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

Written by: John N Lewis Last Updated 9/11/2016

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 (fyi)

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.

Customizing the tool for your needs

The last few pages of this document describe how to add your own image types, as well as modify the NSG and VNET configurations in the script.

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 & StatPvtNoPubSingle – Deploys the server 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

See the Extensions Index at the end of this document for more information on Extensions.

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

```
No Orphans Found, proceeding with deployment..
Completed Pre Execution Verification Checks.
Subnet IP Validation
Registering resource providers

-------------- Start Time 11-09-2016 08:02:47 --------------
Using configuration:
VM Name: pfsense
Resource Group Name: resx
Server Type: pfsense
VNET Name: vnet
VNET Resource Group Name: xRES
Storage Account Name:  pfsensestr
Dual Static Pvt IP & No Public IP
Public Ip Will not be created
Nic1: 10.120.1.9
Nic2: 10.120.2.7
Deploying to Subnet 10.120.1.0/24
Deploying to Subnet 10.120.2.0/24
Availability Set to 'False'
------------------------------------------------------------
```

Running Execution

```
Network Preparation in Process..
Network Preparation completed
Completed Network Configuration of vnet.
Starting Storage Creation..
Completed Storage Creation
Storage Configuration completed: pfsensestr.
Skipping Availability Set configuration
Skipped Availability Set Configuration.
Dual IP Configuration- Static - No Public
Image Creation in Process - Plan Info - Pfsense
Publisher: netgate Offer: netgate-pfsense-appliance Sku: pfsense-router-fw-vpn-225 Version: latest
Completed image prep 'Publisher:'netgate 'Offer:'netgate-pfsense-appliance 'Sku:'pfsense-router-fw-vpn-225 'Version:'latest.
Adding 2 Network Interface(s) pfsense_nic1 pfsense_nic2
Completed adding NICs.
Completing image creation...
Completed Creation of pfsense from pfsense.
```

Post Execution Summary

```
Completed Deployment of:
VM Name: red67v
Resource Group Name: xRES
Server Type: red67
VNET Resource Group Name: xRES
VNET Name: vnet
Storage Account Name:  red67vstr
Server Name: red67v
Local admin: localadmin
Installed Azure Extensions Count 1
Data Disk Count: 0
Provisioning State: Succeeded
Status Code: OK
Network Adapter Count: 1
Availability Set:
Single Pvt IP & Public IP will be created
Extension deployed: eset
```

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

```
Fqdn
----
cheff01.westus.cloudapp.azure.com
mysada2.westus.cloudapp.azure.com
mysadl1.westus.cloudapp.azure.com
```

```
Private Network Interfaces for xRES
check01 : 10.120.5.4 , Dynamic
check01 : 10.120.6.4 , Dynamic
chef001 : 10.120.3.4 , Dynamic
free001 : 10.120.6.5 , Dynamic
free8t : 10.120.2.4 , Dynamic
lamp001 : 10.120.4.7 , Static
mongo003 : 10.120.2.73 , Static
mongo003 : 10.120.3.47 , Static
nodejs1 : 10.120.3.5 , Dynamic
pfsense : 10.120.1.9 , Static
pfsense : 10.120.2.7 , Static
red67v : 10.120.3.7 , Dynamic
shar2013 : 10.120.2.5 , Dynamic
ubu001 : 10.120.3.6 , Dynamic
ubun001 : 10.120.1.4 , Dynamic
win016 : 10.120.4.169 , Static
Public Network Interfaces for xRES
```

```
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

Logs will be created for each VM in the directory the script is executed from with a time date stamp to provide information on what was deployed and how long it took.

