The IaaS Deployment Tool User Guide – everything not included in the PowerShell Script itself

Written by: John N Lewis Last Updated 9/10/2016

Contents

Getting up and Running

This guide is intended to get you up and running with the iaas_deployment_tool.ps1 script in a little under 5 minutes. Let's get through the painful stuff first.

1. The script requires Azure PowerShell version 2.0 or newer in order to run. If the script is executed with an older version of PowerShell, you will see an error indicating the version is not correct. Please update to 2.x or newer and re-run the script if this occurs.
2. Within the script there is a function called 'VerifyProfile' which contains a parameter for a file path and json file. $ProfileFile = "C:\temp\profile.json" . If you are leveraging a json file for authentication "Select-AzureRmProfile -Path $ProfileFile" currently for other scripts you can use the same json here to automate the login of your account to azure when the script is run. If you

haven't yet used this capability login to Azure using 'Add-AzureRmAccount' and execute the command save-azurermprofile -Path (Local path on your workstation). Update the parameter above with the filename and path you saved the json to and you will be able to login using the saved profile json. Otherwise you will be prompted for your credentials at run-time.

3. Note that a .log file leveraging the current vmname and time/date stamp will be created in the directory the script is executed in.

What the tool does and does not handle for you (for your information)

The tool will handle the naming used for the storage and network adapters by using the vmname as a base. If an Availability Set is created and no Availability Set Name is passed, the script will randomly generate a name for the availability set. If you wish to specify things like the storage type used for the storage account, they are available to pass as parameters at runtime. You can also specify the names of the network interfaces by passing the $InterfaceName1 or $InterfaceName2 parameters. The tool does not give your vm a name, tell it what type of image to create or tell it the resource groups or network to connect too.

Command Syntax

Minimum parameters that are required to execute the script

Summary: VMName and the other parameters below are required at runtime. Depending on which values are passed with regards to ConfigIPs, additional information may be required. This configuration expects an existing VNET, if one does not exist with the name passed it will create it. If you are just getting started you could try out creating a windows 2012 r2 server with a single NIC on a new VNET by passing

-VM 'win001' -Image 'w2k12' -Rg 'ResGroup' -vnetrg 'ResGroup' -vnet 'myvnet' -ConfigIPs 'Single'

Note that if a required parameter is missing you will see something like the below.

Options that are available for Network Configuration using -ConfigIps

PvtSingleStat & PvtDualStat – Deploys the server with a Public IP and the private IP(s) specified by the user. * SubnetID and NicIp are required parameters in this scenario.

-sub1 '5' -ConfigIPs 'PvtSingleStat' -nic1 10.120.4.169

NoPubSingle & NoPubDual - Deploys the server without Public IP using automatically generated private IP(s). * SubnetIDs are required parameters in this scenario.

-sub1 '4' -sub2 '5' -ConfigIPs 'NoPubDual'

Single & Dual – Deploys the default configuration of a Public IP and automatically generated private IP(s). * SubnetIDs are suggested parameters in this scenario.

-sub1 '6' -sub2 '7' -configip 'dual'

StatPvtNoPubDual Deploys the server

& StatPvtNoPubSingle – without a Public IP using

the private IP(s) specified by the user. * SubnetID and NicIp are required parameters in this scenario.

-sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.9 -Nic2 10.120.2.7

Optional Commands to pass at runtime

-AddVnet 'True'/'False' – Creates a new VNET with the VNETName specified. By default, the VNET created will be a 10.120.0.x IP space with 8 subnets. You can change the number of subnets and how the IP address space through the paramaters available in the script.

-AddVNet 'True' -VNETName 'vnet'

-NSGEnabled 'True'/'False' – Creates an NSG if one does not exist, updates existing NSG, adds new VM interfaces to existing NSG. The NSG Rules are defined within the function for the NSG in the script.

-NSGEnabled 'True' -NSGName 'NSG'

-AddAvailabilitySet 'True'/'False' – Creates a new availability set with an auto generated name if -AvailabilitySetName is not passed. Updates an existing availability set if one already exists and the -AvailabilitySetName is passed.

