V2: Highly Available, Scalable WordPress Web site on Azure

# Previous Architecture

Previously, we had created a WordPress® blog site using the LAMP stack and had it up and running in minutes. All the software was deployed on one Linux machine and external connectivity was established using the Azure DNS Zone and a static public IP address component. It was and is a simplistic approach to get the WordPress® web site up and running. The architecture looked like following:



As good as the previous approach is, it suffers from some pitfalls;

* The architecture lacks the high-availability, scalability that is the hall mark of any cloud deployment. It has single point of failures and not much resiliency built in.
* The Azure Web plan – although provisioned, is not used – therefore I am unnecessarily paying for a service that I am not using.
* I was using a bigger machine (B2S) to deliver the complete solution. This VM was prices @ ~ 25£ per month. Including the Azure web plan @ £7£ per month, I am was paying way above what is commercially above to host a reliable WordPress hosting site.
  + The aim is to get the hosting cost to be less than £10 per month.

# New Design

In this second iteration of this exercise, we are going to deploy a highly scalable, robust solution which will leverage the cloud capabilities of scalability, resiliency and of back-up and restore. We will also be removing the Azure Service Plan and App Services from our final deployment.

## Target Architecture



In the target architecture – I’ve separated the previous single instance architecture into front-end servers (Apache web server, WordPress installation) and back-end database server (MySQL database). The target architecture will increase the availability and reliability of the front-end service using the Azure Availability Sets capability. Higher availability is also achieved by spreading the incoming request across multiple virtual machines as it will utilize the Azure Load balancer in front of the network interfaces of each VMs. The deployment will be utilising subnets and network security groups (NSGs) to provide secure access. Subnets in Azure are used to control network flow and acts as a security boundary. Back-end database will accept requests only from the front-end sub-net.

Database layer – using MySQL – is still a single instance in our architecture and it can still be a single point of failure. Regular back-up and restore capability of Azure will be utilised to provide a high availability solution – but there is still a possibility that the solution is unavailable for a certain time when the database is being restored. Given the expected load and cost involved, this risk is deemed to be justified.

Let’s start the engines and create the new target architecture. Initially, I’ve removed all the existing components except the following

1. DNS Zone

If you recall, I bought the domain, *ninadkanthi.co.uk* from GoDaddy.com® domain registrar and later changed the default NS records specified there to the four records of the Azure DNS name servers. The steps on how to create this is described [here](https://blogs.ninadkanthi.com/index.php/getting-wordpress-site-up-and-running-in-azure/) under the section ‘Static IP Address, DNS Name Label’. The component within Azure that I am starting from is



## Script, Folder structure and execution principles.

All the scripts related to this project are copied under the folder ‘[V2](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/tree/master/scripts/V2)’. The folder contains a master – controller script – [install.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/install.ps1). This script invokes secondary scripts to create various components under Azure. To uninstall, a similar [uninstall.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/uninstall.ps1) script is also stored under the same folder. If you want to create all the Azure components in one go, just execute the [install.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/install.ps1). similarly, to clean the Azure resources, execute [uninstall.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/uninstall.ps1) script.

All the scripts make use of common parameters file – [parameters.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/parameters.ps1) – which contains all the configuration information for the solution. If you want to change name etc for your deployment, edit this parameters file.

All the scripts make **Login-AzAccount** call and to avoid duplication of login invocations, checks if the last access for expiry and reuses the cached credentials if it can.

Wherever possible, Azure portal will not be used to create, modify or delete the components. To deliver robustness, reliability and certainty of the solution, most of the activities would be done via scripts. Azure portal (<https://portal.azure.com> ) is to be used to validate the script execution results manually – although a non-error run of the script also implicitly applies success.

