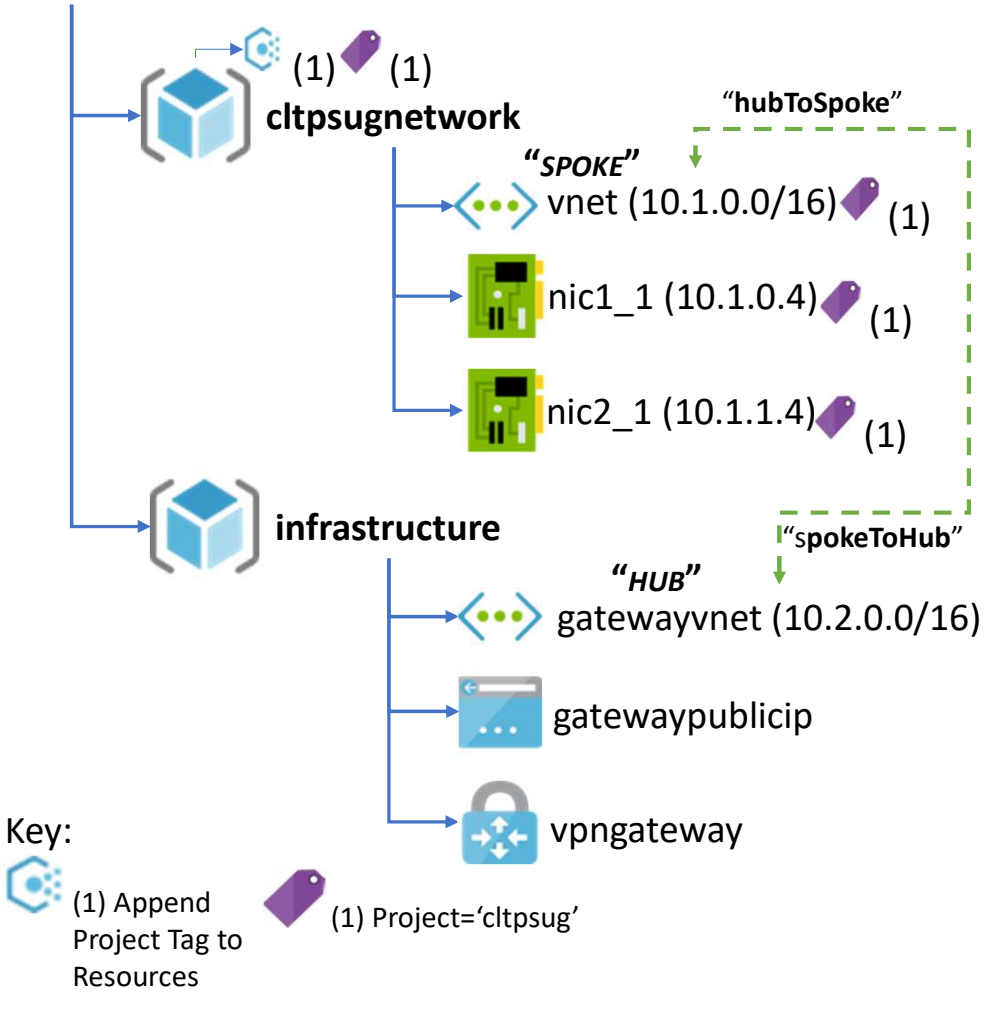
Deployment: Virtual Network Gateway/VNet Peering




```powershell
$resourceLocation = "eastus"
$infrastructureResourceGroupName = "infrastructure";
$gatewayVirtualNetworkAddressSpace = "10.2.0.0/16"
$gatewaySubnetAddrSpace = "10.2.0.0/28"

$gatewayVirtualNetworkCfg = @{
    Name = "gatewayvnet";
    ResourceGroupName = $infrastructureResourceGroupName;
    Location = $resourceLocation;
    AddressPrefix = $gatewayVirtualNetworkAddressSpace;
}

$gatewaySubnetCfg = @{Name = "GatewaySubnet"; AddressPrefix = $gatewaySubnetAddrSpace;}

$gatewayPublicIPCfg = @{
    Name = "gatewaypublicip";
    ResourceGroupName = $infrastructureResourceGroupName;
    Location = $resourceLocation;
    Sku = "Basic";
    AllocationMethod = "Dynamic";
}

$virtualNetworkGatewayCfg = @{
    Name = "vpngateway";
    ResourceGroupName = $infrastructureResourceGroupName;
    GatewayType = "Vpn";
    VpnType = "RouteBased";
    GatewaySku = "Basic";
    Location=$resourceLocation;
}

$vpnClientAddressPool = "172.16.1.0/24";
$vpnClientProtocol = "SSTP"
$rootCertificateCN = "CN=InfraGwayRoot"
$clientCertificateCN = "CN=MyLaptop"

#Get/Configure the Virtual Network
try { $gatewayVnet = Get-AzVirtualNetwork -Name $gatewayVirtualNetworkCfg['Name'] `
    -ResourceGroupName $infrastructureResourceGroupName -ErrorAction Stop }
catch { $gatewayVnet = New-AzVirtualNetwork @gatewayVirtualNetworkCfg }

#Get/Configure the Subnets
try { $gatewaySubnet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $gatewayVnet `
                       -Name $gatewaySubnetCfg['Name'] -ErrorAction Stop }
catch {
    Add-AzVirtualNetworkSubnetConfig -VirtualNetwork $gatewayVnet @gatewaySubnetCfg
    $gatewayVnet = ($gatewayVnet | Set-AzVirtualNetwork)
    $gatewaySubnet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $gatewayVnet -Name $gatewaySubnetCfg['Name'] }

#Get/Configure the PublicIP Resource
try { $gatewayPublicIP = Get-AzPublicIpAddress -Name $gatewayPublicIPCfg['Name'] `
    -ResourceGroupName $infrastructureResourceGroupName -ErrorAction Stop }
catch { $gatewayPublicIP = New-AzPublicIpAddress @gatewayPublicIPCfg }

#Get/Create the VPN Gateway
try { $vpngateway = Get-AzVirtualNetworkGateway -Name $virtualNetworkGatewayCfg['Name'] `
-ResourceGroupName $infrastructureResourceGroupName -ErrorAction Stop }
catch {$vpngatewayIpCfg = New-AzVirtualNetworkGatewayIpConfig -Name "vpngatewayipcfg" -Subnet $gatewaySubnet `
 -PublicIpAddress $gatewayPublicIP;
$vpngateway = New-AzVirtualNetworkGateway -IpConfigurations $vpngatewayIpCfg @virtualNetworkGatewayCfg}

#Configure the VPN Gateway for P2S, SSTP
$vpngateway = ($vpngateway | Set-AzVirtualNetworkGateway -VpnClientAddressPool $vpnClientAddressPool `
-VpnClientProtocol SSTP)
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateVPNGatewayAndVNet.ps1*

Deployment: Virtual Network Gateway/VNet Peering cont'd



Gateway Config cont'd