-AddAvailabilitySet 'True' -AvailabilitySetName 'myavailabilityset'

-AddFQDN 'True'/'False' – Creates a fully qualified domain name for the Public IP Address of the new VM based on the -DNLabel. For example:

-sub1 '4' -ConfigIPs Single -AddFQDN 'True' -fqdn myssda1

-AzExtConfig 'ExtensionName' – Deploys the Azure Extension Specified after the VM has been deployed. Can be used to deploy custom scripts, chef agent, puppet agents, azure diagnostics, OMS agent and DSC.

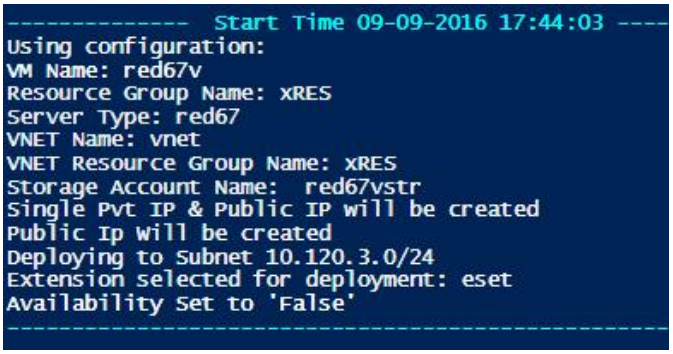
-AzExtConfig diag

Note that some extensions require additional information to passed in order to complete successfully.

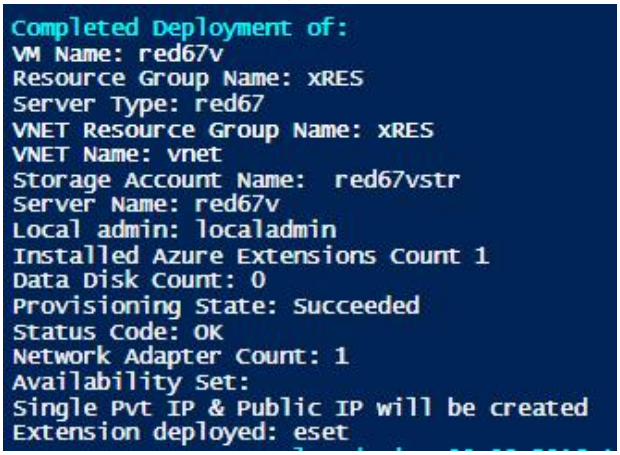
In addition to the options above there are numerous optional parameters that can be passed at runtime or updated directly in the script. For example -VMSize, and -StorageType allow you to choose larger Skus or different Azure Storage types (such as Premium) when creating the VM. SubscriptionID and TenantID can be included if you are leveraging more than one subscription. The local administrator account name can also be passed at runtime. If the user wishes to deploy in a different location for Azure, there is one -Location parameter that can be passed that the script will leverage (the default is uswest).

User Interface Examples

Pre Execution Information

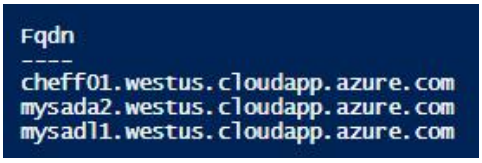


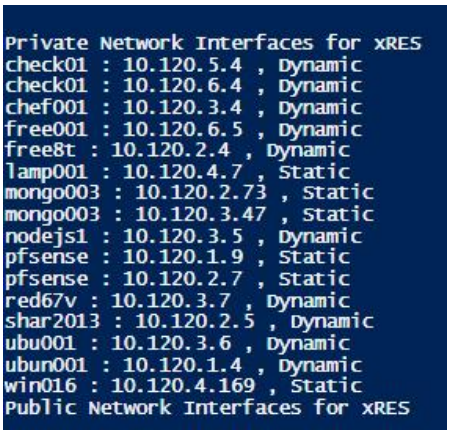
Post Execution Summary



Post Deployment Information

Lists all DNS names, Storage Account Names, Private/Public IPs, and Availability Sets for the Resource Group





```
Name          IpAddress
----          ---------
check01_nic1  138.91.148.241
chef001_nic1  13.88.185.117
free001_nic1  13.88.190.2
free8t_nic1   104.45.226.245
```

```
Availability Sets for xRES

Name       ResourceGroupName
----       -----------------
cEHuKyaip  xRES
myavail1   xRES
OashcMaip  xRES
WsIGbZaip  xRES
```

Logging Example

[09-09-2016 17:33:55] Completed Pre Execution Verification Checks.