```
Steps will be tracked on the log file : [ C:\Users\admin\Source\InProg\AIP_ARM\deploy\pfsense-2016-09-11-08-02.log ]
current Azure PowerShell Version: 2.1.0
```

[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 -PvtIPNic1 $_.PvtIPNic1 -
PvtIPNic2 $_.PvtIPNic2 }
```

Example Simple CSV File

VMName	VMMarke	Resource(	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	Resource(	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	mytesta06
chk001	check	RESX	RESX	TRUE	aipvnet	PvtDualSt	4	5	10.120.3.7	10.120.4.7	TRUE	TRUE	NSG	TRUE	mytesta05
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
New-AzureRmVirtualNetwork -Location $Location -Name $VNetName -ResourceGroupName
$vNetResourceGroupName -AddressPrefix $AddRange -Subnet $subnet1 -Confirm:$false -
WarningAction SilentlyContinue -Force | Out-Null
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.

```
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 "*" -DestinationAddressPrefix
"10.120.0.0/21" -Access Allow -Direction Inbound -Priority 200
$httpsrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_HTTPS" -Description
"HTTPS Exception for Web frontends" -Protocol Tcp -SourcePortRange "443" -
DestinationPortRange "443" -SourceAddressPrefix "*" -DestinationAddressPrefix
"10.120.0.0/21" -Access Allow -Direction Inbound -Priority 201
```

```powershell
$sshrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_SSH" -Description "SSH
Exception for Web frontends" -Protocol Tcp -SourcePortRange "22" -DestinationPortRange
"22" -SourceAddressPrefix "*" -DestinationAddressPrefix "10.120.0.0/21" -Access Allow
-Direction Inbound ` -Priority 203
$nsg = New-AzureRmNetworkSecurityGroup -ResourceGroupName $vNetResourceGroupName -
Location $Location -Name $NSGName -SecurityRules $httprule,$httpsrule, $sshrule -
Confirm:$false -WarningAction SilentlyContinue -Force | Out-Null
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,Direction,SourcePortRange,DestinationPortRange,DestinationPortRange,S
ourceAddressPrefix,Access
$defsecrules = Get-AzureRmNetworkSecurityGroup -Name $NSGName -ResourceGroupName
$vNetResourceGroupName -ExpandResource NetworkInterfaces | Get-
AzureRmNetworkSecurityRuleConfig -DefaultRules | Ft
Name,Description,Direction,SourcePortRange,DestinationPortRange,DestinationAddressPref
ix,SourceAddressPrefix,Access
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.

With Plan Info

```powershell
Function MakeImagePlanInfo_Bitnami_jenkins {
param(
[string]$Publisher = 'bitnami',
[string]$offer = 'jenkins',
[string]$Skus = '1-650',
[string]$version = 'latest',
[string]$Product = 'jenkins',
[string]$name = '1-650'
)
Write-Host "Image Creation in Process - Plan Info - Jenkins" -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
}
```

Without Plan Info

```powershell
Function MakeImageNoPlanInfo_RedHat72 {
param(
[string]$Publisher = "Redhat",
[string]$offer = "rhel",
[string]$Skus = "7.2",
[string]$version = "latest"
)
Write-Host "Image Creation in Process - No Plan Info - Redhat 7.2" -ForegroundColor
White
Write-Host 'Publisher:'$Publisher 'Offer:'$offer 'Sku:'$Skus 'Version:'$version
$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
}
```

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
{
```

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

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

```
Function OrphanChk {
$extvm = Get-AzureRmVm -Name $VMName -ResourceGroupName $ResourceGroupName -
ErrorAction SilentlyContinue
$nic1 = Get-AzureRmNetworkInterface -Name $InterfaceName1 -ResourceGroupName
$ResourceGroupName -ErrorAction SilentlyContinue
$nic2 = Get-AzureRmNetworkInterface -Name $InterfaceName2 -ResourceGroupName
$ResourceGroupName -ErrorAction SilentlyContinue
$pubip =  Get-AzureRmPublicIpAddress -Name $InterfaceName1 -ResourceGroupName
$ResourceGroupName -ErrorAction SilentlyContinue

if($extvm)
{ Write-Host "Host VM Found, please use a different VMName for Provisioning or
manually delete the existing VM" -ForegroundColor Red
 Start-sleep 10
 exit    }
else {if($nic1)
{ Write-Host "Nic1 already Exists, removing orphan" -ForegroundColor DarkYellow
```

```
        Remove-AzureRmNetworkInterface -Name $InterfaceName1 -ResourceGroupName
$ResourceGroupName -Force -Confirm:$False
 }
        if($pubip)
{ Write-Host "PublicIp already Exists, removing orphan" -ForegroundColor
DarkYellow
                                Remove-AzureRmPublicIpAddress -Name $InterfaceName1
-ResourceGroupName $ResourceGroupName -Force -Confirm:$False
}
        if($nic2)
{ Write-Host "Nic2 already Exists, removing orphan" -ForegroundColor DarkYellow
                            Remove-AzureRmNetworkInterface -Name
$InterfaceName2 -ResourceGroupName $ResourceGroupName -Force -Confirm:$False
 }
 else {Write-Host "No Orphans Found, proceeding with deployment.." -
ForegroundColor Green}
 }
$LogOut = "Completed Pre Execution Verification Checks"
Log-Command -Description $LogOut -LogFile $LogOutFile
} # Verifies no left over components will prohibit deployment of the new VM,
cleans up any if they exist.
```