# Steps to create the solution

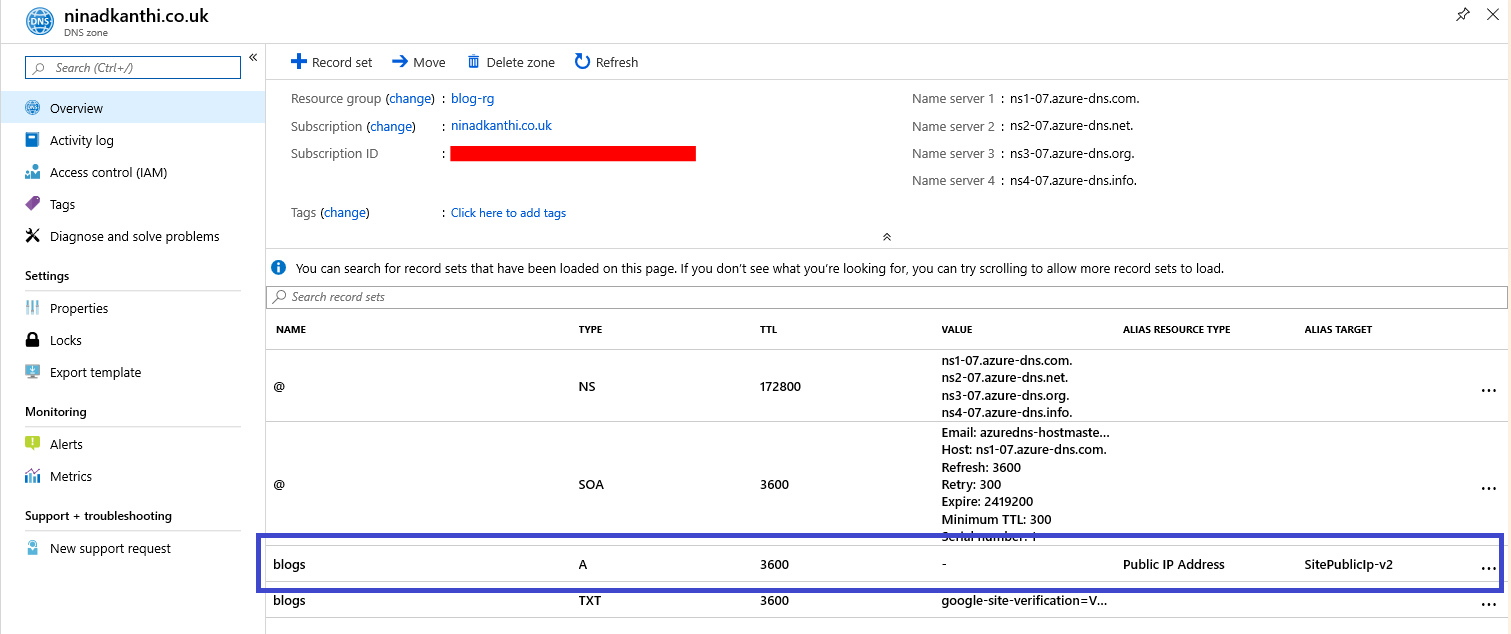
# Create Static IP and ‘A’ record inside DNS record set

As the first step, I will create a static public IP and establish connection with the existing DNS zone by creating an ‘A’ record inside the DNS record set. The scripts to execute is ‘[1-create-public-IP-connect-DNS.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/1-create-public-IP-connect-DNS.ps1)’. The script created a static public IP which bridges the connection between the external internet and our solution.

PowerShell cmdlets used during the script:

* Get-AzPublicIpAddress
* New-AzPublicIpAddress
* Get-AzDnsZone
* New-AzDnsZone
* Get-AzDnsRecordSet
* New-AzDnsRecordSet

[Validation Tests]: Log on to the Azure portal – <https://portal.azure.com>. Manually validate that the component ‘SitePublicIp-v2’ of type ‘Public IP address’ has been created. Also validate that an ‘A’ record associating the created public IP with the DNS zone has been created like shown. Portal navigation: All resources 🡪 ‘ninadkanthi.co.uk DNS Zone’ 🡪 Overview. It should show following added record set.



After the execution of previous step, we’ve achieved the following of our final target architecture.



# Next Step: Creation of V-Net, Sub-net and NSGs

I will create the network infrastructure required and configure security from the very start by plugging in the network security groups within network subnets. The scripts to execute is ‘[2-create-network.ps1](https://github.com/ninadkan/WordPressBlogOnLampAndAzure/blob/master/scripts/V2/2-create-network.ps1)’. The script creates VNET, subnets and NSGs.