[09-09-2016 17:34:31] Completed Network Configuration of vnet.

[09-09-2016 17:34:33] Security Rules added for NSG.

[09-09-2016 17:34:33] Completed NSG Configuration of NSG.

[09-09-2016 17:35:05] Storage Configuration completed: shar2013str.

[09-09-2016 17:35:06] Completed Availability Set configuration myavail1.

[09-09-2016 17:36:10] Completed image prep 'Publisher:'MicrosoftSharePoint 'Offer:'MicrosoftSharePointServer 'Sku:'2013 'Version:'latest.

[09-09-2016 17:36:10] Completed adding NIC.

[09-09-2016 17:43:14] Completed Creation of shar2013 from share2013.

[09-09-2016 17:43:46] Completed Image NSG Post Configuration. Added shar2013_nic1 to NSG.

Batch Operations with a CSV

Using the import-csv module it is possible to deploy whole test or dev environment without entering multiple commands.

Simple Example

Below is an example of using the IaaS Deployment script in conjunction with a CSV in order to create a new environment with a PFSense Server, Chef Server and 10 Red Hat Linux VMs.

```
import-csv -Path $csvin -Delimiter ',' | ForEach-Object{.\azdeploy.ps1 -VMName $_.VMName -vmMarketImage $_.vmMarketImage -ResourceGroupName $_.ResourceGroupName -vNetResourceGroupName $_.ResourceGroupName -VNetName $_.VNetName -ConfigIPs $_.ConfigIPs -subnet1 $_.Subnet1 -subnet2 $_.Subnet2}
```

Example Simple CSV File

VMName	VMMarke	ResourceG	VNetReso	VNetNam	ConfigIPs	Subnet1	Subnet2
pf0002	pfsense	RESX	RESX	vnet	Dual	3	4
chef001	chef	RESX	RESX	vnet	Single	6	2
redh200	red67	RESX	RESX	vnet	Single	5	2
redh201	red72	RESX	RESX	vnet	Single	5	2
redh202	red67	RESX	RESX	vnet	Single	5	2
redh203	red67	RESX	RESX	vnet	Single	5	2
redh204	red67	RESX	RESX	vnet	Single	5	2
redh205	red67	RESX	RESX	vnet	Single	5	2
redh206	red67	RESX	RESX	vnet	Single	5	2
redh207	red67	RESX	RESX	vnet	Single	5	2
redh208	red67	RESX	RESX	vnet	Single	5	2
redh209	red67	RESX	RESX	vnet	Single	5	2

Complex Example

The Complex example shows how static IPs, FQDNs and Availability Sets can be used in a spreadsheet.

```
import-csv -Path $csvin -Delimiter ',' | ForEach-Object{.\azdeploy.ps1 -VMName $_.VMName -
vmMarketImage $_.vmMarketImage -ResourceGroupName $_.ResourceGroupName -
vNetResourceGroupName $_.ResourceGroupName -VNetName $_.VNetName -ConfigIPs $_.ConfigIPs -
subnet1 $_.Subnet1 -subnet2 $_.Subnet2 -PvtIPNic1 $_.PvtIPNic1 -PvtIPNic2 $_.PvtIPNic2 -AddVnet
$_.AddVnet -NSGEnabled $_.NSGEnabled -AddAvailabilitySet $_.AddAvailabilitySet -AddFQDN
$_.AddFqdn -DNLabel $_.DNLabel }
```

