AzureNestedHyperV

Create a Nested HyperV Host in Azure with one click

Plan of Attack

After thinking about this for a while, I came up with a plan of attack for this automation:

This needs to be in an ARM template. We need to provision a Virtual Network, version 3 VM with SSD disks and a public IP address. We will call the automation from the template and leverage the ability to have PowerShell act on our VM during the provisioning process using VM extensions.

Now during the provisioning, we are going to need a two-step automation process from a scripting perspective. We need both a PowerShell script and a PowerShell Desired State Configuration (DSC) to get this going without any human interaction. Remember our goal was to click on a button or execute one command and have the Host up with a running VM.

First, we need to use the PowerShell Custom Script extension to Install the Hyper-V Features and to import the xHyper-V DSC modules which allow us to configure Hyper-V. This script will then call a reboot. We have to do the reboot because the Hyper-V feature requires a reboot before we can call any of the PowerShell cmdlets.

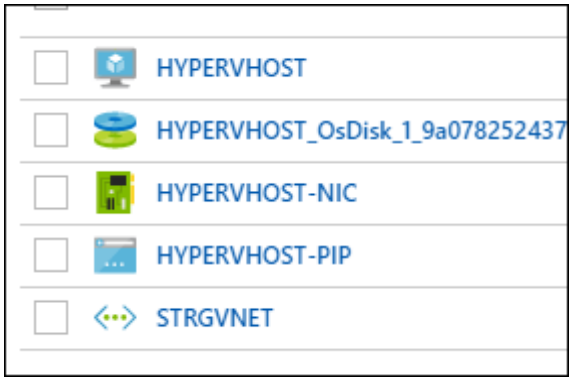
Once the VM reboots, then we will call the PowerShell DSC Extension to configure the Hyper-V Host. We will use the abilities of the DSC modules to create an Internal Switch. Once this is completed then the script will create a new IP address and a network NAT to pass the traffic from the Nested VMs out the NIC of the Azure HOST VM. Once this infrastructure is in place, all we need is the VM which can be downloaded from a blob storage from somewhere on the internet. After unzipping that file and placing in on the Host storage we will then simply run the New-VM Hyper-V Cmdlet. BAM – WE have a nested VM!

Artifacts Review
So, let's dig into what makes this all work. It's pretty amazing, but really there are only three text files that we need to make all of this work.

- **Azure-deploy.json** – ARM template which will be called to run this provisioning
- **InstallHyperV.ps1** – PowerShell script which will be called first and run using the PowerShell Custom Script extension
- **HyperVHostConfig.ps1** (actually packaged as a zip file) – this is the PowerShell DSC script which will be run using the PowerShell DSC Script Extension.

Azure-Deploy.json
This ARM template will create the following resources

- Virtual Machine: HYPERVHOST
- Premium Disk: HYPERVHOST_OsDisk
- Network Interface Card: HYPERVHOST-NIC
- Public IP Address: HYPERVHOST-PIP
- Virtual Network: STRGVNET



Resources created by the ARM template

Here we see the variables section which gives you an idea of how the scripts will be called later in the resources section of the ARM template. Notice how the PowerShell script and the DSC Zip file are called using a URL to the GitHub repo. This could be any URL that Azure can reach on the Web or even Azure Blog Storage if the permissions are set properly.

```
"variables": {
  "STRGVNETPrefix": "10.0.0.0/16",
  "STRGVNETSubnet1Name": "VMHOST",
  "STRGVNETSubnet1Prefix": "10.0.0.0/24",
  "HyperVHostName": "HYPERVHOST",
  "HyperVHostAdminUserName": "stormtrooper",
  "HyperVHostAdminPassword": "Password.1!!",
  "HyperVHostImagePublisher": "MicrosoftWindowsServer",
  "HyperVHostImageOffer": "WindowsServer",
  "HyperVHostWindowsOSVersion": "2016-Datacenter",
  "HyperVHostOSDiskName": "[concat(variables('HyperVHostName'), '-OSDISK')]",
  "HyperVHostVmSize": "Standard_D2s_v3",
  "HyperVHostVnetID": "[resourceId('Microsoft.Network/virtualNetworks', 'STRG
  "HyperVHostSubnetRef": "[concat(variables('HyperVHostVnetID'), '/subnets/',
  "HyperVHostNicName": "[concat(variables('HyperVHostName'), '-NIC')]",
  "HyperVHost-PUBIPName": "[concat(variables('HyperVHostName'), '-PIP')]",
  "HyperVHostConfigArchiveFolder": ".",
  "HyperVHostConfigArchiveFileName": "HyperVHostConfig.zip",
  "HyperVHostConfigURL": "https://github.com/deltadan/AzureNestedHyperV/blob/
  "HyperVHostInstallHyperVScriptFolder": ".",
  "HyperVHostInstallHyperVScriptFileName": "InstallHyperV.ps1",
  "HyperVHostInstallHyperVURL": "https://raw.githubusercontent.com/deltadan/A
```

Variables section of the ARM Template

InstallHyperV.ps1

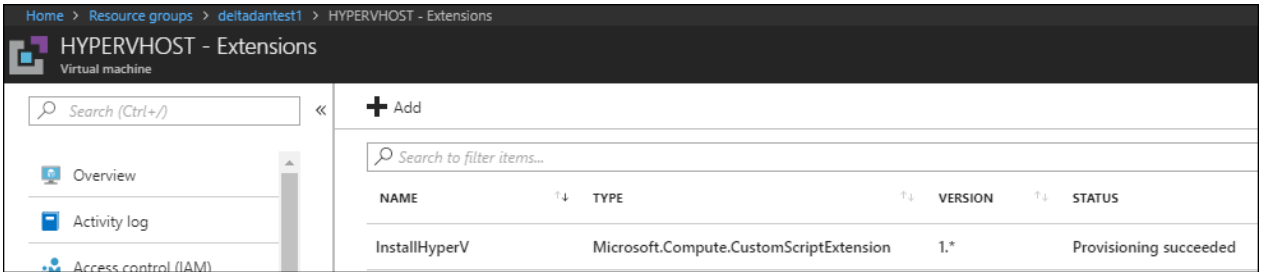
Next, let's look at the code in the PowerShell script. The first part will install the NuGet package manager and then install the xHyper-V modules that will be run later by the DSC script. After this is installed, then the Hyper-V Windows Feature is added using the Install-WindowsFeature cmdlets and then, of course, the Restart.

```
Set-ExecutionPolicy Unrestricted -Force
Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force
Find-Module -Includes DscResource -Name xHyper-v | Install-Module -Force

#Install Hyper-V and Reboot
Install-WindowsFeature -Name Hyper-V `
                       -IncludeAllSubFeature `
                       -IncludeManagementTools `
                       -Verbose `
                       -Restart
```

Code from the InstallHyperV.ps1 which is executed by the Custom Script Extention

If you watch the deployment progress within the Azure Portal you will see the Custom script and DSC extension showing as running, transitioning and then (hopefully), Provisioning Succeeded.