```
#Create Self-Signed Root and Client certificates to use Client Authentication
#Create the Self-Signed Root Certificate (which the Public Key is uploaded to the VPN Gateway as a "Root Certificate"
New-SelfSignedCertificate -Type Custom -KeySpec Signature -Subject $rootCertificateCN -KeyExportPolicy Exportable `
-HashAlgorithm sha256 -KeyLength 2048 -KeyUsageProperty Sign -KeyUsage CertSign -CertStoreLocation "Cert:\CurrentUser\My"

#Now use that Self-Signed Root Certificate to generated a Client Certificate that will be used to authenticate
#the local computer

$rootCertificate = Get-ChildItem -Path "Cert:\CurrentUser\My" | where {$_.Subject -eq $rootCertificateCN }
New-SelfSignedCertificate -Type Custom -KeySpec Signature -Subject $clientCertificateCN -KeyExportPolicy Exportable `
-HashAlgorithm sha256 -KeyLength 2048 -CertStoreLocation "Cert:\CurrentUser\My" -Signer $rootCertificate `
-TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.2")

#Upload the Root Certificate
$rootCertBase64 = [System.Convert]::ToBase64String($rootCertificate.RawData)
Add-AzVpnClientRootCertificate -VpnClientRootCertificateName "InfraGwayRoot" -PublicCertData $rootCertBase64 `
-VirtualNetworkGatewayName $virtualNetworkGatewayCfg['Name'] -ResourceGroupName $infrastructureResourceGroupName
```

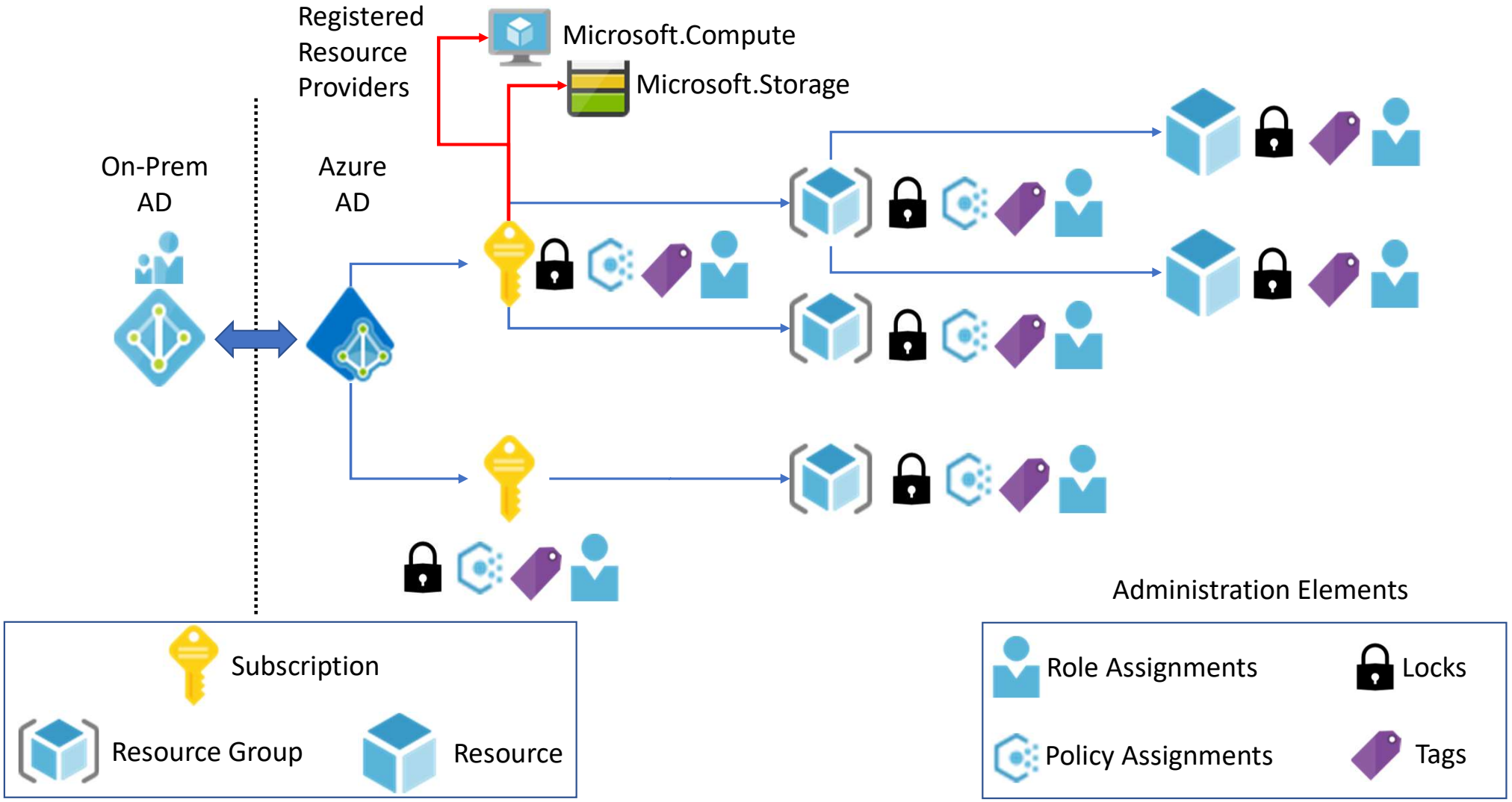
URI: *github.com/coderUT/cltpsug-july2019/Code/CreateVPNGatewayAndVNet.ps1*

VNet Peering