VMName	VMMarke	ResourceG	VNetReso	AddVnet	VNetNam	ConfigIPs	Subnet1	Subnet2	PvtIPNic1	PvtIPNic2	AddAvaila	NSGEnabl	NSGName	AddFqdn	DNLabel
pf0005	pfsense	RESX	RESX	TRUE	aipvnet	StatPvtNo	4	5	10.120.3.7	10.120.4.7	TRUE	TRUE	NSG	TRUE	mytesta0
chk001	check	RESX	RESX	TRUE	aipvnet	PvtDualSt	4	5	10.120.3.7	10.120.4.7	TRUE	TRUE	NSG	TRUE	mytesta0
redh202	red67	RESX	RESX	TRUE	aipvnet	StatPvtNo	6	1	10.120.1.1	127.0.0.1	TRUE	TRUE	NSG	TRUE	myesta03
redh201	red72	RESX	RESX	TRUE	aipvnet	Single	4	1	10.120.3.1	127.0.0.1	FALSE	TRUE	NSG	TRUE	myesta02
free001	free	RESX	RESX	TRUE	aipvnet	Single	5	1	10.120.4.1	127.0.0.1	FALSE	TRUE	NSG	TRUE	myesta01
win001	w2k12	RESX	RESX	TRUE	aipvnet	PvtSingle	6	1	10.120.5.1	127.0.0.1	FALSE	FALSE	NSG	FALSE	blah
chef001	chef	RESX	RESX	TRUE	aipvnet	PvtSingle	7	1	10.120.6.1	127.0.0.1	FALSE	FALSE	NSG	FALSE	blah
suse001	suse	RESX	RESX	TRUE	aipvnet	StatPvtNo	5	1	10.120.4.5	127.0.0.1	FALSE	FALSE	NSG	FALSE	blah
ubuntu01	ubuntu	RESX	RESX	TRUE	aipvnet	StatPvtNo	7	1	10.120.6.5	127.0.0.1	FALSE	FALSE	NSG	FALSE	blah

Script Structure

Functions

The script leverages numerous functions to execute both validation and execution of the parameters passed by the user. The actual execution of the script only takes place in the last 50 or so lines within the script itself.

Key Functions

ImageConfig Function contains the steps for creating the VM and executes off of the VMMarketImage parameter.

```
Provvms

"*jenkins*" {
    CreateStorage
    AvailSet
    ConfigNet  #Sets network connection info
    MakeImagePlanInfo_Bitnami_jenkins # Begins Image Creation
    ConfigSet # Adds Network Interfaces
    AddDiskImage # Completes Image Creation
    Provvms
```

ProvisionVnet Function deploys the VNET based on the parameters passed by the user.

```
Function ProvisionNet {
Write-Host "Network Preparation in Process.."
$subnet1 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix1 -Name $SubnetNameAddPrefix1
$subnet2 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix2 -Name $SubnetNameAddPrefix2
$subnet3 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix3 -Name $SubnetNameAddPrefix3
$subnet4 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix4 -Name $SubnetNameAddPrefix4
$subnet5 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix5 -Name $SubnetNameAddPrefix5
$subnet6 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix6 -Name $SubnetNameAddPrefix6
$subnet7 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix7 -Name $SubnetNameAddPrefix7
$subnet8 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix8 -Name $SubnetNameAddPrefix8
New-AzureRmVirtualNetwork -Location $Location -Name $VnetName -ResourceGroupName $VnetResourceGroupName -AddressPrefix $AddRange -Subnet $subnet1,$subnet2,$subnet3,$subnet4,$subnet5,$subnet6,$subnet7,$su
Get-AzureRmVirtualNetwork -Name $VnetName -ResourceGroupName $VnetResourceGroupName | Get-AzureRmVirtualNetworkSubnetConfig -WarningAction SilentlyContinue | Out-Null
Write-Host "Network Preparation completed" -ForegroundColor White
$LogOut = "Completed Network Configuration of $VnetName"
Log-Command -Description $LogOut -LogFile $LogOutFile
}
```

