# Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual LAN – Part 1

In this post, I want to walk you through creating of a Service Fabric Cluster in Microsoft Azure, and make it a secure one. Moreover, we do not use the public IP address for the load balancer which is the default configuration of the Azure Portal when you are creating a Service Fabric. We will assign an internal IP address to the load balancer and place the Service Fabric on an Existing VNET in your Azure subscription.

I have divided this post into two parts:

Part1 - Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual LAN

Part2 - Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual LAN

For a general understanding of the Service Fabric, you can read this article:

<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview>

For a comparison of the Azure Service Fabric and Azure Cloud Services, you can read my former post:

<http://www.sharepointjunkies.com/azure-service-fabric-vs-azure-cloud-services-for-cloud-native-applications/>

Microsoft has a couple of articles on TechNet and MSDN regarding the steps involved in creating a Service fabric and customize it in a way that you can place it on your existing virtual network. In this article, I try to consolidate all those posts and walk you through this, step by step.

<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-creation-via-portal>

<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-patterns-networking>

## Steps highlights

1. Create the keyvault and certificates
2. Generate the default Service Fabric ARM Template from the Azure portal
3. Modify the template.json file
4. Deploy the modified template to the Azure subscription

## Assumptions

* You have an Azure subscription
* Azure Powershell is setup in your box and you have a basic knowledge of Powershell cmdlets
* You have an existing VNET with a subnet to assign it to the Service Fabric in your Azure subscription (these values are test values for my virtual network):
  + Resource Group Name = test\_res\_12
  + VNET Name = Server-VNET
  + Subnet Name = Server-Subnet
  + Subnet address space = 172.16.0.0/24

## Step 1 – create the keyvault and certificates

For a general information regarding the Azure keyvault, you can read this link:

<https://docs.microsoft.com/en-us/azure/key-vault/key-vault-whatis>

To have a secure cluster in your Service Fabric cluster in Microsoft Azure, you need to create a certificate for your cluster and upload it to the Azure keyvault. Azure Service Fabric uses X.509 certificates to secure a cluster. In the production environment, your certificate file should be signed by an authoritative Certificate Authority. If you are creating this environment for your development or testing purposes, you can use a self-signed certificate to encrypt your cluster.

The process of creating certificate and upload it to keyvault to get the information you want is quite cumbersome in Powershell. Fortunately, there is a prepackaged Powershell cmdlets available for this purpose that you can use it. Here is the link:

<https://github.com/ChackDan/Service-Fabric/tree/master/Scripts/ServiceFabricRPHelpers>

you should clone or download the whole package from the github on your local computer, and then browse to the location and import the module:

Import-Module .\ServiceFabricRPHelpers.psm1

If you get an error when you want to run this cmdlet, it is possible that the file is being blocked because you have downloaded it from the public internet. You should right click on the file and unblock the psm1 and dll files in the folder and then import the module.

Figure – unblock the file

After that, run these commands to create a new self-signed certificate and upload it to Azure keyvault and get the required values:

$ResouceGroup = "test\_res\_14"

$VName = "liqsfvault"

$SubID = <your subscription id>

$locationRegion = "westus2"

$newCertName = "sfcert1"

$dnsName = "\*.liquwaresf.westus.cloudapp.azure.com" #The certificate's subject name must match the domain used to access the Service Fabric cluster.

$localCertPath = "C:\MyCertificates" # location where you want the .PFX to be stored

Invoke-AddCertToKeyVault -SubscriptionId $SubID -ResourceGroupName $ResouceGroup -Location $locationRegion -VaultName $VName -CertificateName $newCertName -CreateSelfSignedCertificate -DnsName $dnsName -OutputPath $localCertPath

It asks you for a password for the private key encryption. For the vault name, you should come up with a unique name. and, pay attention to the dns name to modify it in a way that suits you in the future. You can use wild character \* to create a wildcard certificate for your Service Fabric cluster. Moreover, the location of the keyvault should be the same as the Service Fabric cluster that you want to create (in this example, westus2)

After you run these cmdlets successfully, you will get the values of these items:

* CertificateThumbprint
* SourceVault
* CertificateURL

keep them in a text file. We need these values in the next steps.

## Step 2 – generate the default Service Fabric ARM template

In this step, we use the Azure Portal to create a new Service Fabric cluster. We will fill out all the parameters and going to the end of the process. At the very last step, instead of creating the Service Fabric Cluster, we will download the ARM template for further modifications.

Login to the Azure portal and click on the new and search for the ‘service fabric’. Then click the Create button.

Enter the basic configuration parameters.

In the next step, select the node type count, and then configure each node type accordingly. For the first node type in your service fabric cluster, you need at least a scaleset of 5 instances of virtual machines.

Click ok and go to the step 3 for the security parameters. Click on the Secure mode and copy the values of the Powershell cmdlet output which you had run in the previous step and click ok.

In the step 4 Summary, you see the link to download the template.

Click on that to download the template.

In the next page you see the generated ARM template. Click on the download to download the ARM template.

A template.zip file will be downloaded in your local machine. Unzip it. There are multiple files with different formats. We need template.json to continue work on.

# Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual network – Part 2

This is the second part of the post in this series that I am talking about the implementation of a customized Service Fabric on Azure. The default functionality of the Microsoft Azure to create Service Fabric via portal does not allow you to use an existing virtual network or an internal load balancer. In this blog post I walk you through creating a secure Service Fabric on Microsoft Azure on an existing virtual network with an internal load balancer.