```
$resourceLocation = "eastus"

$infrastructureResourceGroupName = "infrastructure";
$hubVNetName = "gatewayvnet"

$networkResourceGroupName = "cltpsugnetwork";
$spokeVNetName = "vnet"

$hubVNet = Get-AzVirtualNetwork -Name $hubVNetName `
           -ResourceGroupName $infrastructureResourceGroupName
$spokeVNet = Get-AzVirtualNetwork -Name $spokeVNetName `
           -ResourceGroupName $networkResourceGroupName

#Create the Peering from "Hub" to "Spoke"
Add-AzVirtualNetworkPeering -Name "hubToSpoke" -VirtualNetwork $hubVNet `
    -RemoteVirtualNetworkId $spokeVNet.Id -AllowGatewayTransit


#Create the Peering from "Spoke" to "Hub"
Add-AzVirtualNetworkPeering -Name "spokeToHub" -VirtualNetwork $spokeVNet `
    -RemoteVirtualNetworkId $hubVNet.Id -UseRemoteGateways
```

URI: *github.com/coderUT/cltpsug-july2019/Code/CreateVNetPeerings.ps1*

Azure Resource Manager (ARM) and Governance



ARM: Example Resource Group Configuration

Engineering_Compute

- **Role Assignments**
  - Owner
    - Engineering Admins
    - IT Admins
  - Contributor
    - Engineering Users
- **Policy Assignments**
  - *Allowed resource locations*
    - East US, West US
  - *Allowed resource types*
    - Microsoft.Compute/*
  - *Allowed VM SKUs*
    - Standard_B2ms
  - Append resource group tags to resources
- **Tags**
  - Dept = Engineering
- **Locks**
  - Delete

Administrative Requirements

- Engineering “Users” CAN NOT delete this Resource Group
  - Has to be enforced at the Subscription-level
- All “Users” should only be able to create Compute (Microsoft.Compute) Resources inside this Resource Group
  - VM Resources are further Restricted by SKU (Standard_B2ms only)
- All “Users” should only be able to create Resources in regions “East US” or “West US”
- Any Resource created must be associated with Department “Engineering”
- Only Admins can Delete Resources from this Resource Group
  - I add this requirement because there is no analog to NTFS “CREATOR OWNER” in ARM
  - As a result, your Administrative Model will likely not be as broad as this contrived example (as far as the number of user’s with Access)
    - You may end up with many more Resource Groups with narrow Access grants
    - You may choose a different way to associate Resources (maybe by Application instead of Department and Resource Type)

ARM: Role Assignments/Security Model

Roles

- Collections of Permissions on Resource Provider Operations
- Built-In Roles
  - "Owner" vs "Contributor" vs "Reader"
- "Compute" Provider Operation Permissions for Built-In Role "Contributor"
  - **Microsoft.Compute/virtualMachines**
  - **Microsoft.Compute/disks**
- Assigned to User Identities at Subscription, Resource Group or Resource Scopes and can be inherited ("**Role Assignments**")
  - If a given User Identity was assigned a Role lacking "Read: Get Virtual Machines" at the Subscription level
    - They can't see ANY Virtual Machines in the entire Subscription

Virtual Machines

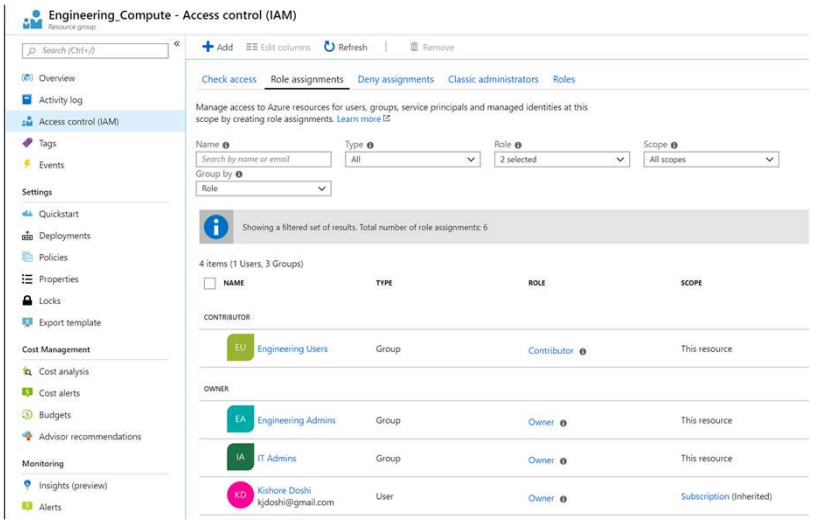
Permitted actions - Virtual Machine Contributor (preview)

MANAGEMENT ACTIONS	PERMISSIONS
Read: Get Virtual Machine	✓
Write: Create or Update Virtual Machine	✓
Delete: Delete Virtual Machine	✓
Other actions	
Capture Virtual Machine	✓
Convert Virtual Machine disks to Managed Disks	✓
Deallocate Virtual Machine	✓
Generalize Virtual Machine	✓
Perform Maintenance Redeploy	✓
Power Off Virtual Machine	✓
Redeploy Virtual Machine	✓
Reimage Virtual Machine	✓
Restart Virtual Machine	✓
Run Command on Virtual Machine	✓
Start Virtual Machine	✓

Microsoft.Compute/disks

Permitted actions - Contributor (preview)

MANAGEMENT ACTIONS	PERMISSIONS
Read: Get Disk	✓
Write: Create or Update Disk	✓
Delete: Delete Disk	✓
Other actions	
Get Disk SAS URI	✓
Revoke Disk SAS URI	✓




```powershell
$location = "eastus"
$engComputeRG = "Engineering_Compute"
$iTAdmins = Get-AzADGroup -DisplayName "IT Admins"
$engAdmins = Get-AzADGroup -DisplayName "Engineering Admins"
$engUsers = Get-AzADGroup -DisplayName "Engineering Users"

try { Get-AzResourceGroup -Name $engComputeRG }
catch { New-AzResourceGroup -Name $engComputeRG -Location $location }

#IT Admins are Owners
New-AzRoleAssignment -ObjectId $iTAdmins.Id -ResourceGroupName $engComputeRG `
-RoleDefinitionName "Owner"