Viewing the scripts as they run in the Azure Portal

The beauty of this is that even though the machine has to perform a reboot of the OS, Azure will retain the context of this provisioning process and move on to the DSC script.

HyperVHostConfig.ps1

When the machine comes back up Azure will next install and run the DSC file which is inside of the Zip file. DSC is a different language than normal PowerShell, but you can also run PowerShell script and call cmdlets from within by using the right strategy. So, first here we see how the xHyper-V modules that were installed in the first script are called to configure Hyper-V.

```
Configuration Main
{
    Param ( [string] $nodeName )

    Import-DscResource -ModuleName 'PSDesiredStateConfiguration', 'xHyper-V'

    node $nodeName
    {
        # Ensures a VM with default settings
        xVMSwitch InternalSwitch
        {
            Ensure         = 'Present'
            Name           = 'Nat Switch'
            Type           = 'Internal'
        }

        Script ConfigureHyperV
        {
            GetScript =
            {
                @{Result = "ConfigureHyperV"}
            }

            TestScript =
            {
                return $false
            }
```

PowerShell DSC script which configures the Hyper-V Virtual Switch

After this section has run we next see how the SetScript section will run where we can configure the New NAT Switch, adding a new IP Address and then a NAT. Again, this is what will route our packets from any Nested VMs that are connected to this switch in Hyper-V.

```
SetScript =
{

    $NatSwitch = Get-NetAdapter -Name "vEthernet (NAT Switch)"
    New-NetIPAddress -IPAddress 192.168.0.1 -PrefixLength 24 -InterfaceIndex $NatSwitch.ifIndex
    New-NetNat -Name NestedVMNATnetwork -InternalIPInterfaceAddressPrefix 192.168.0.0/24 -Verbose
```

SetScript which creates an Internal IP Address and the NAT which will translate IP traffic from the Nested VM to the NIC of the Azure VM

Once this is complete, then you are ready to rock!
Now, I commented out the rest of the script in the GitHub repo for the automated deployment of the VM. You can easily create a VM using Hyper-V and then zip the file up and place it in a blob to be auto-provisioned. *Just review the script and you will be able to figure it out.* The tricky part is getting the folder structure correct in your zip file for the VHD, so watch out for that. ***Note, also that the IP Address for your Nested VMs will need to be on the 192.168.0.0/24 network.***

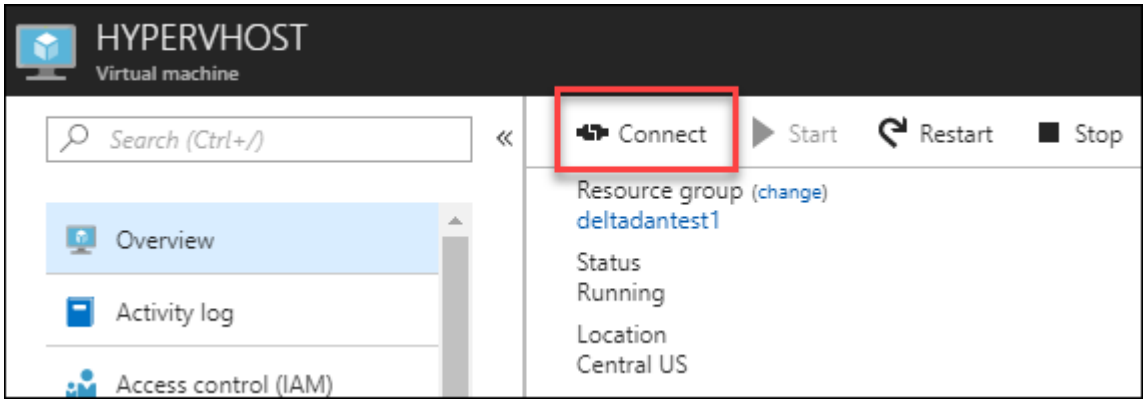
```
#               $zipDownload = "http://YOUR-URL-HERE/FILENAME.ZIP"
#               $downloadedFile = "D:\FILENAME.zip"
#               $vmFolder = "C:\VM"
#               Invoke-WebRequest $zipDownload -OutFile $downloadedFile
#               Add-Type -assembly "system.io.compression.filesystem"
#               [io.compression.zipfile]::ExtractToDirectory($downloadedFile, $vmFolder)
#               New-VM -Name VMNAME `
#                      -MemoryStartupBytes 2GB `
#                      -BootDevice VHD `
#                      -VHDPath 'C:\VM\PATH\OnPremLinuxVM.vhdx' `
#                      -Path 'C:\VM\PATH' `
#                      -Generation 1 `
#                      -Switch "NAT Switch"
```

Leverage this commented code to build your automated one-click Nested VM.

Provision HYPERVHOST and then a Nested VM
To deploy the template, click the deploy button above or over at the Repo. This will bring you to the Azure Portal where you can update the DNS Name for your host and it will configure the HYPERVHOST for you!
Ok, now its time to build a Nested VM. Connect to HYPERVHOST using the Azure Portal. The username and password are hardcoded in this version of the template:

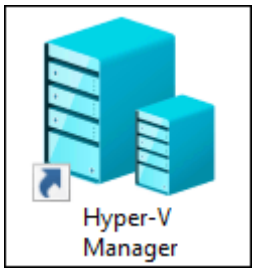
User: stormtrooperio

Password: Password.!!



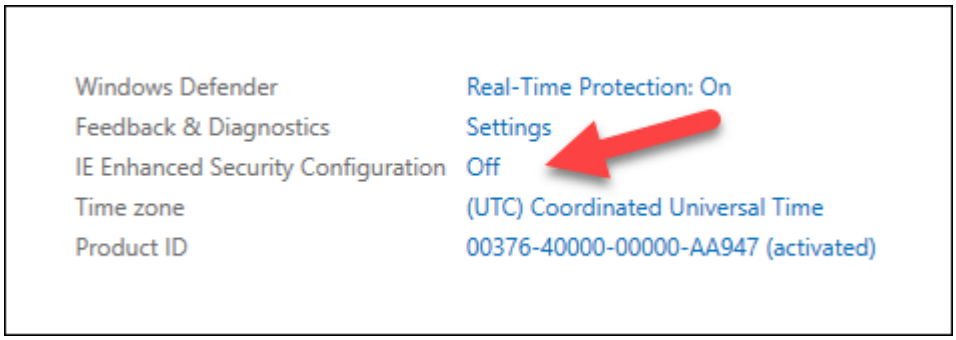
Click Connect to start a remote desktop to HYPERVHOST

After you are connected, click Start and then Windows Administrative Tools, Hyper-V Manager.