I have divided this post into two parts:

Part1 - Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual LAN

Part2 - Creating a secure Service Fabric Cluster in Microsoft Azure with internal load balancer on an existing virtual LAN

In the first part of this post, we have talked about the basic information regarding Service Fabric implementation, prerequisites and the first two steps of the process. Now we continue on that basis.

## Step 3 - Modifying the template.json

As you see, the basic service fabric cluster creation via portal does not let you to add the scaleset to an existing VNET, or have an internal load balancer. We modify the template file to have these features available to us.

First, make a copy of the template.json to work on. Open it in your favorite code editor application (mine is Visual Studio Code!)

Then change these items in your newtemplate.json file

### Change to the ‘parameters’ section of the ARM template

You have to change some parameters in your ARM template to suit your needs.

Find the subnet0Name in the parameters section of the ARM template (around line 42 – all of the line numbers may change in the future due to azure new releases). Change the default value to the name of the existing subnet in your existing vnet (Server-Subnet)

Change the subnet0Prefix parameter (line 46) default value to the address space of your subnet (172.16.0.0/24)

Then add two new parameters after that, for the resource group name and name of your VNET:

"existingVNetRGName": {

"type": "string",

"defaultValue": "test\_res\_12"

},

"existingVNetName": {

"type": "string",

"defaultValue": "Server-VNET"

},

Change “test\_res\_12” and “Server-VNET” values accordingly to reflect your own environment in the above code.

Then, because we want to use internal load balancer, we no longer need the dns name in the parameters section. So comment out the dnsName parameter since we no longer need it. (line 100)

And also you can comment out the virtualNetworkName, addressPrefix since we are using an existing VNET (line 92)

For the internal load balancer, you have to add an address for it (put it at the line 238 at the end of all parameters):

"internalLBAddress": {

"type": "string",

"defaultValue": "172.16.0.250"

Make sure to use the IP address of your own environment here.

That’s all with the parameter and you need no further changes. But since the ARM template for the Service Fabric use some default names for the parameters, if you want to have more customized environment, it is good to change the following names as well:

* clusterName (Line 11): the default name is Cluster. It is good to change it to a more unique name.
* adminUserName (line 79): the username to login to the instances of the scaleset. The default is testadm. You can change it to something else
* vmNodeType0Name (line 229): the node type name

### Change to the “variables” section of the ARM template

We have to change the vnetID variable to use the existing VNET (line 249)

/\*old "vnetID": "[resourceId('Microsoft.Network/virtualNetworks',parameters('virtualNetworkName'))]",\*/

"vnetID": "[concat('/subscriptions/', subscription().subscriptionId, '/resourceGroups/', parameters('existingVNetRGName'), '/providers/Microsoft.Network/virtualNetworks/', parameters('existingVNetName'))]",

### Change to the “resources” section of the ARM template

We don’t need to create a VNET since we already have it. So you have to comment out the VNET creation section in the resources. Search for “Microsoft.Network/virtualNetworks”

In the resources section and comment it out (from line 300 to 325)

Then you have to remove the dependency on the VNET creation in other section. So comment out the the virtual network from the ‘dependsOn’ attribute of the Microsoft.Compute/virtualMachineScaleSets

Search for the Microsoft.Compute/virtualMachineScaleSets (line 595) then on the dependsOn section comment out the virtual network:

Because we do not need public ip address to be created, search for Microsoft.Network/publicIPAddress in the resources section and comment out the creation of this resource type (line 326 to 341)

Remove the IP address dependsOn attribute of Microsoft.Network/loadBalancers, so you don't depend on creating a new IP address. Add the virtual network dependsOn attribute because the load balancer now depends on the subnet from the virtual network (line 348)

Change the load balancer's frontendIPConfigurations setting from using a publicIPAddress, to using a subnet and privateIPAddress. privateIPAddress uses a predefined static internal IP address. To use a dynamic IP address, remove the privateIPAddress element, and then change privateIPAllocationMethod to Dynamic. (line 351)

In the Microsoft.ServiceFabric/clusters resource (line 778), change managementEndpoint (line 816)to point to the internal load balancer address. Make sure you use https.

//"managementEndpoint": "[concat('https://',reference(concat(parameters('lbIPName'),'-','0')).dnsSettings.fqdn,':',parameters('nt0fabricHttpGatewayPort'))]",

"managementEndpoint": "[concat('https://',reference(variables('lbID0')).frontEndIPConfigurations[0].properties.privateIPAddress,':',parameters('nt0fabricHttpGatewayPort'))]",

## Step 4 – deploy the modified template to the Azure

Now you have a new modified template.json file that you can deploy to the Azure to create your customized service fabric environment!

You have to run these cmdlets in order to deploy the template.

New-AzureRmResourceGroup -Name test\_res\_16 -Location westus2

New-AzureRmResourceGroupDeployment -Name deployment -ResourceGroupName test\_res\_16 -TemplateFile C:\template\newtemplate.json

Make sure to replace your values for the resource name, location and path to the new template.json file.

It will ask you for some parameters here and you have to enter them:

clusterLocation:

computeLocation:

vmStorageAccountName:

certificateThumbprint:

sourceVaultValue:

certificateUrlValue:

After entering these values it takes a couple of minutes to create all the resources and become ready to use.

Happy using Service Fabric!