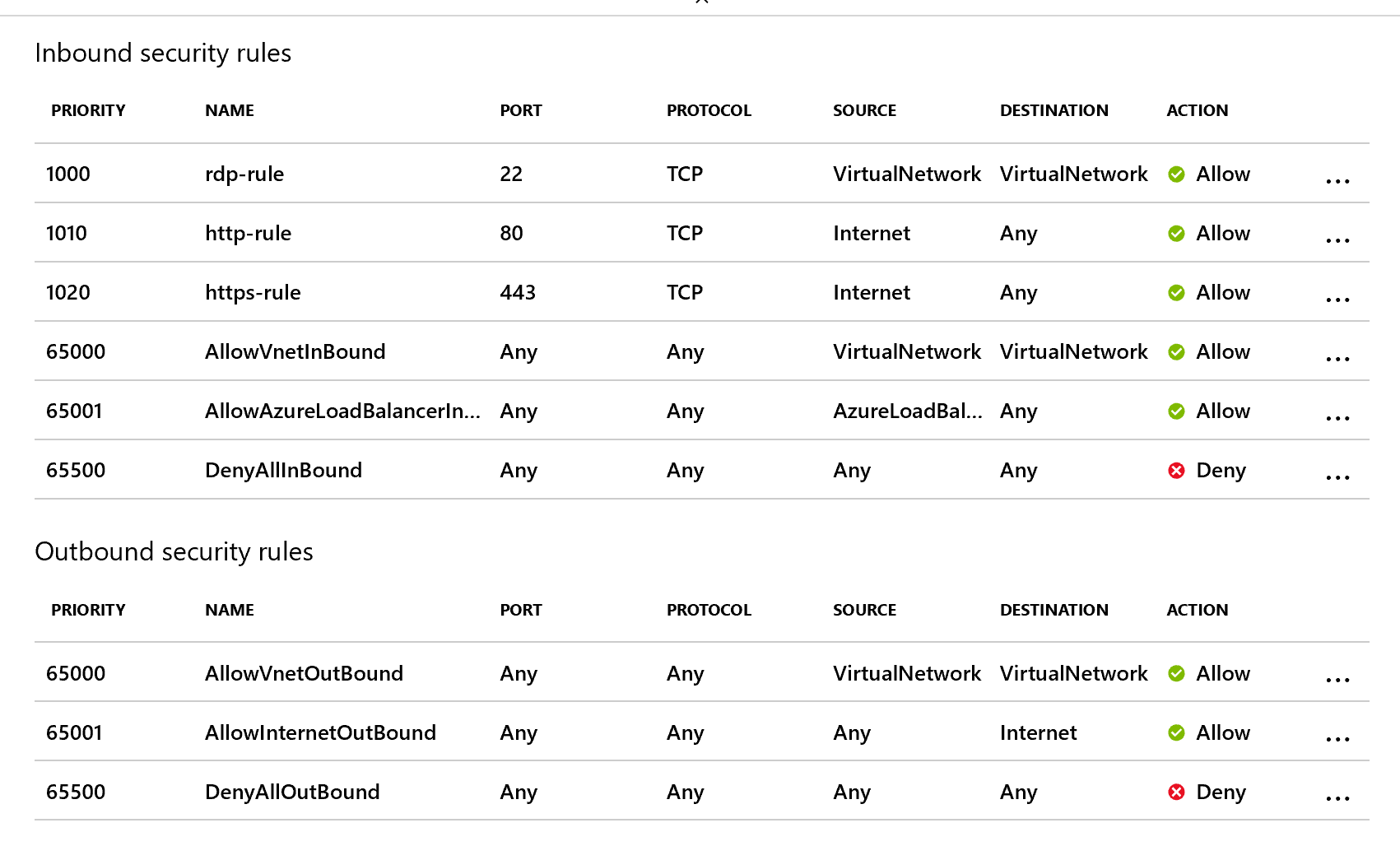
PowerShell cmdlets used during the script:

* Get-AzNetworkSecurityGroup
* New-AzNetworkSecurityRuleConfig
* New-AzNetworkSecurityGroup
* Get-AzVirtualNetworkSubnetConfig
* Add-AzVirtualNetworkSubnetConfig
* Get-AzVirtualNetwork
* New-AzVirtualNetwork
* Set-AzVirtualNetwork
* Set-AzVirtualNetworkSubnetConfig

[Validation Tests]: Log on to the Azure portal and manually validate following components have been created

|  |  |
| --- | --- |
| Name | Type |
| nsg-backend-v2 | Network security group |
| nsg-frontend-v2 | Network security group |
| blogs-vnet | Virtual network |
| subnet-frontend-v2 | Subnets |
| subnet-backend-v2 | Subnets |

The Network security group ‘nsg-frontend-v2’ should be configured with following rules.



Important to emphasize that only two ports are open for internet. RDP-connection is only allowed for devices connected within VNet only. After the execution of previous step, we’ve achieved the following of our final target architecture.