CreateNSG Function provisions the NSG. The function contains the security rules the NSG will leverage when provisioned.

```
# End of Provision VNET Function
Function CreateNSG {
Write-Host "Network Security Group Preparation in Process.."
$httprule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_HTTP" -Description "HTTP Exception for Web frontends" -Protocol Tcp -SourcePortRange "80" -DestinationPortRange "80" -SourceAddressPrefix
$httpsrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_HTTPS" -Description "HTTPS Exception for Web frontends" -Protocol Tcp -SourcePortRange "443" -DestinationPortRange "443" -SourceAddressPr
$sshrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_SSH" -Description "SSH Exception for Web frontends" -Protocol Tcp -SourcePortRange "22" -DestinationPortRange "22" -SourceAddressPrefix "*"
$nsg = New-AzureRmNetworkSecurityGroup -ResourceGroupName $VnetResourceGroupName -Location $Location -Name $NSGName -SecurityRules $httprule,$httpsrule, $sshrule -Confirm:$false -WarningAction SilentlyCo
Get-AzureRmNetworkSecurityGroup -Name $NSGName -ResourceGroupName $VnetResourceGroupName -WarningAction SilentlyContinue | Out-Null
Write-Host "Network Security Group configuration completed" -ForegroundColor White
$LogOut = "Security Rules added for $NSGName"
$secrules =Get-AzureRmNetworkSecurityGroup -Name $NSGName -ResourceGroupName $VnetResourceGroupName -ExpandResource NetworkInterfaces | Get-AzureRmNetworkSecurityRuleConfig | Ft Name,Description,Directio
$defsecrules = Get-AzureRmNetworkSecurityGroup -Name $NSGName -ResourceGroupName $VnetResourceGroupName -ExpandResource NetworkInterfaces | Get-AzureRmNetworkSecurityRuleConfig -DefaultRules | Ft Name,De
Log-Command -Description $LogOut -LogFile $LogOutFile
$LogOut = "Completed NSG Configuration of $NSGName"
Log-Command -Description $LogOut -LogFile $LogOutFile
}
# End of Provision Network Security Groups Function
```

InstallExt – Installs Azure Extensions based on AzExtConfig parameter.

Supporting Functions

For each image type there is a corresponding Function named MakeImageNoPlanInfo_ or MakeImagePlanInfo_ depending on if the image requires Plan Info or not.

```
Function MakeImageNoPlanInfo_SharePoint2k16 {
param(
[string]$Publisher = "MicrosoftSharePoint",
[string]$offer = "MicrosoftSharePointServer",
[string]$Skus = "2016",
[string]$version = "latest"
)
Write-Host "Image Creation in Process - No Plan Info - SharePoint 2016 server" -ForegroundColor White
Write-Host 'Publisher:'$Publisher 'Offer:'$offer 'Sku:'$Skus 'Version:'$version
$global:VirtualMachine = Set-AzureRmVMOperatingSystem -VM $VirtualMachine -Windows -ComputerName $VMName -Credential $Credential1 -ProvisionVMAgent
$global:VirtualMachine = Set-AzureRmVMSourceImage -VM $VirtualMachine -PublisherName $Publisher -Offer $offer -Skus $Skus -Version $version
$LogOut = "Completed image prep 'Publisher:'$Publisher 'Offer:'$offer 'Sku:'$Skus 'Version:'$version"
Log-Command -Description $LogOut -LogFile $LogOutFile
}
```

```
Function MakeImagePlanInfo_Bitnami_postgresql {
param(
[string]$Publisher = 'bitnami',
[string]$offer = 'postgresql',
[string]$Skus = '9-5',
[string]$version = 'latest',
[string]$Product = 'postgresql',
[string]$name = '9-5'
)
Write-Host "Image Creation in Process - Plan Info - postgresql" -ForegroundColor White
Write-Host 'Publisher:'$Publisher 'Offer:'$offer 'Sku:'$Skus 'Version:'$version
$global:VirtualMachine= Set-AzureRmVMPlan -VM $VirtualMachine -Name $name -Publisher $Publisher -Product $Product
$global:VirtualMachine = Set-AzureRmVMOperatingSystem -VM $VirtualMachine -linux -ComputerName $VMName -Credential $Credential1
$global:VirtualMachine = Set-AzureRmVMSourceImage -VM $VirtualMachine -PublisherName $Publisher -Offer $offer -Skus $Skus -Version $version
$LogOut = "Completed image prep 'Publisher:'$Publisher 'Offer:'$offer 'Sku:'$Skus 'Version:'$version"
Log-Command -Description $LogOut -LogFile $LogOutFile
}
Function MakeImagePlanInfo_Oracle_linux {
```

Validation Functions

AzureVersion Function – Verifies Azure Runtime version

```
Function AzureVersion{
$name='Azure'
if(Get-Module -ListAvailable |
    Where-Object { $_.name -eq $name })
{
$ver = (Get-Module -ListAvailable | Where-Object{ $_.Name -eq $name }) |
    select version -ExpandProperty version
    Write-Host "current Azure PowerShell Version:" $ver
$currentver = $ver
    if($currentver -le '2.0.0'){
    Write-Host "expected version 2.0.1 found $ver" -ForegroundColor DarkRed
    exit
    }
}
else
{
    Write-Host "The Azure PowerShell module is not installed."
    exit
}
}
```

