Deploying VM-Series Firewalls to Secure Internet facing Web Workloads in Azure

# Introduction

This document provides background and instruction on how to use VM-Series firewalls to secure internet facing workload in Azure. The document is a companion to the ARM templates that can be found in the ARM template repository found at: <https://github.com/fullscale180/PAN>

Table of Contents

[Introduction 1](#_Toc481765929)

[Architectural Overview 2](#_Toc481765930)

[Inbound Firewall Details 3](#_Toc481765931)

[Outbound Firewall Details 4](#_Toc481765932)

[Combined Inbound and Outbound Firewall 5](#_Toc481765933)

[Azure Resources 6](#_Toc481765934)

[ARM Templates 6](#_Toc481765935)

[Resource Groups 6](#_Toc481765936)

[Virtual Networks 6](#_Toc481765937)

[Load Balancers 7](#_Toc481765938)

[VM-Series Firewall 7](#_Toc481765939)

[Network Interface 8](#_Toc481765940)

[Inbound firewall network interface attachments 8](#_Toc481765941)

[Outbound firewall network interface attachments 8](#_Toc481765942)

[Application Backend 8](#_Toc481765943)

[Storage 8](#_Toc481765944)

[Template Deployment 8](#_Toc481765945)

[Deployment Process 8](#_Toc481765946)

# Architectural Overview

The VM-Series firewall protects web workloads by enforcing security policy on both inbound and outbound internet traffic. The design described here uses separate load-balanced firewalls for inbound and outbound traffic. Each firewall consists of two or more VM-Series firewalls in an availability set so they can be independently managed and scaled in or out to accommodate load.

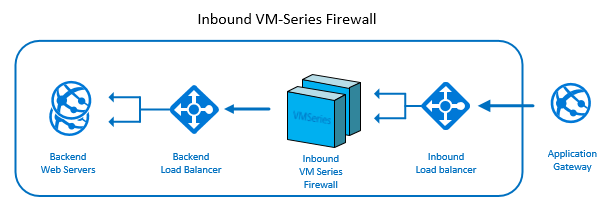
At a high level, Inbound traffic from the application gateway is received by the inbound load balancer which distributes the load to an instance of the inbound VM-Series firewall (see Figure 1). The firewall applies security policy and routes secure traffic to the backend load balancer which distribute the load to an instance of the backend web workload.

Figure 1- Inbound VM-Series firewall

Outbound traffic originates from the application backend then passes through the outbound load balancer which distributes the load to an instance of the outbound VM-Series firewall (see Figure 2). The firewall applies security policy and routes secure traffic out to internet.

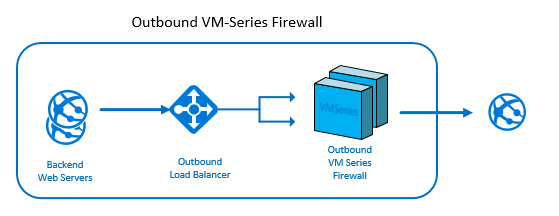


Figure 2 - Outbound VM-Series Firewall

The design described here uses a single Azure virtual network. Having separate inbound and outbound firewalls makes it possible for them to be managed and scaled independently.

## Inbound Firewall Details

In this section we'll look in detail at the specific Azure resources that make up the Inbound VM-Series firewall shown in Figure 3.

The inbound firewall architecture is shown in the figure below. All inbound traffic for the web workload is directed to the public IP defined for the application (a1-gateway-ip). The public IP is assigned an IP address and DNS name of your choice. Traffic arriving on the Public IP is directed to the applications inbound load balancer (a1-inbound-lb) where it is distributed an instance of the inbound firewall (a1-inseries-vmn). All instances of the inbound firewall are created within the same availability set (a1-firewall-avset).

Each VM-Series Firewall has a separate network interfaces in the Trust, Untrust, and Mgmt subnets. The network interface in the Untrusted subnet is attached to the backend of the inbound load balancer. As the inbound load balancer receives traffic from the public IP, it distributes that traffic to one of the VM-Series firewalls. The firewall analyzes the traffic and routes secure traffic through to application over the network interface in the Trust subnet. Each instance of the VM-Series firewall has an additional Public IP (a1-inmgmt-pipn) for remote management which is attached to the network interface in the Mgmt subnet.

The web workload consists of 2 or more backend VMs (a1-backend-vmn) within a common availability set (a1-backend-avset). Traffic that makes it though the inbound firewall is routed to a backend load balancer (a1-backend-lb) which distributes load to a backend VM instance. Each backend VM has a single network interface attached to the backend pool of the backend load balancer.

