MultiServerDeploy Solution Documentation

Prepared for

9/14/2016

Prepared by

Boklyn Wong

boklynw@microsoft.com

Table of Contents

[1 Overview 3](#_Toc461780403)

[2 Configure multideploy parameters file. 3](#_Toc461780404)

[3 Appendix 5](#_Toc461780405)

[3.1 Dynamic IP 5](#_Toc461780406)

[3.2 Domain Join servers 6](#_Toc461780407)

[3.3 Tagging support 7](#_Toc461780408)

[3.4 Custom Script Extension support. 8](#_Toc461780409)

[3.4.2 Scenario 1 8](#_Toc461780410)

1. Overview

The MultiServerDeployDoc provides a library of modules available with the multideploy solution. This document will provide code swaping instructions to modify the solution..

1. Configure multideploy parameters file.

The following parameter values need to be changed/validated for using the Multi-Deploy template:

* diagStorageAccountName
  + This solution seperates out the location of the VM diagnostic files to a different storage account than the account in which the VHDs are located.
* VHDStorageAccountNames
  + This is a single string that represents the storage account to store the OS disks.
* existingVNETName
  + The Vnet in which the tier of servers will be deployed to.
* Vnetrgpname
  + The resource group in which the vNet you want to depoloy your servers is a member of.
* existingSubnetName
  + the subnet of the vnet that will house your servers.
* domainToJoin
  + This is the domain in which your machines will be joined to.
* domainUsername
  + This option does not need to be used for templates that are not leveraging domain join extension.
  + This is a domain user that has the rights to join the machine to.
  + This is an optional value when provisioning stand alone servers.
* domainPassword
  + This option does not need to be used for templates that are not leveraging domain join extension.
  + You need to ensure that the ID of the KeyVault is correct
  + You need to ensure that the secret of the KeyVault is correct. If choosing the keyvault option.
  + This domain password would be retrieved from KeyVault.
  + This is an optional value when provisioning stand alone servers.
* vmAdminUsername
  + The local useradmin name
* vmAdminPassword
  + You need to ensure that the ID of the KeyVault is correct.
  + You need to ensure that the secret of the KeyVault is correct.
* optionsselectpath
  + You need to ensure that the ID of the KeyVault is correct.
  + You need to ensure that the secret of the KeyVault is correct.
  + This solution depends on this secondary template to enable the ability to provision
* Nicoffset
  + The NIC offset enables the deployment solution to append a numberical number to the nicnameformat. If your nickname format is specificed as cust1w1dr-nic- and the nickname offset is 1, then the first nic will have cust1w1dr-nic-1 and the second will have cust1w1dr-nic-2
* Sitecount
  + The sitecount value represents the number of systems expected to be deployed.
* Nicnameformat
  + This represents the name format of the nic(s) to be provisioned by the solution.
* Vmnames
  + This represents the name(s) of virtual machines to be provisioned. If your building two servers, then the value for the is parameter will be [ "cust1e1-bld-05", "cust1e1-bld-06" ]
* vmSizes
  + This parameter value repsents the size(s) of the virtual machines you need to be provisioned. For the two server example, the value will be [ "Standard\_DS2", "Standard\_DS6"], These values can from any officially supported vmSizes.
* Vmstaticips
  + This solution assumes that the virtual machines you will provision will leverage static ip(s). The value for the two server will be [ "172.26.14.17", "172.26.14.18" ].
* diskcount
  + The nature of this solution requires you to always have at least 1 data disk. The scenario for 0 data disk is not covered by this solution.
* Disksizes
  + This value represents the sizes of VHDs you want the solution to provision. Bear in mind that the limitation for disk sizes is 1 TB. This solution supports up to 16 disks. The following example would create two disks with each disk having 1000 GB for the first server and two disks with each disk having 200 GB for the second server.

[

[ 1000, 1000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],

[ 200, 200, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]

]

You have to specifiy all 16 disks options. In scenarios where you need less than 16 disks you would have to specify the other disk with 0 values.

* Initdisksizes
  + This option does not need to be used for templates not using the custom script extension
  + This solution also allows you to also initialize the disks on the OS level through Windows Server storage pool. The solution will use this value to provision the virtual disks. The following example will create 4 virtual disks each of size of 450 GB for the first server and 4 virtual disks each of size 75 GB. Bear in mind the total sizes of the virtual disks must be less than the size of the total disks requestd in the Disksizes parameters.