VerifyProfile Function – Uses AzureRm-Profile to login if profile exists

```
Function VerifyProfile {
$ProfileFile = $Profile
$fileexist = Test-Path $ProfileFile
  if($fileexist)
  {Write-Host "Profile Found"
  Select-AzureRmProfile -Path $ProfileFile
  }
  else
  {
  Write-Host "Please enter your credentials"
  Add-AzureRmAccount -TenantId $TenantID -SubscriptionId $SubscriptionID
  }
}
```

Chknull Function – validates parameters for runtime execution

```
function chknull {
if(!$vmMarketImage) {
Write-Host "Please Enter vmMarketImage"
exit }
    elseif(!$VMName) {
    Write-Host "Please Enter vmName"
    exit}
        elseif(!$VNetName) {
        Write-Host "Please Enter VNet Name"
        exit}
            elseif(!$ResourceGroupName) {
            Write-Host "Please Enter Resource Group Name"
            exit}
                elseif(!$Location) {
                Write-Host "Please Enter Location"
                exit}
                    elseif(!$ConfigIPs) {
                    Write-Host "Please Enter IP Configuration"
                    exit}
                        elseif(!$VNETResourceGroupName) {
                        Write-Host "Please Enter VNET Resource Group Name"
                        exit
                                        }
}
```

OrphanChk Function – validates no orphans exists for the VM being created

VerifyNet – Verifies Private IP addresses (when applicable)

StorageNameCheck – Verifies Storage Name being created does not exist.

```
Function StorageNameCheck
{
$checkname =  Get-AzureRmStorageAccountNameAvailability -Name $StorageName | Select-Object -ExpandProperty NameAvailable
if($checkname -ne 'True') {
Write-Host "Storage Account Name in use, please choose a different name for your storage account"
Start-Sleep 5
exit
}
}
```

```
No Orphans Found, proceeding with deployment..
Completed Pre Execution Verification Checks.
Storage Account Name in use, please choose a different name for your storage account
```

Examples Index

Firewall/Proxy Images – Dual Homed

.\azdeploy.ps1 -VM pfsense -Image pfsense -Rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.9 -Nic2 10.120.2.7

.\azdeploy.ps1 -vm check01 -image check -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 6 -sub2 7 -configip dual -avset 'True'

.\azdeploy.ps1 -VM barr001 -Image barrahourngfw -Rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.11 -Nic2 10.120.2.11

.\azdeploy.ps1 -VM barr002 -Image barrahourspam -Rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.12 -Nic2 10.120.2.12

.\azdeploy.ps1 -VM f5app01 -Image f5appfire -Rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.13 -Nic2 10.120.2.13

.\azdeploy.ps1 -VM f5lb01 -Image f5bigip -Rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '2' -sub2 '3' -ConfigIPs StatPvtNoPubDual -Nic1 10.120.1.16 -Nic2 10.120.2.16

Microsoft Images

.\azdeploy.ps1 -vm win016 -image w2k16 -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '5' -ConfigIPs PvtSingleStat -nic1 10.120.4.169 -AddFQDN 'True' -fqdn mysadl1

.\azdeploy.ps1 -vm shar2013 -image share2013 -rg ResGrp -vnetrg ResGrp -vnet vnet -NSGEnabled 'True' -sub1 '3' -ConfigIPs Single -avset 'True' -AvailSetName myavail1

.\azdeploy.ps1 -vm win018 -image w2k12 -rg ResGrp -vnetrg ResGrp -vnet vnet -ConfigIPs Single

.\azdeploy.ps1 -vm win009 -image w2k12 -rg ResGrp -vnetrg ResGrp -vnet vnet -NSGEnabled 'True' -sub1 6 -sub2 7 -ConfigIPs Dual -AddFQDN 'True' -fqdn mysadl54

.\azdeploy.ps1 -vm sql007 -image sql2016 -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '5' -ConfigIPs PvtSingleStat -nic1 10.120.4.197