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

.\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

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

Set Post Install Extension Options

MSAV Extension   Linux OS Patch Extension
AzDiagnostics Extension   VM Access Extension
CustomScript Extension   Join Domain Extension
LinuxBackup Extension

Select Availability Set Options

Use Availability Set
Availability Set Name

Select Network Configuration Options

Dual Nic   Public IP Address
Single Nic   Static Private IP
Deployment Subnet   NSG Enabled
Public FQDN

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

Storage (Azure)   Storage (Azure)   Storage (Azure)
Red Hat 1   Red Hat 1   Red Hat 1

Storage (Azure)   Storage (Azure)   Storage (Azure)
PFsense1   Windows 2   Windows 1

Script Customization

Changing the VNET Configuration

To modify the VNET Address space and Subnets locate the function below.

Address Range - The Address Space can be modified by changing the 'AddRange' parameter in the global parameters.

```
[Parameter(Mandatory=$False,ValueFromPipelinebyPropertyName=$true)]
[string]
$AddRange = '10.120.0.0/21',
```

For each subnet you wish to add you will need to add the values in Green.

```
Function ProvisionNet {
Write-Host "Network Preparation in Process.."
$subnet1 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix1 -
Name $SubnetNameAddPrefix1
$subnet2 = New-AzureRmVirtualNetworkSubnetConfig -AddressPrefix $SubnetAddPrefix2 -
Name $SubnetNameAddPrefix2
New-AzureRmVirtualNetwork -Location $Location -Name $VNetName -ResourceGroupName
$vNetResourceGroupName -AddressPrefix $AddRange -Subnet $subnet1,$subnet2 -
Confirm:$false -WarningAction SilentlyContinue -Force | Out-Null
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
}
```

You will also need to add 2 new parameters to the global parameters for the new Subnet Range and Name, examples below.

```
[Parameter(Mandatory=$False,ValueFromPipelinebyPropertyName=$true)]
[string]
$SubnetAddPrefix2 = "10.120.0.128/25",
[Parameter(Mandatory=$False,ValueFromPipelinebyPropertyName=$true)]
[string]
$SubnetNameAddPrefix2 = 'perimeter',
```

Changing the NSG Security Group Configuration

To modify the security rules deployed with NSG locate the function below and modify the high-lighted lines.

```
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 "*" -DestinationAddressPrefix
"10.120.0.0/21" -Access Allow -Direction Inbound -Priority 200
$httpsrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_HTTPS" -Description
"HTTPS Exception for Web frontends" -Protocol Tcp -SourcePortRange "443" -
DestinationPortRange "443" -SourceAddressPrefix "*" -DestinationAddressPrefix
"10.120.0.0/21" -Access Allow -Direction Inbound -Priority 201
$sshrule = New-AzureRmNetworkSecurityRuleConfig -Name "FrontEnd_SSH" -Description "SSH
Exception for Web frontends" -Protocol Tcp -SourcePortRange "22" -DestinationPortRange
"22" -SourceAddressPrefix "*" -DestinationAddressPrefix "10.120.0.0/21" -Access Allow
-Direction Inbound ` -Priority 203
$nsg = New-AzureRmNetworkSecurityGroup -ResourceGroupName $vNetResourceGroupName -
Location $Location -Name $NSGName -SecurityRules $httprule,$httpsrule, $sshrule -
Confirm:$false -WarningAction SilentlyContinue -Force | Out-Null
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,Direction,SourcePortRange,DestinationPortRange,DestinationPortRange,S
ourceAddressPrefix,Access
$defsecrules = Get-AzureRmNetworkSecurityGroup -Name $NSGName -ResourceGroupName
$vNetResourceGroupName -ExpandResource NetworkInterfaces | Get-
AzureRmNetworkSecurityRuleConfig -DefaultRules | Ft
Name,Description,Direction,SourcePortRange,DestinationPortRange,DestinationAddressPref
ix,SourceAddressPrefix,Access
Log-Command -Description $LogOut -LogFile $LogOutFile
$LogOut = "Completed NSG Configuration of $NSGName"
Log-Command -Description $LogOut -LogFile $LogOutFile
}
```