[

"450,450,450,450",

"75,75,75,75"

]

* windowsOSversion
  + This is where you would specify the type of OS version to install on the virtual machines. Currently 2012-R2-Datacenter, has been tested and validated.

1. Appendix

This appendix will cover additional scenarios cover by multi-deploy but would require some modification of the template file. Each scenario will have detailed instructions on what you need to change to achieve the additional functionality. The sections below assumes modifications will be made from the basic-template-multisvrdeploy-staticip.json file.

* 1. Dynamic IP

If you want all machines in your environment to leverage dynamic ip instead of the static ip option you would perform the following steps.

* + - 1. Make a copy of the cust1-template-multisvrdeploy\_withkey.cse.json file with the name cust1-template-multisvrdeploydynip\_withkey.cse
      2. To switch the current template from static ip you would need to swap out the following code starting at line 219:

The following code allows for static ip:

"properties": {

"privateIPAllocationMethod": "Static",

"privateIPAddress": "[parameters('vmstaticips')[copyIndex()]]",

"subnet": {

"id": "[variables('subnetId')]"

}

}

Replace it with the following code:

"properties": {

"privateIPAllocationMethod": "Dynamic",

"subnet": {

"id": "[variables('subnetId')]"

}

}

Now you have a template that supports provisioning of multiple servers and all servers provisioned will leverage dynamic ip allocation method. This solution would work wheter you choose/not choose to remove the following section of your parameters file:

"vmstaticips": {

"value": [ "172.26.14.17" ]

},

**NOTE:** We do however reccomend you remove the value to provide a clean and easy to understand parameters file.

The template in the event that it does not detect this parameter value will substitute a empty value since it is no longer being consumed.

* 1. Domain Join servers

Sometimes there are scenarios in which you need to provision a series of servers in which they have to be join to an existing domain. To leverage multideploy to join to the domain you would need to add the following code lines after 319.

{

"apiVersion": "[variables('apiVersion')]",

"type": "Microsoft.Compute/virtualMachines/extensions",

"name": "[concat(parameters('vmnames')[copyIndex()],'/joindomain')]",

"location": "[resourceGroup().location]",

"dependsOn": [

"[concat('Microsoft.Compute/virtualMachines/', parameters('vmnames')[copyIndex()])]",

"[resourceId('Microsoft.Resources/deployments', concat(toLower(parameters('vmnames')[copyIndex()]),'-disksel'))]",

"[resourceId('Microsoft.Compute/virtualMachines/extensions', parameters('vmnames')[copyIndex()],'cseexec')]"

],

"properties": {

"publisher": "Microsoft.Compute",

"type": "JsonADDomainExtension",

"typeHandlerVersion": "1.0",

"settings": {

"Name": "[parameters('domainToJoin')]",

"OUPath": "[parameters('ouPath')]",

"User": "[concat(parameters('domainToJoin'), '\\', parameters('domainUsername'))]",

"Restart": "true",

"Options": "[parameters('domainJoinOptions')]"

},

"protectedsettings": {

"Password": "[parameters('domainPassword')]"

}

}

}

So before the code starting at line 312 should look like:

"diagnosticsProfile": {

"bootDiagnostics": {

"enabled": "true",

"storageUri": "[concat('http://',parameters('diagStorageAccountName'),'.blob.core.windows.net')]"

}

}

}

}

After the addition of new code the code should look like:

"diagnosticsProfile": {

"bootDiagnostics": {

"enabled": "true",

"storageUri": "[concat('http://',parameters('diagStorageAccountName'),'.blob.core.windows.net')]"

}

}

}

},

{

"apiVersion": "[variables('apiVersion')]",

"type": "Microsoft.Compute/virtualMachines/extensions",

"name": "[concat(parameters('vmnames')[copyIndex()],'/joindomain')]",

"location": "[resourceGroup().location]",

"dependsOn": [

"[concat('Microsoft.Compute/virtualMachines/', parameters('vmnames')[copyIndex()])]",