Linux Images

.\azdeploy.ps1 -vm ubu001 -image ubuntu -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '4' -ConfigIPs Single -AddFQDN 'True' -fqdn myssda1

.\azdeploy.ps1 -vm free8t -image free -rg ResGrp -vnetrg ResGrp -vnet vnet -NSGEnabled 'True' -sub1 '3' -ConfigIPs Single -avset 'True'

.\azdeploy.ps1 -vm suse67x -image suse -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '4' -ConfigIPs Single -AzExtConfig linuxOsPatch

.\azdeploy.ps1 -vm cent8dy -image centos -rg ResGrp -vnetrg ResGrp -vnet vnet -NSGEnabled 'True' -sub1 '4' -ConfigIPs NoPubDual -avset 'True' -AzExtConfig diag

Config Management Images

.\azdeploy.ps1 -vm chef001 -image chef -rg ResGrp -vnetrg ResGrp -vnet vnet -sub1 '4' -configip single -AddFQDN 'True' -fqdn 'cheff01'

Error Reference and Information

The most common issue when using the Iaas deployment tool is that the user will encounter an error for 'Not having accepted legal terms' for the image they are attempting to deploy. If you encounter this error (below) it means you need to provision the image manually in the Azure Portal one time in order to accept the licensing terms for that particular image. Once you have deployed the image one time from the portal you should have no issues with deploying the same image programmatically.

New-AzureRmVM : User failed validation to purchase resources. Error message: 'Legal terms have not been accepted for this item on this subscription. To accept legal terms, please go to the Azure portal

(http://go.microsoft.com/fwlink/?LinkId=534873) and configure programmatic deployment for the Marketplace item or create it there for the first time'

ErrorCode: ResourcePurchaseValidationFailed

ErrorMessage: User failed validation to purchase resources. Error message: 'Legal terms have not been accepted for this item on this subscription. To accept legal terms, please go to the Azure portal

(http://go.microsoft.com/fwlink/?LinkId=534873) and configure programmatic deployment for the Marketplace item or create it there for the first time'