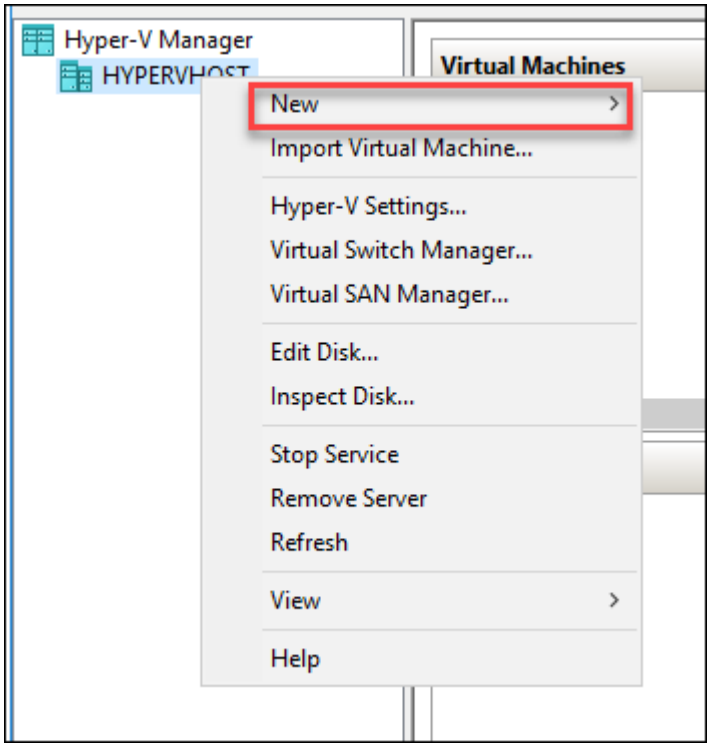
HyperV Manager on the HYPERVHOST Azure VM

Once you are in Hyper-V Manager you can create VMs. A quick way to get started would be to download the Ubuntu Server ISO from https://www.ubuntu.com/download/server/thank-you?version=17.10.1&architecture=amd64 from within the HOST.
You will need to turn off the IE Enhanced Security Configuration from Server Manager.



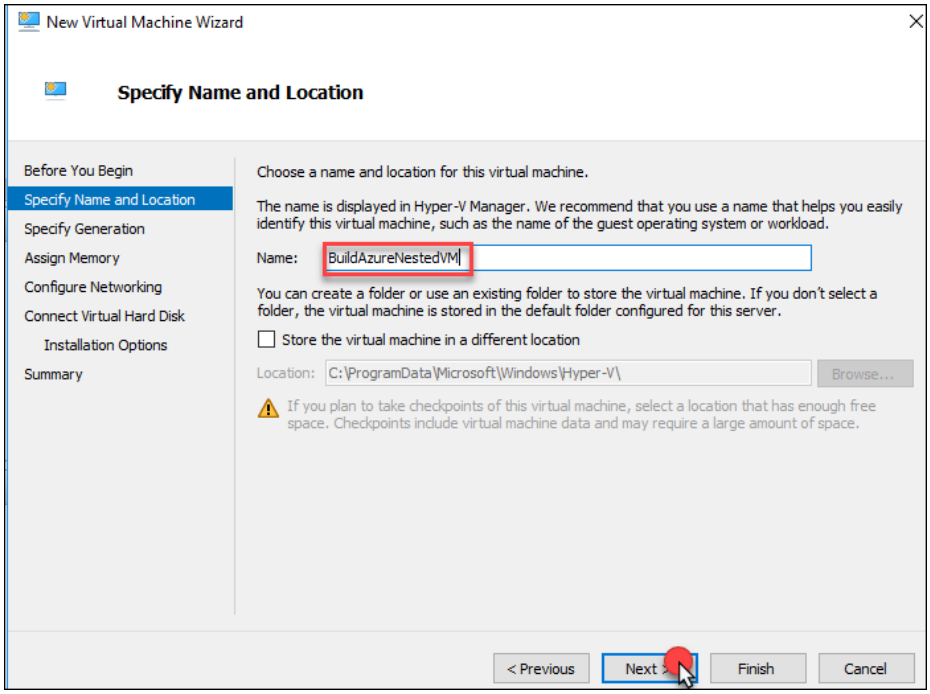
Turn of the IE Enhanced Security

After you have the ISO downloaded use these steps to get it up and running using Hyper-V Manager. First, right-click HYPERVHOST and then click New, Virtual Machine.



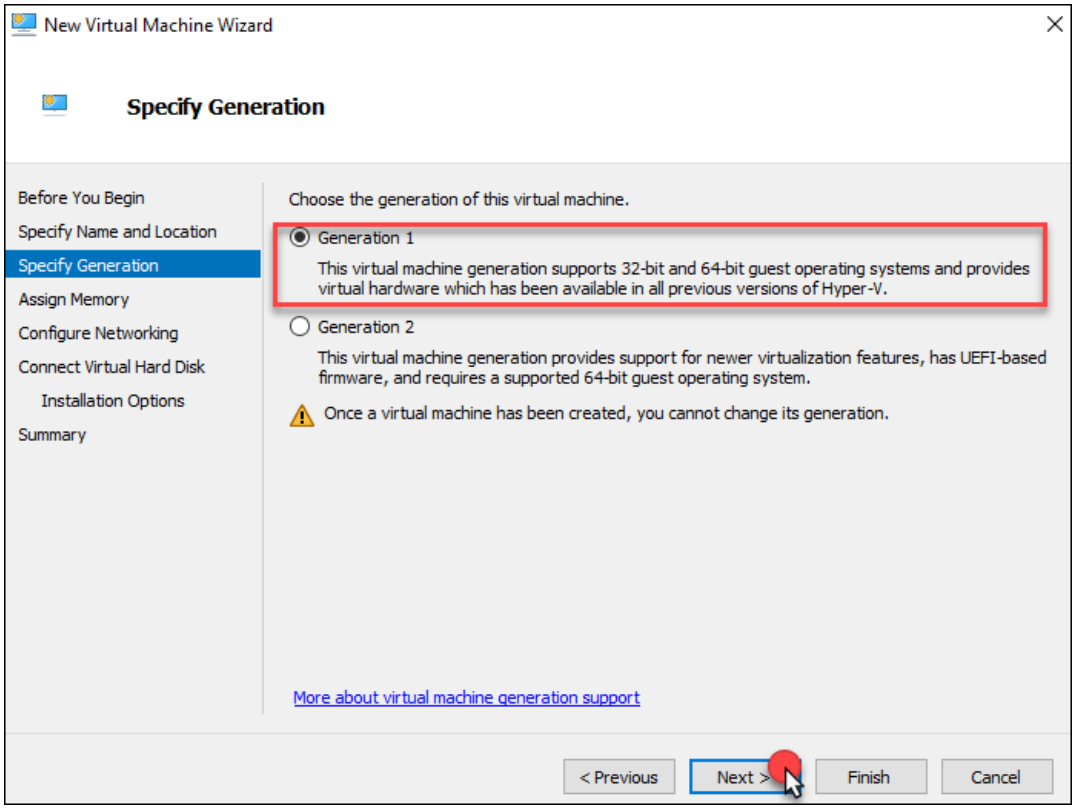
Create your first Nested VM using HyperV Manager