The inbound VM-Series firewalls and backend VMs all use a common storage account to host VHDs (a1-storage) and the entire deployment is contained within a single resource group (App1).

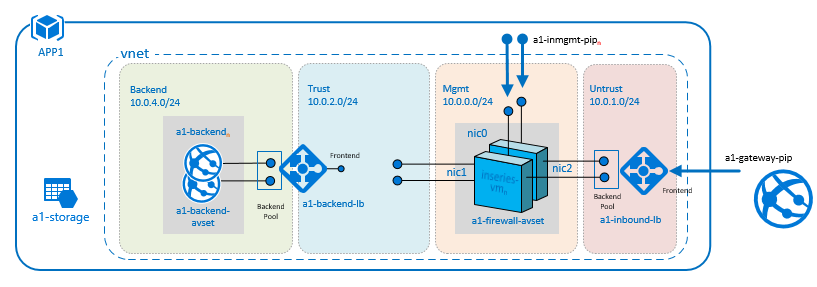


Figure 3 - Inbound VM-Series Firewall Details

## Outbound Firewall Details

In this section well look in detail at the specific Azure resources that make up the outbound VM-Series firewall shown in Figure 4.

The outbound firewall architecture is shown in the figure below. All outbound traffic originates from the VMs that make up the application backend and is routed through the outbound firewall before going to the internet.

The Azure virtual network includes a routing table (route-rt) with a user defined route (UDR) that directs all outbound network traffic within the virtual network to the outbound load balancer (outbound-lb). Network traffic that arrives at the outbound load balancer is distributed to an instance of the outbound firewall (outseries-vmn). All instances of the outbound firewall are created within the same availability set (firewall-avset).

Like the inbound firewall, each instance of the outbound Firewall has a separate network interface in the Trust, Untrust, and Mgmt subnets. The network interface in the Trusted subnet is attached to the backend of the outbound load balancer. As the outbound load balancer receives traffic from the backend VMs, it distributes that traffic to one of the outbound firewalls. The firewall analyzes the traffic and routes secure traffic through to the Internet over the network interface in the Untrust subnet. Each instance of the outbound VM-Series firewall has an additional Public IP (outmgmt-pipn) for remote management which is attached to the network interface in the Mgmt subnet.

The outbound VM-Series firewalls all use a common storage account to host VHDs (pan-storage) and the entire deployment is contained within a single resource group (pan).

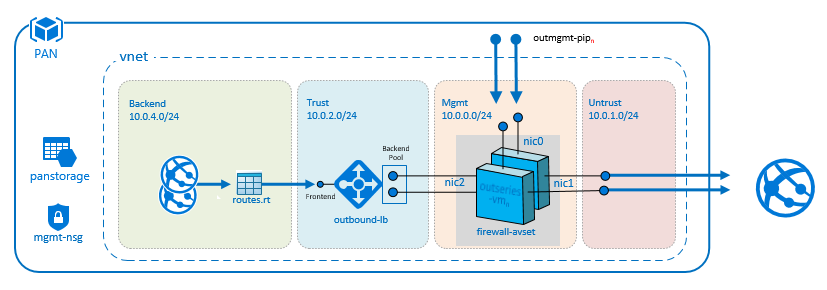


Figure 4- Outbound VM-Series Firewall Details

## Combined Inbound and Outbound Firewall

The design described above can be combined to provide both inbound and outbound protection for a single application as shown in Figure 5.

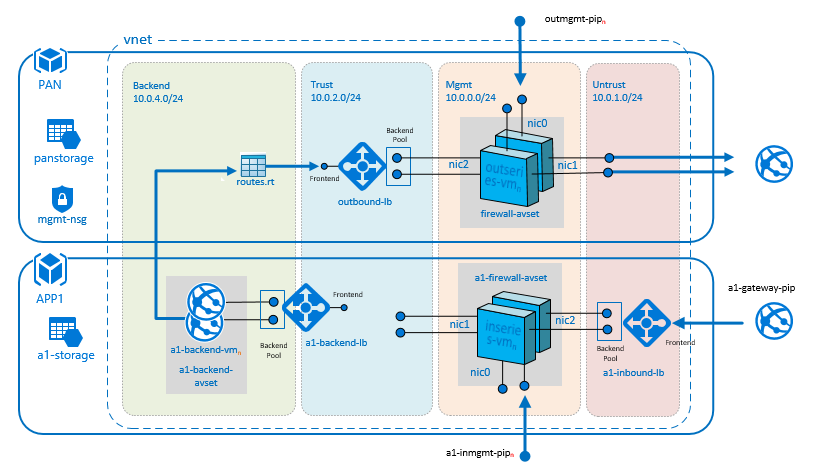


Figure 5 - Combined Inbound/Outbound VM-Series Firewall Details serving a single application

Additional applications can be deployed to the same virtual network that share the same outbound fire wall as shown in Figure 6.