#Engineering Admins are Owners
New-AzRoleAssignment -ObjectId $engAdmins.Id -ResourceGroupName $engComputeRG `
-RoleDefinitionName "Owner"

#Engineering Users are Contributors
New-AzRoleAssignment -ObjectId $engUsers.Id  -ResourceGroupName $engComputeRG `
-RoleDefinitionName "Contributor"
```

ARM: Policy Assignments

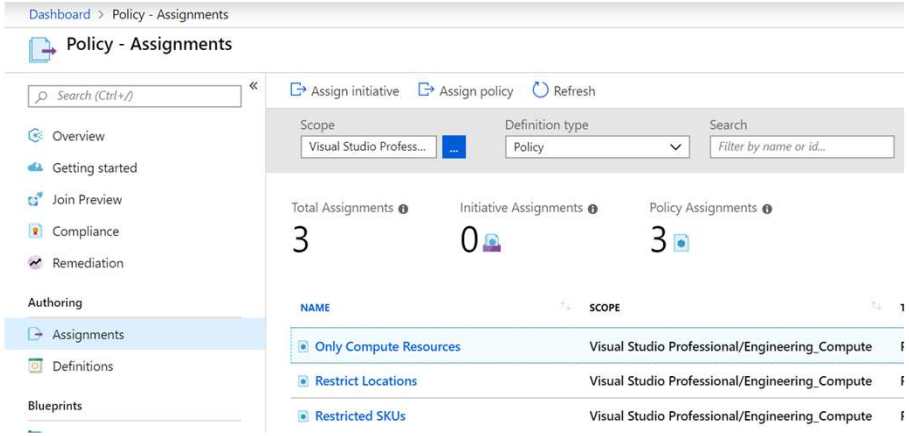
Policies

- Audit or restrict actions
- Assigned at Subscription or Resource Group levels
- Built-In Policy Definitions (many now)
  - Allowed Resource Types
  - Allowed Virtual Machine SKUs
  - Allowed Locations
- When a Policy is triggered, its noted in the Azure Activity Log

```
$location = "eastus"
$engComputeRG = "Engineering_Compute"
$providerNamespace = "Microsoft.Compute"
$allowedSkusList = @('Standard_B2ms')
$allowedLocationsList = @('East US','West US')
$resourceGroup = Get-AzResourceGroup -Name $engComputeRG

#Restrict Resource Locations
$allowedLocations = @{'listOfAllowedLocations'=$allowedLocationsList}

$policyDefinition = Get-AzPolicyDefinition -Builtin `
| where {$_.Properties.displayName -eq 'Allowed locations'}
try { Get-AzPolicyAssignment -Name "Restrict Locations" -Scope $resourceGroup.ResourceId -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name "Restrict Locations" `
-Scope $resourceGroup.ResourceId -PolicyParameterObject $allowedLocations `
-PolicyDefinition $policyDefinition }

#Restrict Resource Types to Microsoft.Compute
$computeResourceProvider = Get-AzResourceProvider -Location $location `
-ProviderNamespace $providerNamespace
$resourceTypes = $resourceProvider.ResourceTypes.ResourceTypeName | `
foreach {('{0}/{1}' -f $providerNamespace, $_)}
$allowedResourceTypes = @{'listOfResourceTypesAllowed'=@($resourceTypes)}

$policyDefinition = Get-AzPolicyDefinition -BuiltIn `
| where {$_.Properties.displayName -eq 'Allowed resource types'}
try { Get-AzPolicyAssignment -Name "Only Compute Resources" -Scope $resourceGroup.ResourceId -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name "Only Compute Resources" `
-Scope $resourceGroup.ResourceId -PolicyParameterObject $allowedResourceTypes `
-PolicyDefinition $policyDefinition }

#Restrict the Allowed VM SKUS
$allowedSkus = @{'listOfAllowedSKUs'=$allowedSkusList}

$policyDefinition = Get-AzPolicyDefinition -BuiltIn `
| where {$_.Properties.displayName -eq 'Allowed virtual machine SKUs'}
try { Get-AzPolicyAssignment -Name "Restricted SKUs" -Scope $resourceGroup.ResourceId -ErrorAction Stop}
catch { New-AzPolicyAssignment -Name "Restricted SKUs" `
-Scope $resourceGroup.ResourceId -PolicyParameterObject $allowedSkus `
-PolicyDefinition $policyDefinition }
```

```
{
  "properties": {
    "displayName": "Allowed resource types",
    "policyType": "BuiltIn",
    "mode": "Indexed",
    "description": "This policy enables you to specify the resource types that your organization can deploy.
    "metadata": {
      "category": "General"
    },
    "parameters": {
      "listOfResourceTypesAllowed": {
        "type": "Array",
        "metadata": {
          "description": "The list of resource types that can be deployed.",
          "displayName": "Allowed resource types",
          "strongType": "resourceTypes"
        }
      }
    },
    "policyRule": {
      "if": {
        "not": {
          "field": "type",
          "in": "[parameters('listOfResourceTypesAllowed')]"
        }
      },
      "then": {
        "effect": "deny"
      }
    }
  },
  "id": "/providers/Microsoft.Authorization/policyDefinitions/a08ec900-254a-4555-9bf5-e42af04b5c5c",
  "type": "Microsoft.Authorization/policyDefinitions",
  "name": "a08ec900-254a-4555-9bf5-e42af04b5c5c"
}
```

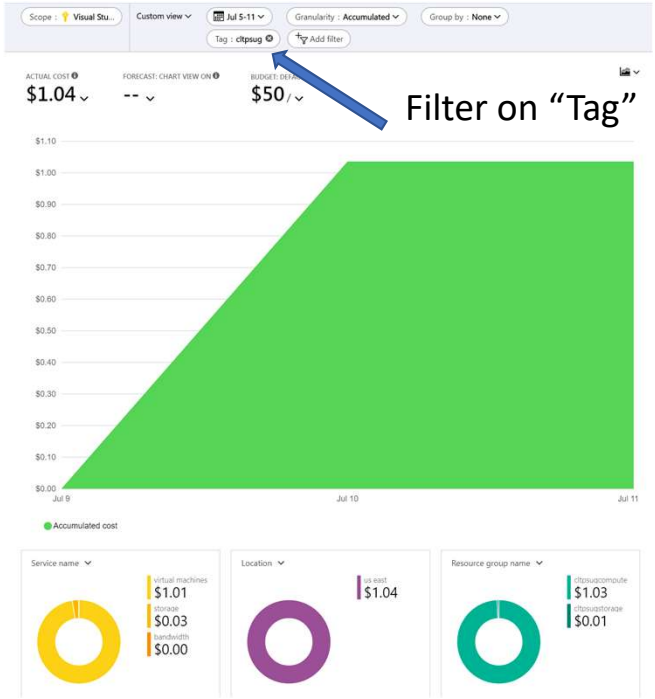
Policy Definition: Allowed Resource Types



ARM: Resource Tags

Tags

- Custom Name/Value Pairs
- Attach Business Metadata and provide logical organization for Resources
- Attach to a Resource Group and/or individual Resources
- Two Very Common Uses for Tags
  - Cost Management Tracking
  - Pin a Tag to your Portal Dashboard to quickly go to those Resources