Click Next, and then Provide a Name for the VM and a Location on HYPERVHOST file system for your new VM.



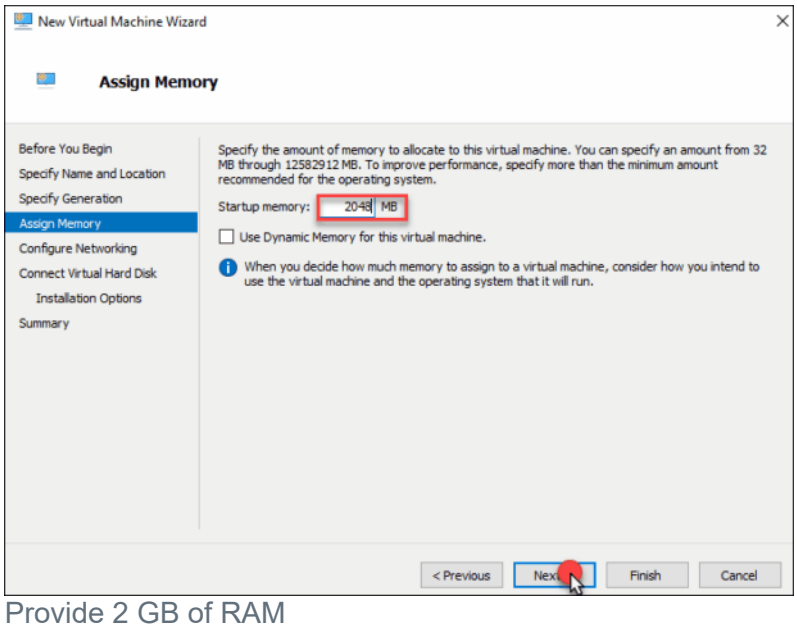
Provide a Name for the Nested VM

Stick with Generation 1 VM.



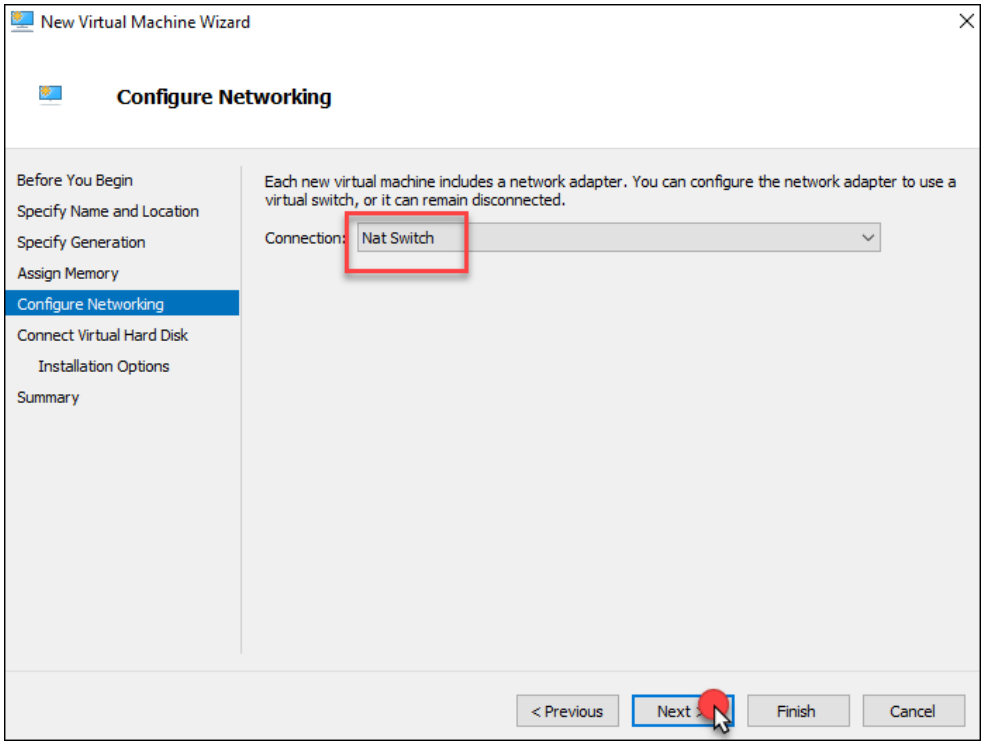
Select Generation 1

Give the VM 2 GB of RAM.



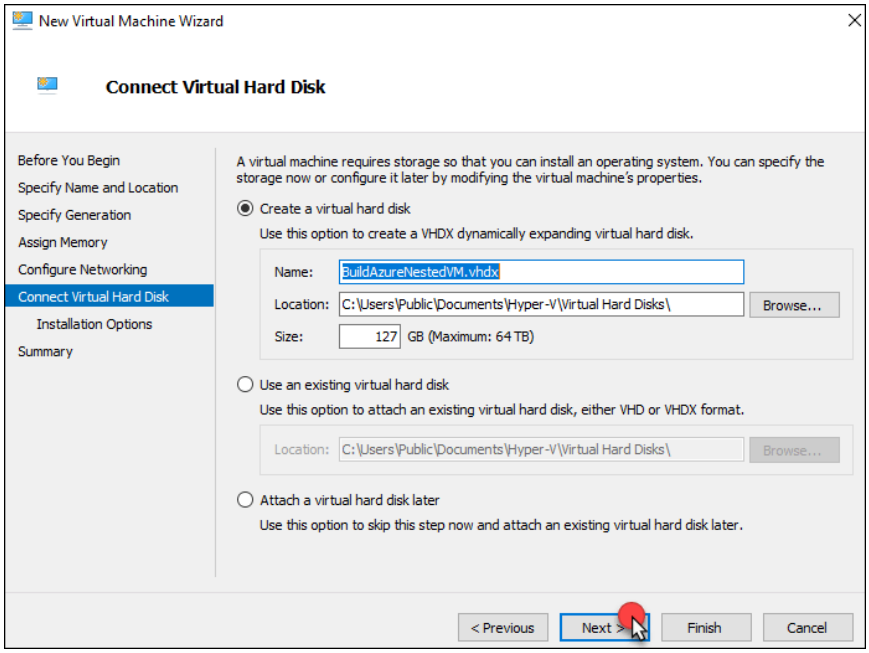
Provide 2 GB of RAM

Select the NAT Switch. Remember this was created by the PowerShell DSC xHyper-V Module.