Adding your own Image as an option for deployment

To add your own image there are several steps involved, but once you have done it once it is very straight forward with the exception of one piece, does the image require Plan Info or not? More on that later.

1. To get started we need to make sure we can locate the correct image in the Market for the region we are using. To do this we either need to use Azure PowerShell and the Get-AzureRmVmImagePublisher, Get-AzureRmVMImageOffer and Get-AzureRmVMImageSKU commands or the azure cli with the 'vm image list-publishers', and 'vm image list' sub-commands to determine the values to use for the next step in the configuration. You should obtain a Publisher, Offer and SKU for example:

    Get-AzureRmVMImagePublisher -Location westus

    Get-AzureRmVMImageOffer -Location westus -PublisherName Puppet

    Get-AzureRmVMImageSku -Location westus -Offer puppet-enterprise -PublisherName puppet

    Which provides the sku (2016-1) you will need in the next step along with the other information.

2. Below are two examples of the functions which handle the types of image being created. Roughly 2/3 of the images I have deployed require the Plan Information to be included, for example Chef and PFSense require it. Essentially what that means is that you will need two additional fields (with plan info) where you will use the same information collected above. Below are the highlighted values that must be updated when the new function is created. Also note that you need to use the same image OS Type of the new machine you are creating (i.e Linux or Windows). Make a copy of the original Function with updated values for the items highlighted below.

```
Function MakeImagePlanInfo_Bitnami_jenkins {
param(
[string]$Publisher = 'bitnami',
[string]$offer = 'jenkins',
[string]$Skus = '1-650',
[string]$version = 'latest',
[string]$Product = 'jenkins',
[string]$name = '1-650'
)
```

```
Write-Host "Image Creation in Process - Plan Info - Jenkins" -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
}
```

3. Once we have the new function created we need to add it to the ImageConfig Function (below) by making a copy of one of the existing nodes and updating the MakeImage Function with the new image function we just added. For example if I had created a new function for J-ruby I could add the node by adding this:

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

4. Lastly we need to make this image type available in the script, which we can do by updating the vmMarketImage parameter (below) to include the name we want to use (in this case 'jruby'.

```
[Parameter(Mandatory=$False,ValueFromPipelinebyPropertyName=$true,Position=1)]
[ValidateSet("w2k12","w2k8","red67","red72","suse","free","ubuntu","centos","w2k16
","sql2016","chef","check","pfsense","lamp","jenkins","nodejs","elastics","postgre
ssql","splunk","oracle-linux","puppet","web-logic","stddb-oracle","entdb-
oracle","serverr","sap","solarwinds","f5bigip","f5appfire","barrahourngfw","barrab
yolngfw","barrahourspam","barrabyolspam","mysql","share2013","share2016","mongodb"
,"nginxstack","hadoop","neos","tomcat","redis","gitlab","jruby")]
[Alias("image")]
[string]
$vmMarketImage = "",
```

Once these steps are completed you are ready to try out the image type you added!