```powershell
$engComputeRG = "Engineering_Compute"
$tagName = "Department"
$tagValue = "Engineering"
Set-AzResourceGroup -Name $engComputeRG -Tag @{$tagName=$tagValue}
$resourceGroup = Get-AzResourceGroup -Name $engComputeRG

#Add the Tag to all Resources within the Resource Group
$policyTag = @{'tagName'=$tagName}

$policyDefinition = Get-AzPolicyDefinition -BuiltIn `
| where {$_.Properties.displayName -eq 'Append tag and its value from the resource group'}

try { Get-AzPolicyAssignment -Name "Append Department Tag to Resources" `
-Scope $resourceGroup.ResourceId -ErrorAction Stop }
catch { New-AzPolicyAssignment -Name "Append Department Tag to Resources" `
-Scope $resourceGroup.ResourceId -PolicyParameterObject $policyTag `
-PolicyDefinition $policyDefinition }
```

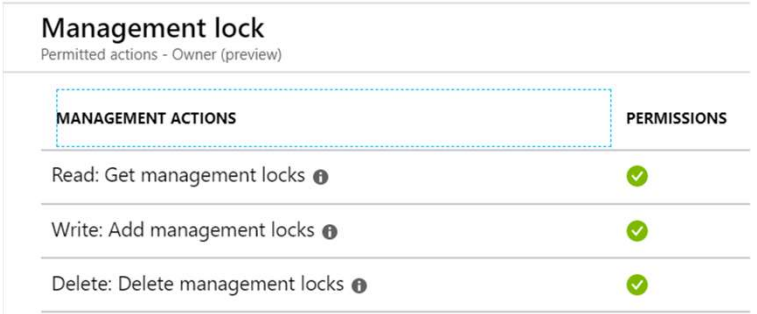
Real-Time Cost Tracking:

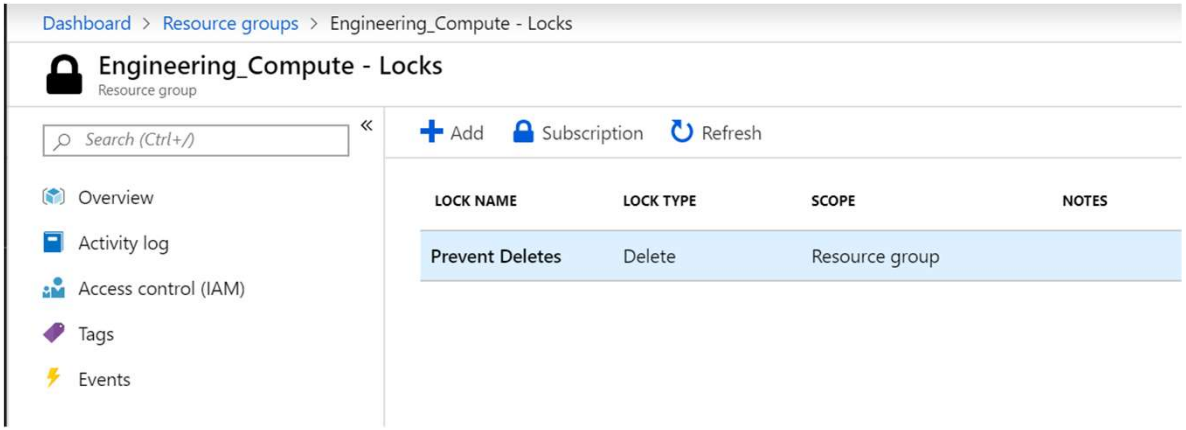
Portal => Cost Management + Billing => Cost Analysis

ARM: Resource Locks

Locks

- Applies to ALL User Identities regardless of Role Assignment(s)
- Restrict all management plane operations…until the Lock is removed
- Two Types of Locks
  - ReadOnly
  - Delete
- Assigned at the Resource Group or Resource level
  - Locks applied to a Resource Group are inherited to all contained Resources
- Removing a Lock requires a Role assignment with the **Microsoft.Authorization/locks** delete permission
  - "Contributor" Built-In Role DOES NOT have this permission
- Removing a Lock is noted in the Azure Activity Log






```
$engComputeRG = "Engineering_Compute"