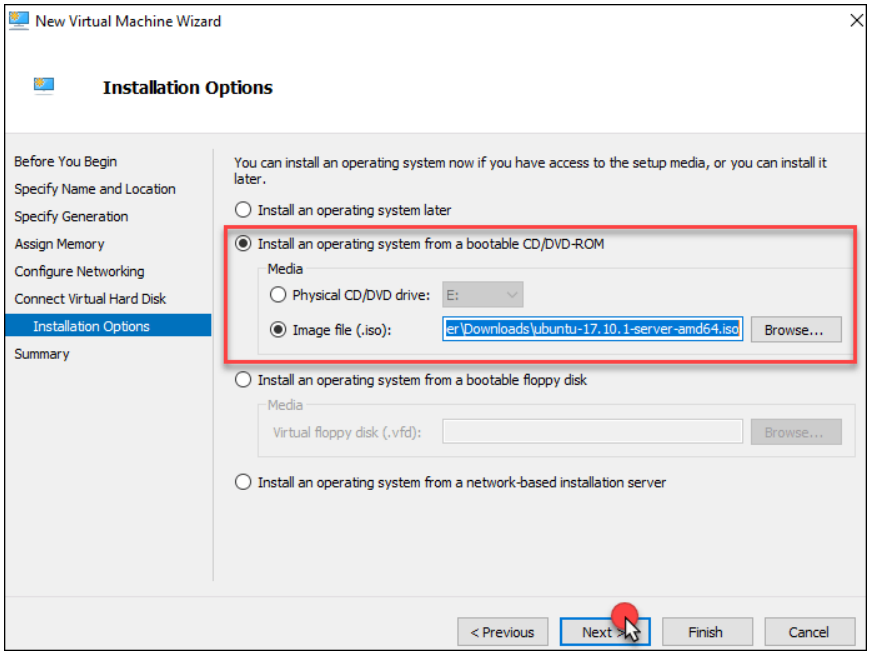
Select the NAT Switch.

Configure your Disk settings.



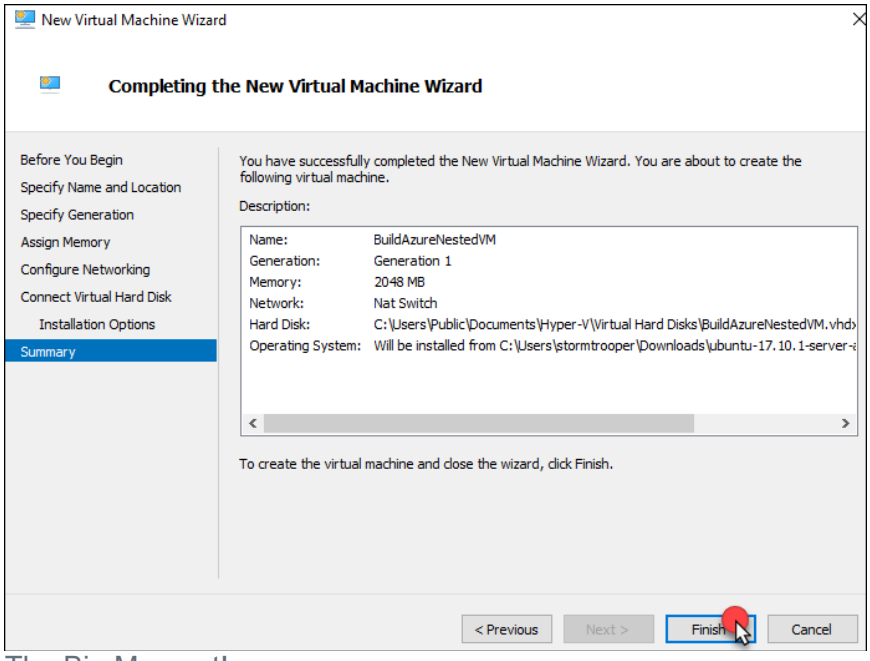
Disk settings for the Nested VM

Select the Install an operating system from bootable CD/DVD-ROM and then select the ISO you downloaded. Make sure to chose the file from your download folders.



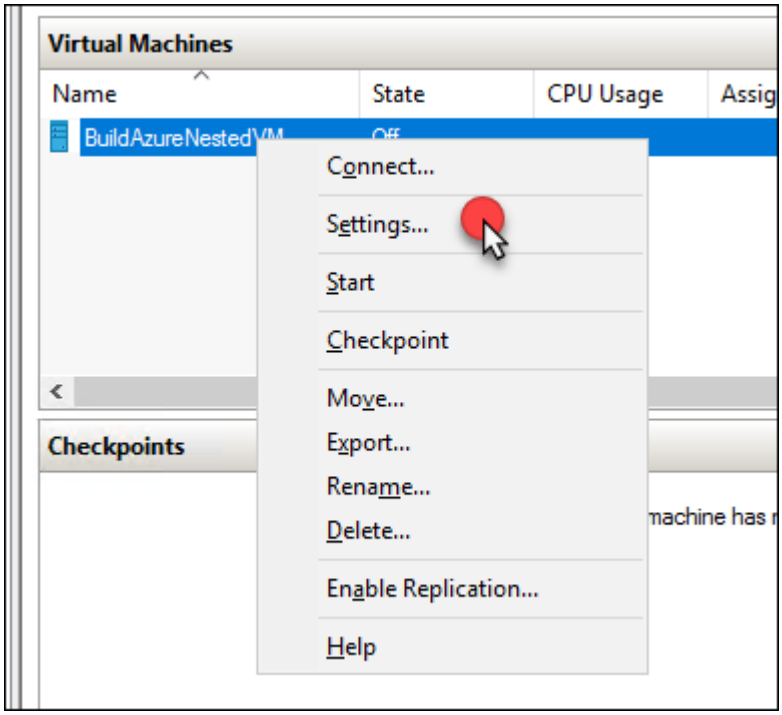
Connect the ISO to the DVD drive of the Nested VM

Click Finish!