"[resourceId('Microsoft.Resources/deployments', concat(toLower(parameters('vmnames')[copyIndex()]),'-disksel'))]",

"[resourceId('Microsoft.Compute/virtualMachines/extensions', parameters('vmnames')[copyIndex()],'cseexec')]"

],

"properties": {

"publisher": "Microsoft.Compute",

"type": "JsonADDomainExtension",

"typeHandlerVersion": "1.0",

"settings": {

"Name": "[parameters('domainToJoin')]",

"OUPath": "[parameters('ouPath')]",

"User": "[concat(parameters('domainToJoin'), '\\', parameters('domainUsername'))]",

"Restart": "true",

"Options": "[parameters('domainJoinOptions')]"

},

"protectedsettings": {

"Password": "[parameters('domainPassword')]"

}

}

}

* 1. Tagging support

This solution can support tagging. In order to provide tagging support you would need to add the following code after line 269:

,

"tags": {

"tag1": "[parameters('tag1')[copyIndex()]]"

},

So before the code should look like:

"location": "[resourceGroup().location]",

"dependsOn": [

"[resourceId('Microsoft.Network/networkInterfaces', concat(parameters('nicnameformat'), copyIndex(parameters('nicoffset'))))]",

"[resourceId('Microsoft.Resources/deployments', concat(toLower(parameters('vmnames')[copyIndex()]),'-disksel'))]"

],

After the modification the code should look like:

"location": "[resourceGroup().location]",

"tags": {

"tag1": "[parameters('tag1')[copyIndex()]]"

},

"dependsOn": [

"[resourceId('Microsoft.Network/networkInterfaces', concat(parameters('nicnameformat'), copyIndex(parameters('nicoffset'))))]",

"[resourceId('Microsoft.Resources/deployments', concat(toLower(parameters('vmnames')[copyIndex()]),'-disksel'))]"

],

* 1. Custom Script Extension support.

There are scenarios in which you would need to leverage the custom script extension to perform additional steps after the server is provisioned. This section will describe the changes you would need to make to implement custom script extension. The two common scenarios of using custom script extension when you are deploying multiple servers are:

* + - 1. All your servers require the same series of actions after the provisioning phase.
      2. Each server require a custom series of actions after the provisioning phase.

In our example we are going to have a custom script create a new storage pool and add the disks we provisioned into the pool. The sript will then initializ the virtual disk so that when we log onto the box the disks should be ready for use without additional customizations. For each scenario we are going to show you how we are going to modify the template to support custom script extensions.

* + 1. Scenario 1

For scenario 1 the following code would needed to be added after line 319

,

{

"type": "Microsoft.Compute/virtualMachines/extensions",

"name": "[concat(parameters('vmnames')[copyIndex()],'/cseexec')]",

"apiVersion": "2015-05-01-preview",

"location": "[resourceGroup().location]",

"dependsOn": [

"[concat('Microsoft.Compute/virtualMachines/', parameters('vmnames')[copyIndex()])]",

"[resourceId('Microsoft.Resources/deployments', concat(toLower(parameters('vmnames')[copyIndex()]),'-disksel'))]"

],

"properties": {

"publisher": "Microsoft.Compute",

"type": "CustomScriptExtension",

"typeHandlerVersion": "1.3",

"autoUpgradeMinorVersion": "true",

"settings": {

"fileUris": [

"[parameters('initidiskpath')]"

]

},

"protectedSettings": {

"commandToExecute": "[concat('powershell.exe -Executionpolicy Unrestricted -File ','initdisk\_cse.ps1 -disks ', parameters('initdisksizes')[copyIndex()])]"

}

}

}

I have provided you with an already modified template called basic-template-multisvrdeploy-csesame.json for reference.

In this example the one script that will be executed will be called initdisk\_cse.ps1

This modified template will now leverage the already established parameter, initidiskpath. This should be the location in which the server will pull down the the powershell script to run on the provisioned box.

One of the input parameters of the script requires disk sizes for the virtual disk. The solution will support the scenario in which you woulkd want two 40 GB virtual disk from a physical disk of 100 GB. You would now need to provide a value for the parameter initdisksizes. We have provide sample parameter file that would allow you to leverage this new template, called basic-parameters-multisvrdeploy-cseinitdisk.json