try { Get-AzResourceLock -LockName "Prevent Deletes" `
                         -ResourceGroupName $engComputeRG `
                         -ErrorAction Stop }
catch { New-AzResourceLock -LockName "Prevent Deletes" `
                           -LockLevel CanNotDelete `
                           -ResourceGroupName $engComputeRg `
                           -Force }
```

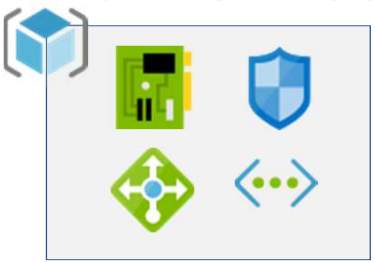
ARM: Administrative Model Example

Engineering_Compute

- **Role Assignments**
  - Owner
    - Engineering Admins
    - IT Admins
  - Contributor
    - Engineering Users
- **Policy Assignments**
  - *Allowed resource locations*
    - East US, West US
  - *Allowed resource types*
    - Microsoft.Compute/*
  - *Allowed VM SKUs*
    - Standard_B2ms
    - Standard_D2s_v3
  - Append resource group tags to resources
- **Tags**
  - Dept = Engineering
- **Locks**
  - Delete

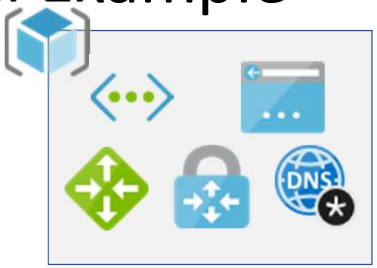
Engineering_Storage

- **Role Assignments**
  - Owner
    - Engineering Admins
    - IT Admins
  - Contributor
    - Engineering Users
- **Policy Assignments**
  - *Allowed resource locations*
    - East US, West US
  - *Allowed resource types*
    - Microsoft.Storage/*
  - *Append resource group tags to resources*
- **Tags**
  - Dept = Engineering
- **Locks**
  - Delete



Engineering_Network

- **Role Assignments**
  - Owner
    - Engineering Admins
    - IT Admins
  - Contributor
    - **Engineering Networking User**
- **Policy Assignments**
  - *Allowed resource locations*
    - East US, West US
  - *Allowed resource types*
    - NI, NSG, Vnets, LBs
  - *Append resource group tags to resources*
- **Tags**
  - Dept = Engineering
- **Locks**
  - Delete



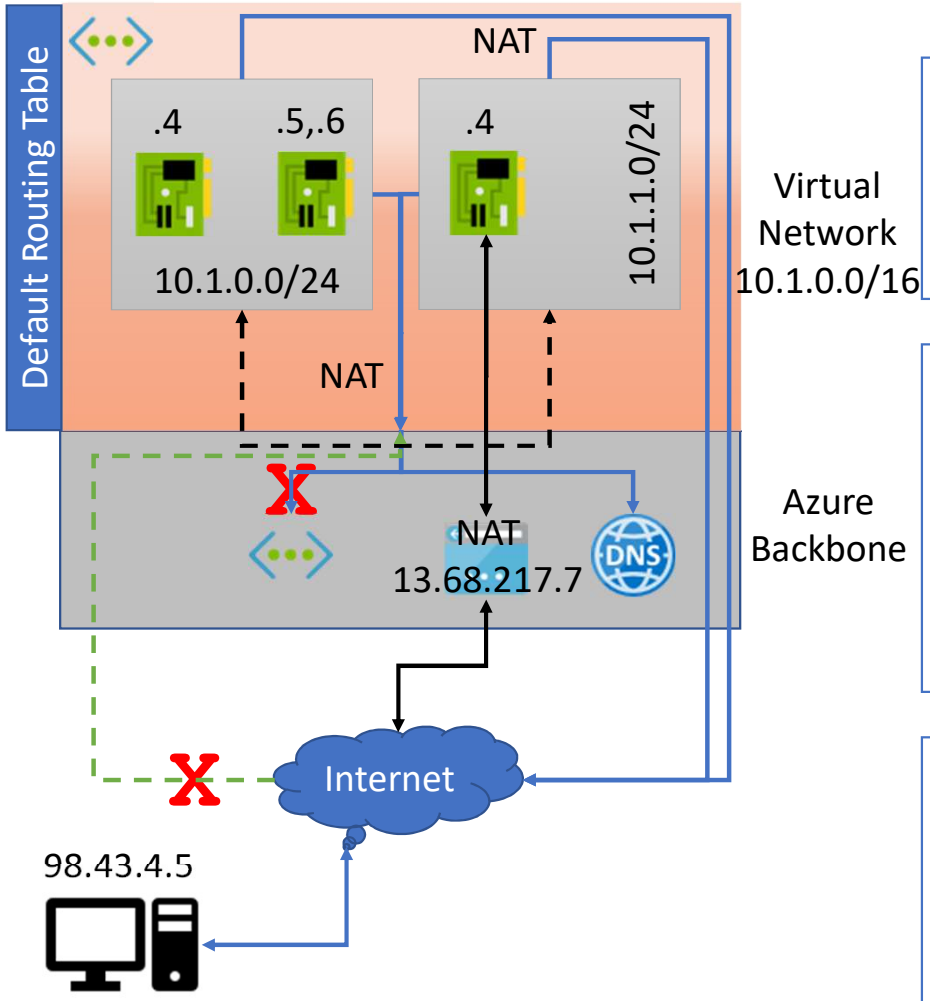
IT_Shared

- **Role Assignments**
  - Owner
    - IT Admins
- **Policy Assignments**
  - *Allowed resource locations*
    - East US, West US
  - *Append resource group tags to resources*
- **Tags**
- **Locks**
  - Read Only

Subscription

- **Role Assignments**
  - Owner
    - IT Admins
- **Policy Assignments**
- **Tags**
- **Locks**

NetworkingProfile – Public IPs Addresses



Virtual Network 10.1.0.0/16

Azure Backbone

Public IP Address (Azure Resource)

- Separate Azure Resource for billing and management purposes
- Attached to at most 1 IP configuration on a Network Interface
  - I would not do this without also attaching a Network Security Group
- Traffic destined for the Public IP Resource is NAT translated to the private IP address of the configuration its attached to

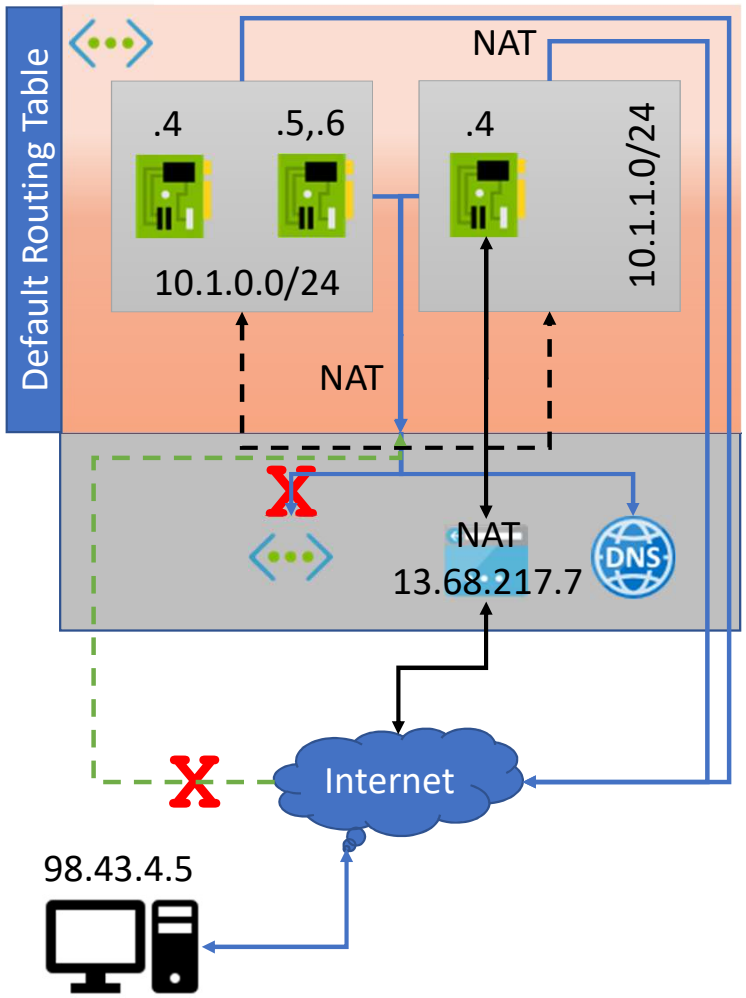
Allocation and SKUs

- Allocation can be Dynamic or static
- Basic SKU
  - Supports Dynamic or Static allocation
- Standard SKU
  - Static allocation only
  - Requires a Network Security Group on the Network Interface or containing Subnet to “whitelist” inbound traffic
  - Zone redundant OR Zonal

DNS

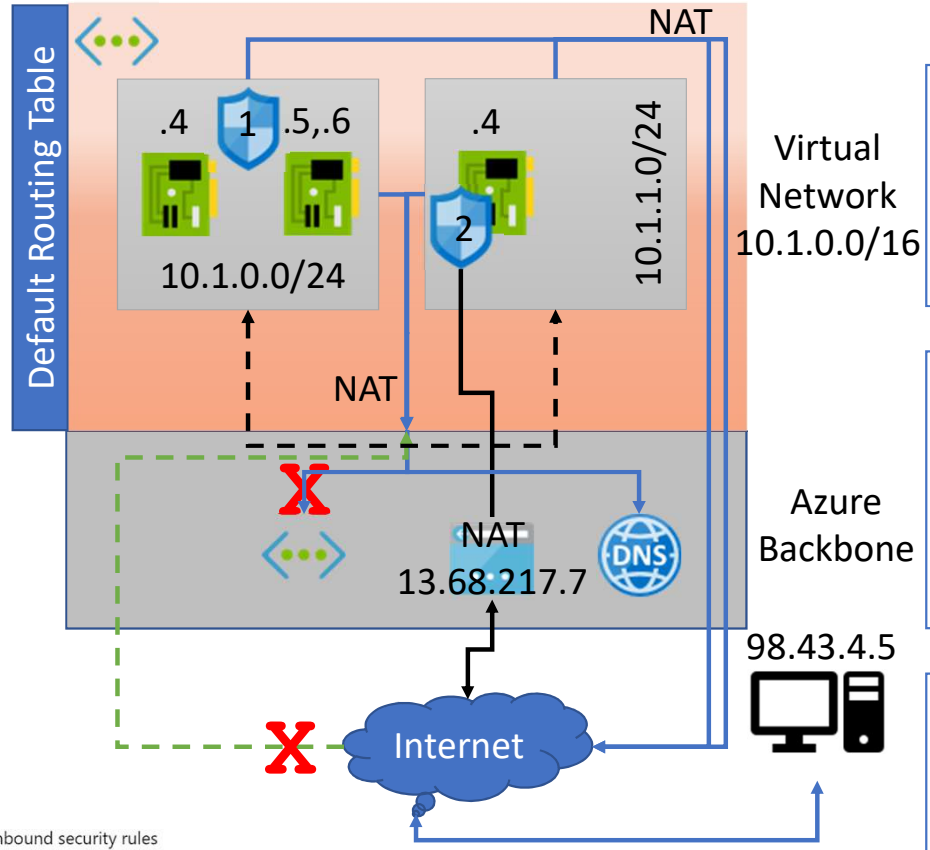
- Can designate DNS name for a Public IP resource
  - That name stays with the Public IP resource if its Network Interface association is changed
- Name Suffix is <region>.cloudapp.azure.com
  - E.g., eastus.cloudapp.azure.com
  - Since Public IP resources are regional, best practice is to create a CNAME alias for your VM public DNS name

NetworkingProfile – Public IPs Addresses