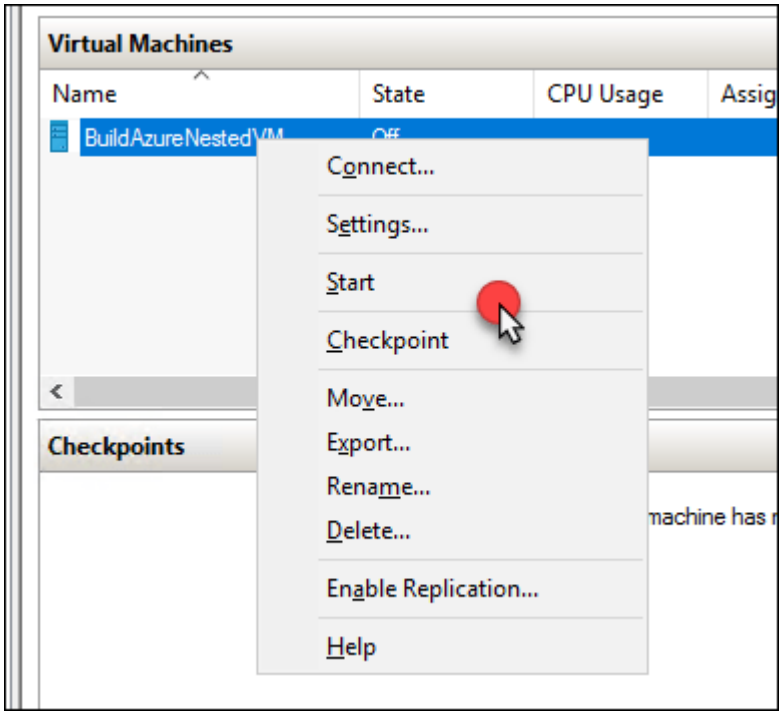
The Big Moment!

Your VM will now show in the list on the HYPERVHOST. Right-click the file and then click Settings. Take a look at the settings, but there are no required changes.



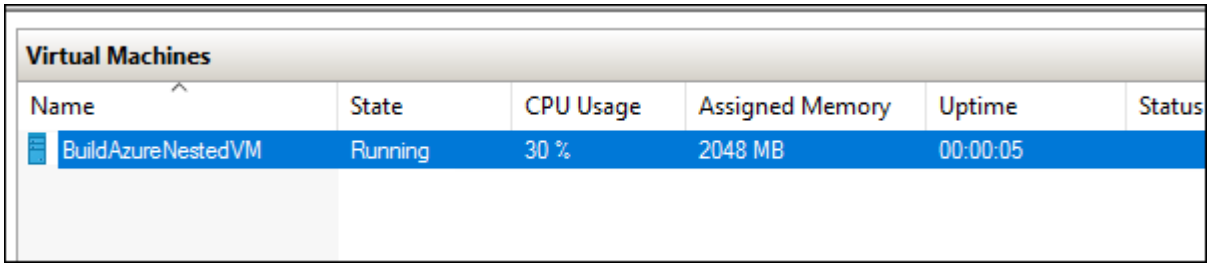
Click settings on the Nested VM

Start the VM.



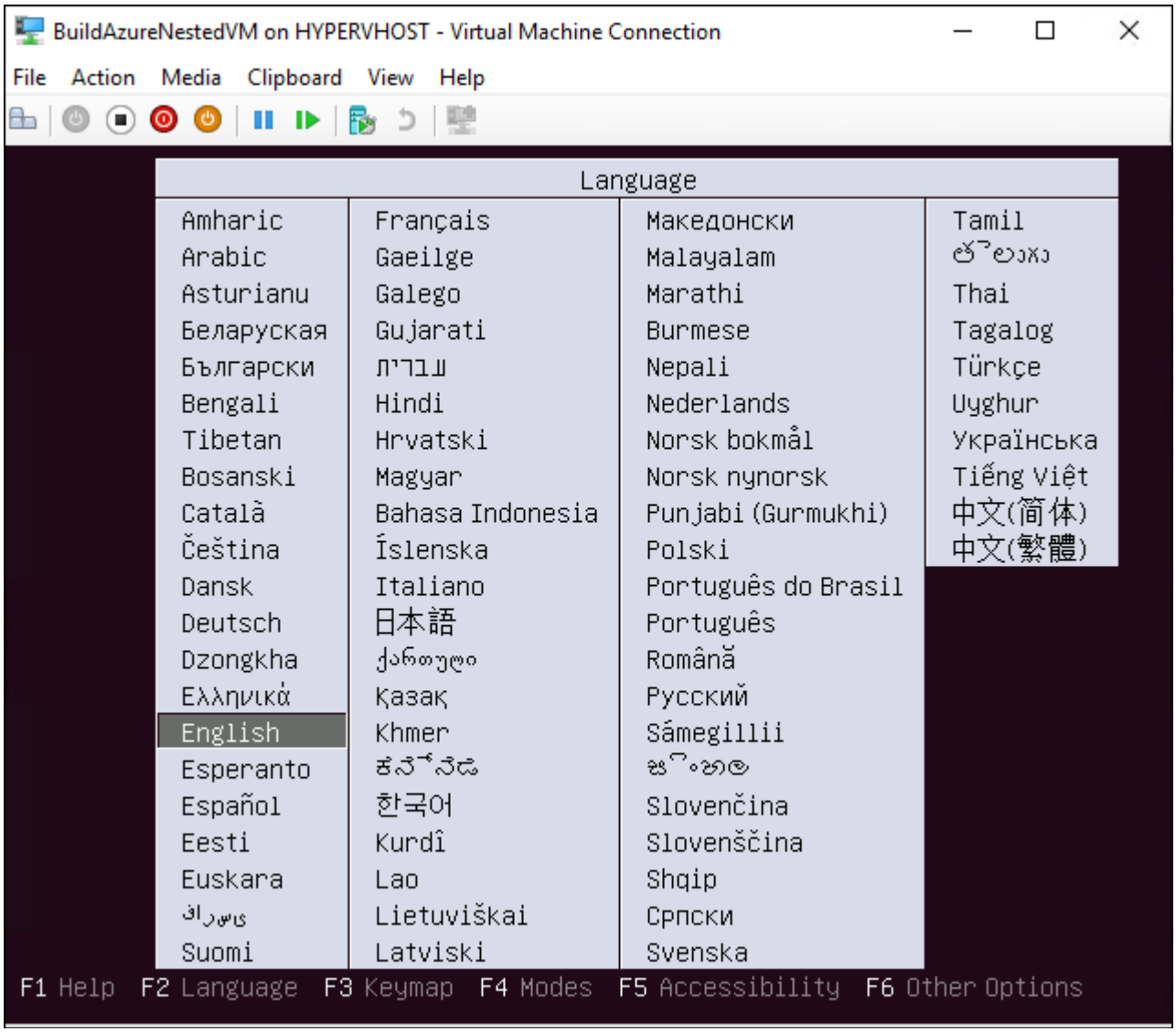
The moment you've been waiting for, your first nested vm in the cloud

There she blows!