StatusCode: 400

ReasonPhrase: Bad Request

OperationID : 321fde37-cdb5-4af7-8bb3-e72ddbac14e0

At C:\Users\admin\Source\InProg\AIP_ARM\deploy\azdeploy.ps1:739 char:3

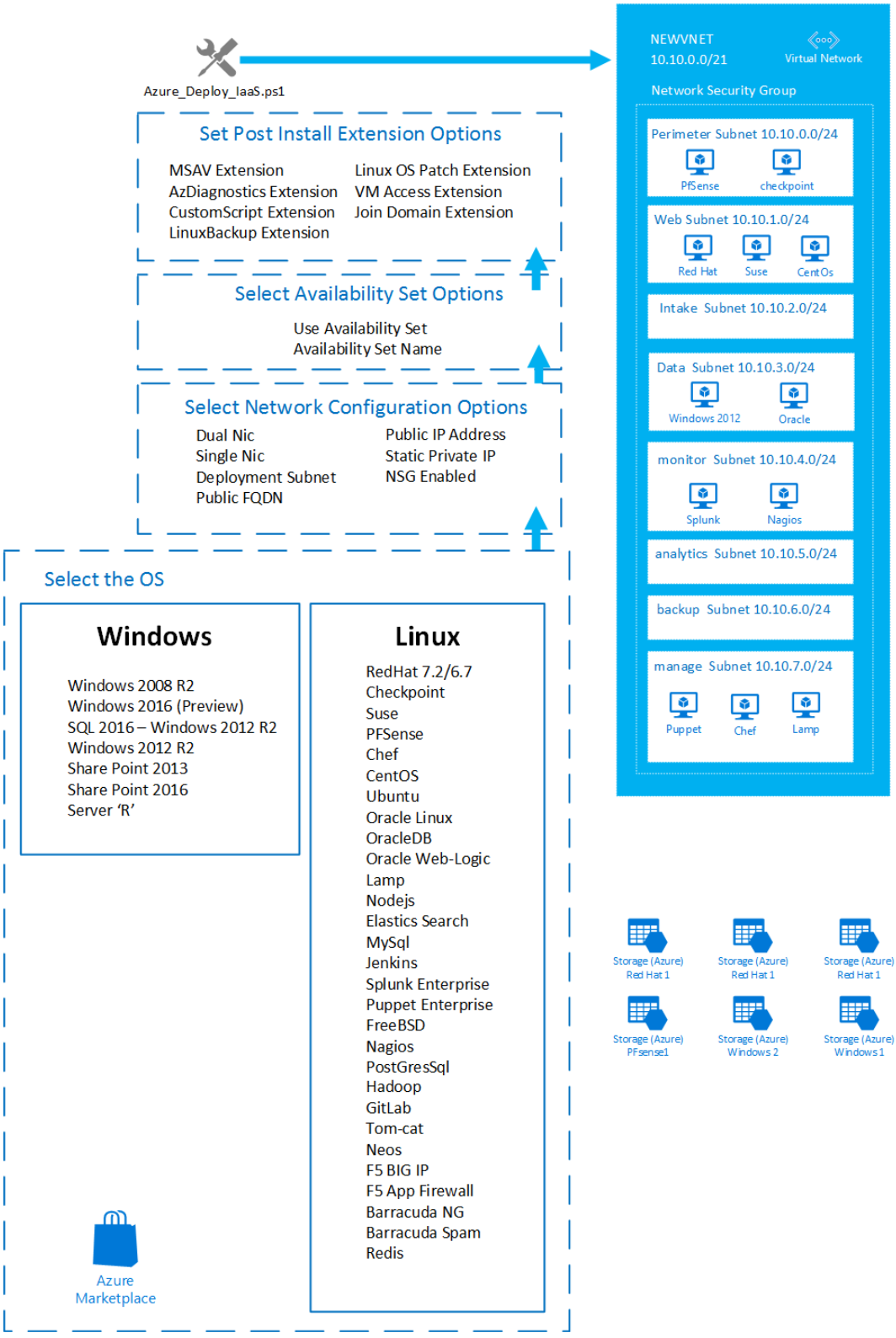
+ New-AzureRmVM -ResourceGroupName $ResourceGroupName -Location ...

+ ~~~~~~

+ CategoryInfo : CloseError: (:) [New-AzureRmVM], ComputeCloudException

+ FullyQualifiedErrorId : Microsoft.Azure.Commands.Compute.NewAzureVMCommand

Diagram


Azure_Deploy_IaaS.ps1
Set Post Install Extension Options
MSAV Extension
AzDiagnostics Extension
CustomScript Extension
LinuxBackup Extension
Linux OS Patch Extension
VM Access Extension
Join Domain Extension
Select Availability Set Options
Use Availability Set
Availability Set Name
Select Network Configuration Options
Dual Nic
Single Nic
Deployment Subnet
Public FQDN
Public IP Address
Static Private IP
NSG Enabled
Select the OS
Windows
Windows 2008 R2
Windows 2016 (Preview)
SQL 2016 – Windows 2012 R2
Windows 2012 R2
Share Point 2013
Share Point 2016
Server 'R'
Linux
RedHat 7.2/6.7
Checkpoint
Suse
PFSense
Chef
CentOS
Ubuntu
Oracle Linux
OracleDB
Oracle Web-Logic
Lamp
Nodejs
Elastics Search
MySql
Jenkins
Splunk Enterprise
Puppet Enterprise
FreeBSD
Nagios
PostGresSql
Hadoop
GitLab
Tom-cat
Neos
F5 BIG IP
F5 App Firewall
Barracuda NG
Barracuda Spam
Redis
Azure Marketplace
NEWVNET
10.10.0.0/21
Virtual Network
Network Security Group
Perimeter Subnet 10.10.0.0/24
PfSense
checkpoint
Web Subnet 10.10.1.0/24
Red Hat
Suse
CentOs
Intake Subnet 10.10.2.0/24
Data Subnet 10.10.3.0/24
Windows 2012
Oracle
monitor Subnet 10.10.4.0/24
Splunk
Nagios
analytics Subnet 10.10.5.0/24
backup Subnet 10.10.6.0/24
manage Subnet 10.10.7.0/24
Puppet
Chef
Lamp
Storage (Azure) Red Hat 1
Storage (Azure) Red Hat 1
Storage (Azure) Red Hat 1
Storage (Azure) PFsense1
Storage (Azure) Windows 2
Storage (Azure) Windows 1