```
97  #Configure a Public IP Address Resource for the 10.1.1.4 Network Interface (Subnet 2)
98  $publicIPCfg = @{
99      Name = "cltpsugpublicip_nic2_1";
100     ResourceGroupName = $resourceGroupName;
101     Location = $resourceLocation;
102     Sku = "Basic";
103     AllocationMethod = "Dynamic";
104 }
105
106 #Create the Public IP Resource
107 try { $publicIP = Get-AzPublicIpAddress -Name $publicIPCfg['Name'] -ResourceGroupName $resourceGroupName `
108         -ErrorAction Stop }
109 catch {$publicIP = New-AzPublicIpAddress @publicIPCfg }
110
111 #Bind it to the 10.1.1.4 Network Interface
112 #We'll overwrite an existing binding if there is one
113 $nic2_1ipcfg1 = Get-AzNetworkInterfaceIpConfig -Name $subnet2Nic1IPCfg1['Name'] -NetworkInterface $nic2_1
114 $nic2_1ipcfg1.PublicIpAddress = $publicIP
115 $nic2_1 = ($nic2_1 | Set-AzNetworkInterface)
```

NetworkingProfile – Network Security Groups



Inbound security rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
101	AllowHomeInbound	Any	Any	98.43.4.5	10.1.1.4	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBou...	Any	Any	AzureLoadBala...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Network Security Groups (Azure Resource)

- Associated to a Subnet OR Network Interface
- Basic, stateful "Traffic Filtering"
- If a Network Interface has a Public IP Attached
  - It is NAT translated to the Private IP BEFORE Inbound rules are applied
  - It is NAT translated to the Public IP AFTER Outbound rules are applied

Default Security Rules

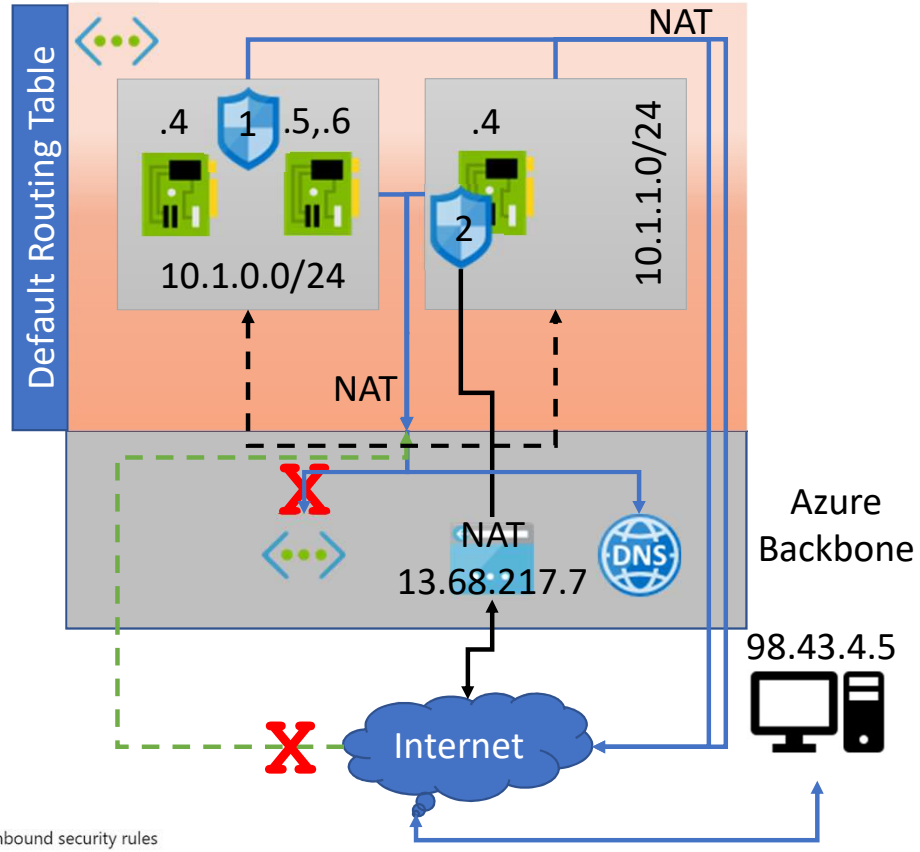
- Allow Inbound traffic from VNet, DHCP, DNS, health monitoring
  - Deny all other Inbound Traffic
- Allow Outbound traffic to VNet, Internet
  - Deny all other Outbound Traffic
- Can't disable Default Rules, but can create Custom rules with higher priority to "override"

Custom Security Rules

- Combination of Source/Destination (Address & Port Range), Protocol, Access, Direction and Priority
- "Service Tags" can be used to simplify Rules
  - Predefined tags mapped to groups of Address Prefixes maintained by Azure
  - "VirtualNetwork" maps to the address space of the Virtual Network
  - "Storage" maps to the Address Prefixes for Azure Storage
- Rule to allow Inbound Access from a specific Computer
  - Source: 98.43.4.5/*, Destination: 10.1.1.4/*, Protocol *, Priority 101

NetworkingProfile – Network Security Groups

Virtual Network
10.1.0.0/16



Inbound security rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
101	AllowHomeInbound	Any	Any	98.43.4.5	10.1.1.4	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBou...	Any	Any	AzureLoadBala...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