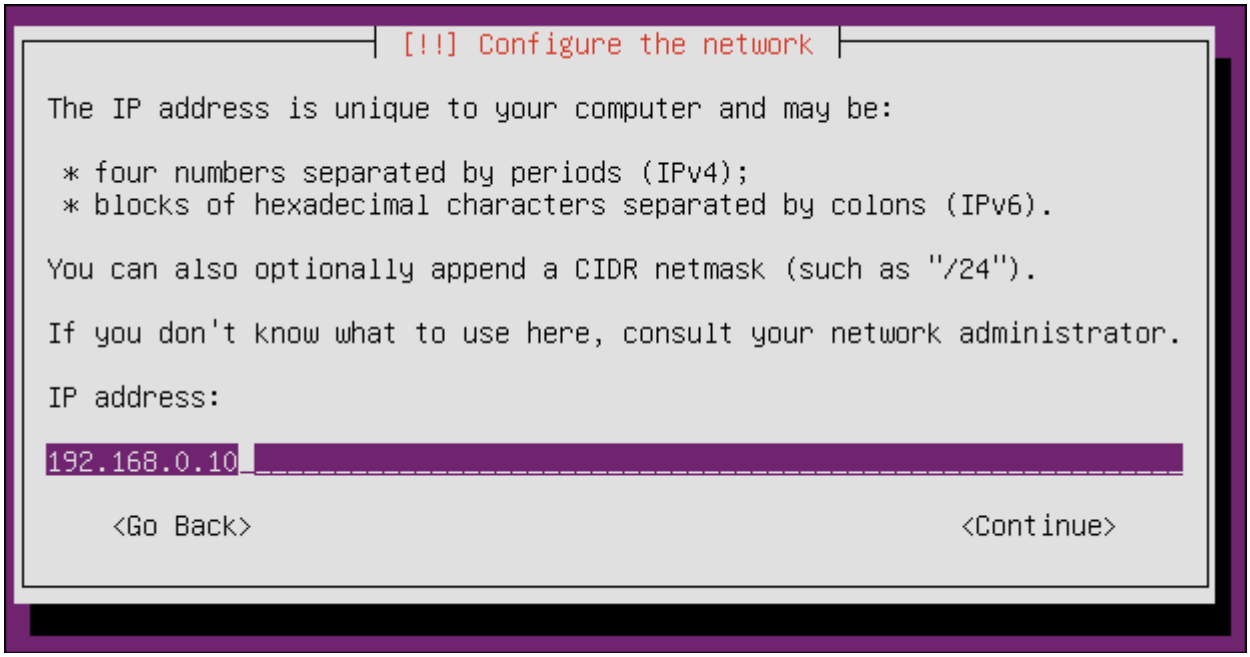
It's alive!

Double click the VM name and you will see the Ubuntu Install manager.



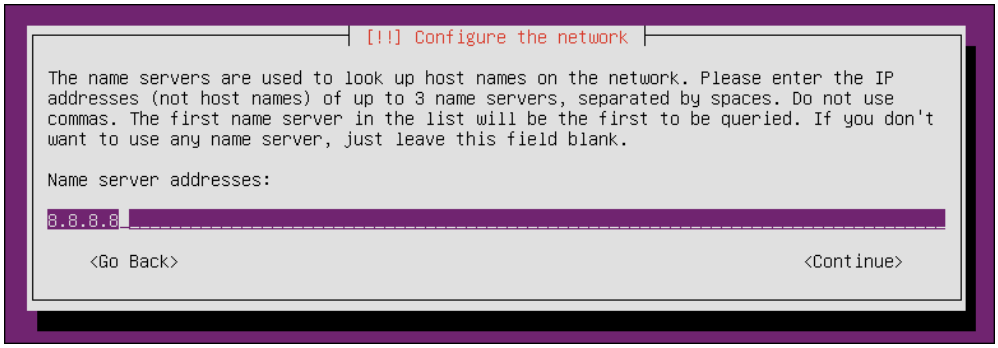
Ubuntu Install

Follow the instructions and complete the Installer using defaults. When you get to the network you will need to add a manual IP address. The internal network is 192.168.0.0/24, so you can use 192.168.0.2-192.168.0.253. Don't worry you won't have an issue with duplicate IPs if you use that range. This is a private internal network only on this HYPERVHOST. The subnet mask should be 255.255.255.0 and the default gateway is 192.168.0.1.



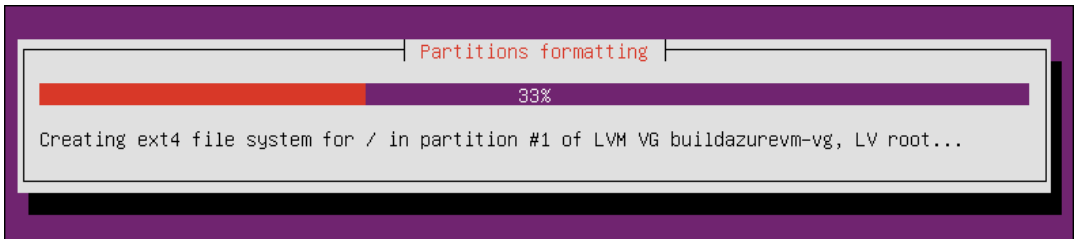
Provide an IP address in the 192.168.0.0/24 Range

The DNS server should be configured as 8.8.8.8.