Azure Extensions Index

Compatibility

Name	VMAccess Set-AzureRmVMAccessExtension	linuxOsPatch	linuxBackup Set-AzureRmVMBackupExtension
Command ImageType Compatability		Set-AzureRmVMExtension OSPatchingForLinux	
	Windows	Ubuntu	Suse
			Ubuntu
			Red Hat
			Cent Os

Name	opsinsightLinux	opsinsightWin	ESET
Command ImageType	Set-AzureRmVMExtension OmsAgentForLinux	Set-AzureRmVMExtension MicrosoftMonitoringAgent	Set-AzureRmVMExtension FileSecurity
Compatability	Suse	Windows	Windows
	Ubuntu		
	Red Hat		
	Cent Os		

Name	chefAgent	diag Set-AzureRmVMAEMExtension	Puppet
Command ImageType	Set-AzureRmVMExtension ChefClient		Set-AzureRmVMExtension PuppetEnterpriseAgent
Compatability	Suse	Suse	Windows
	Ubuntu	Ubuntu	
	Red Hat	Red Hat	
	Cent Os	Cent Os	
	Windows	Windows	

Name Command ImageType Compatability	msav Set-AzureRmVMExtension IaaSAntimalware
	Windows

Azure Extensions Information

File Security

```
Set-AzureRmVMExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -Location
$Location -Name "ESET" -ExtensionType "FileSecurity" -Publisher "ESET" -
typeHandlerVersion "6.0" -InformationAction SilentlyContinue -Verbose
```

Ops Insight – Windows

```powershell
Set-AzureRmVMExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -Location
$Location -Name "OperationalInsights" -ExtensionType "MicrosoftMonitoringAgent" -
Publisher "Microsoft.EnterpriseCloud.Monitoring" -typeHandlerVersion "1.0" -
InformationAction SilentlyContinue -Verbose
```

Ops Insight – Linux

```powershell
Set-AzureRmVMExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -Location
$Location -Name "OperationalInsights" -ExtensionType "OmsAgentForLinux" -Publisher
"Microsoft.EnterpriseCloud.Monitoring" -typeHandlerVersion "1.0" -InformationAction
SilentlyContinue -Verbose
```

Chef

```powershell
Set-AzureRmVMExtension -ResourceGroupName $ResourceGroupName -VMName $VMName -Name
"ChefStrap" -ExtensionType "ChefClient" -Publisher "Chef.Bootstrap.WindowsAzure" -
typeHandlerVersion "1210.12" -Location $Location -Verbose -ProtectedSettingString -
SettingString
```

Linux Backup

```powershell
Set-AzureRmVMBackupExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -
Name "VMBackup" -Tag "OSBackup" -WarningAction SilentlyContinue
```

Linux OS Patch

```powershell
Set-AzureRmVMExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -Location
$Location -Name "OSPatch" -ExtensionType "OSPatchingForLinux" -Publisher
"Microsoft.OSTCExtensions" -typeHandlerVersion "2.0" -InformationAction
SilentlyContinue -Verbose
```

Puppet

```powershell
Set-AzureRmVMExtension -VMName $VMName -ResourceGroupName $ResourceGroupName -Location
$Location -Name "PuppetEnterpriseAgent" -ExtensionType "PuppetEnterpriseAgent" -
Publisher "PuppetLabs" -typeHandlerVersion "3.2" -InformationAction SilentlyContinue -
Verbose
```

Domain Join -

```powershell
Set-AzureRmVMADDomainExtension -DomainName $DomName -ResourceGroupName
$ResourceGroupName -VMName $VMName -Location $Location -Name 'DomJoin' -WarningAction
SilentlyContinue
```

Azure Diags

```powershell
Set-AzureRmVMAEMExtension -ResourceGroupName $ResourceGroupName -VMName $VMName -
WADStorageAccountName $StorageName -InformationAction SilentlyContinue -WarningAction
SilentlyContinue
```

MSAV

```powershell
Set-AzureRmVMExtension -ResourceGroupName $ResourceGroupName -VMName $VMName -Name
"MSAVExtension" -ExtensionType "IaaSAntimalware" -Publisher "Microsoft.Azure.Security"
-typeHandlerVersion 1.4 -Location $Location | ft ProvisioningState
```

VM Access

```
Set-AzureRmVMAccessExtension -ResourceGroupName $ResourceGroupName -VMName $VMName -
Name "VMAccess" -typeHandlerVersion "2.0" -Location $Location -Verbose -username
$locadmin -password $locpassword
```