```
#Create the Network Security Group Resource for Subnet1 (NSG 1)
$subnet1_nsgCfg = @{
    Name = "cltpsugnsg_subnet1";
    ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
}

#Create the Network Security Group
try { $subnet1_nsg = Get-AzNetworkSecurityGroup -Name $subnet1_nsgCfg['Name'] `
                     -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch { $subnet1_nsg = New-AzNetworkSecurityGroup @subnet1_nsgCfg }

#Bind the Network Security Group to Subnet1
#We'll overwrite an existing binding if there is one
$subnet1.NetworkSecurityGroup = $subnet1_nsg
$vnet = ($vnet | Set-AzVirtualNetwork)
$subnet1 = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name $subnet1Cfg['Name']

#Create the Network Security Group Resource for the 10.1.1.4 Network Interface (NSG 2)
#We will have a rule that will restrict inbound access to just a specific IP address
$nic2_1_nsgCfg = @{
    Name = "cltpsugnsg_nic2_1";
    ResourceGroupName = $resourceGroupName;
    Location = $resourceLocation;
}

#Configure the Security Rule, the destination is the Private IP address because
#NAT translation takes place before NSG evaluation on Inbound traffic
$manageInboundAccessRule = @{
    Name = "AllowHomeInbound";
    Protocol = "*";
    Access = "Allow";
    Direction = "Inbound";
    Priority = "101";
    SourceAddressPrefix = "98.43.4.5";
    SourcePortRange = "*";
    DestinationAddressPrefix = $nic2_1.IpConfigurations[0].PrivateIpAddress;
    DestinationPortRange = "*";
}

#Create the Network Security Group
try { $nic2_1_nsg = Get-AzNetworkSecurityGroup -Name $nic2_1_nsgCfg['Name'] `
          -ResourceGroupName $resourceGroupName -ErrorAction Stop }
catch {
    $securityRule = New-AzNetworkSecurityRuleConfig @manageInboundAccessRule
    $nic2_1_nsg = New-AzNetworkSecurityGroup @nic2_1_nsgCfg -SecurityRules $securityRule
}

#Bind the Network Security Group to the 10.1.1.4 Network Interface
#We'll overwrite an existing binding if there is one
$nic2_1.NetworkSecurityGroup = $nic2_1_nsg
$nic2_1 = ($nic2_1 | Set-AzNetworkInterface)
```