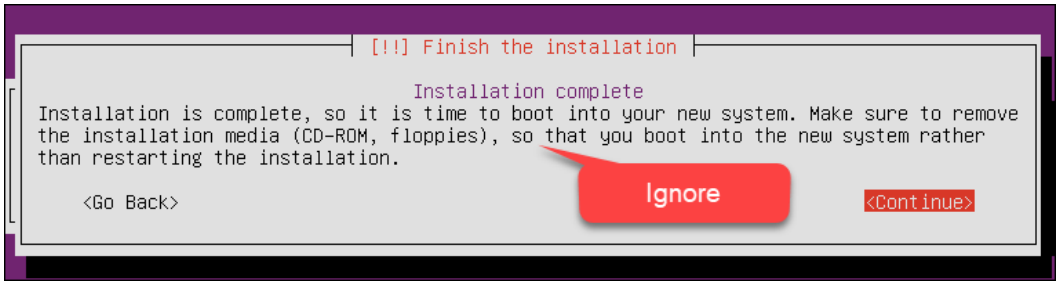
Set DNS to the Google Server

Follow the prompts and accept the defaults for the disk configuration.



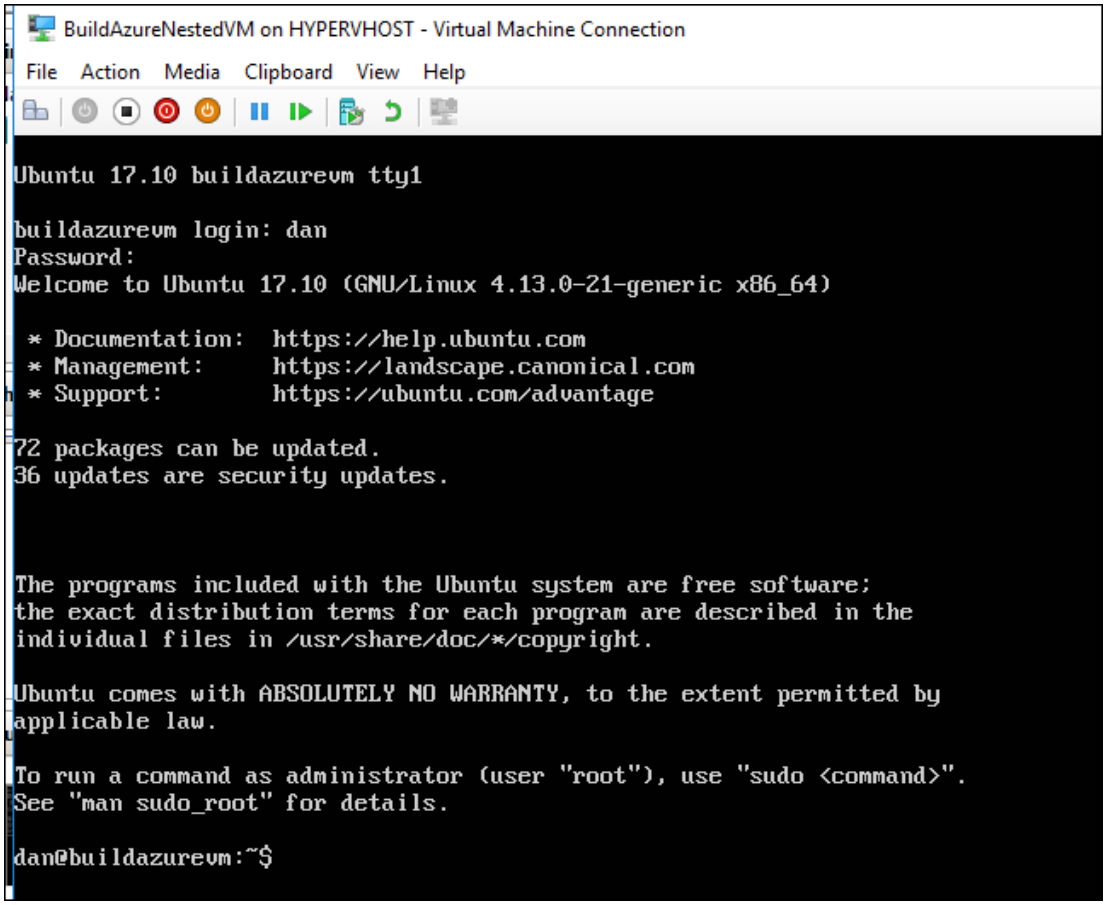
Use the defaults for the disk in Ubuntu

After the Ubuntu installation has completed you will get a warning that you need to remove the ISO from the virtual DVD drive of the VM, but you can ignore this.



Hyper-V will eject the ISO for you

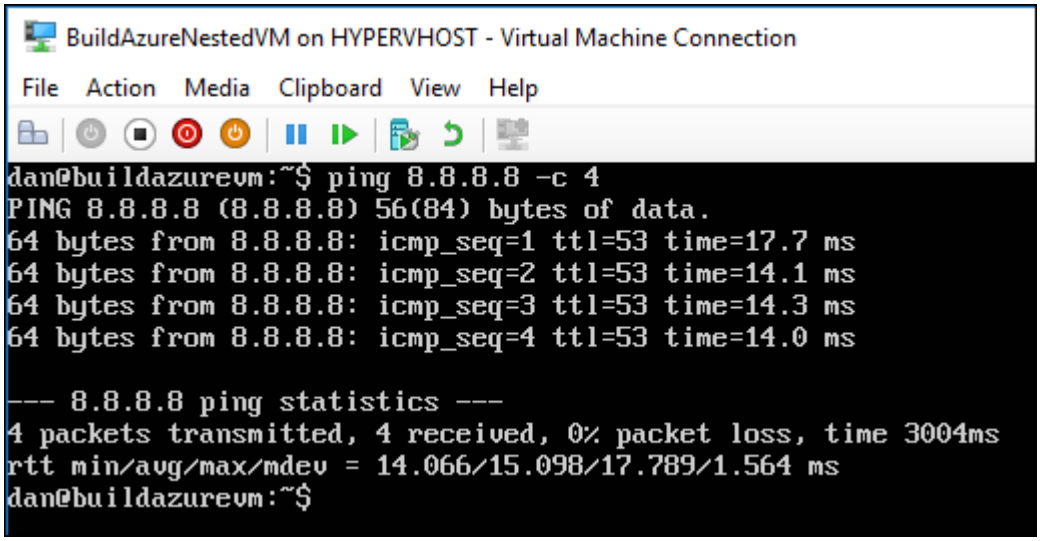
After the VM boots, you can login using the username and password that you provided during the install.



Login to your Nested VM

After the login, you can run a ping to see if the connection works. Here I did a Ping to the Google DNS Server:

```
ping 8.8.8.8 -c 4
```



Well, I hope this post helps you get going with Nested VMs in Azure. This could be used for almost anything you can think up with respect Dev/Test or any other number of situations.

Fork the Repo and enjoy!

https://github.com/sunilake/AzureNestedHyperV.git