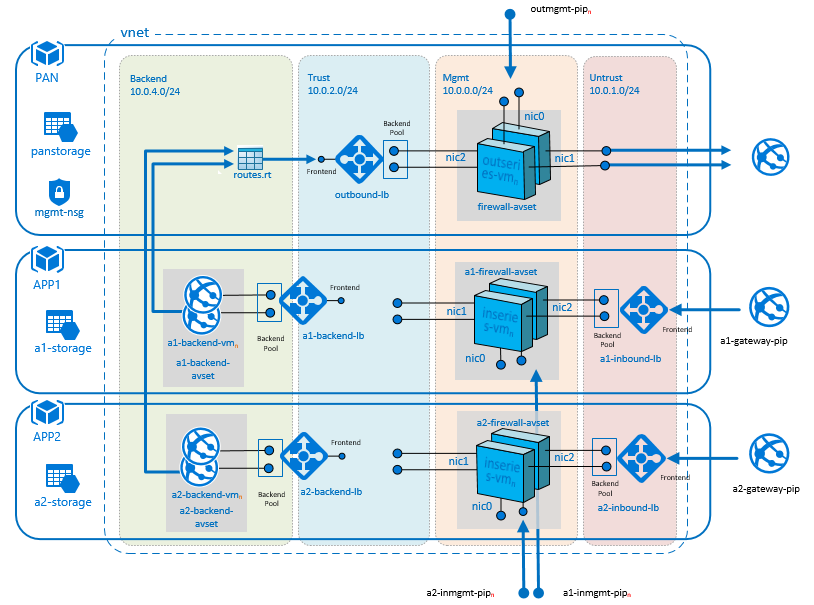


Figure 6 - Combined Inbound/Outbound VM-Series Firewall serving multiple applications

# Azure Resources

## ARM Templates

The design described here can be easily deployed with the Azure Resource Manager (ARM) templates referenced at the beginning of this document. There are separate templates for deploying the inbound and outbound firewalls. The **AzureDeployInfra.json** template is used to deploy the virtual network, the outbound load balancer and outbound firewall. The **AzureDeployApp.json** template is used to deploy the application backend, backend load balancer, inbound firewall, and inbound load balancer. The **AzureDeployApp.json** template can be used repeatedly to install multiple applications the same virtual network or it can be customized to install different application backends within the same virtual network.

## Resource Groups

The firewall infrastructure and each application are deployed in separate Azure resource groups making it easy to add or remove applications as necessary. You can also assign unique permissions to each resource group.

## Virtual Networks

An Azure virtual network provides network isolation from other workloads within the same data center. The design shown here uses a single Azure virtual network with 4 subnets. The virtual network as well as the individual subnets names and addresses can be customized by overriding the default parameters passed to the **AzureDeployInfra.json** template. The default names and addresses are shown in the table below and can be customized by overriding the default parameters.

|  |  |
| --- | --- |
| Name | *Default Address Space* |
| Vnet (virtual network) | 10.0.0.0/16 |
| Mgmt (subnet) | 10.0.0.0/24 |
| Untrust (subnet) | 10.0.1.0/24 |
| Trust (subnet) | 10.0.2.0/24 |
| Backend (subnet) | 10.0.3.0/24 |

The virtual network also includes a custom routing table (shown below) with a user defined route that directs all network traffic to the outbound load balancer. This forces all outbound traffic from the application backend through the outbound load balancer. To allow communication between subnets, additional user defined routes would need to be added to the routing table.

|  |  |  |
| --- | --- | --- |
| Destination IP | *Hop Type* | *Next Hop Address* |
| 0.0.0.0/0 | Virtual Appliance | outbound load balancer frontend IP |

## Load Balancers

Azure Load Balancer is a layer 4 (TCP, UDP) load balancer that distributes incoming traffic among healthy instances of services defined in a load-balanced set. Each load balancer consists of a frontend IP and one or more backend pools. There can be multiple network interfaces attached to each backend pool. The role of the load balancer is to route traffic received on the frontend IP one of the interfaces in a backend pool. The load balancers distribution mode determines how traffic is distributed among the network interfaces attached to the backend pool. The default distribution mode provides stickiness within a transport session. It uses a 5-tuple hash to map the source IP, source port, destination IP, destination port, protocol type to a specific backend interface. This ensure that each new session is paired with a new backend interface. For more information on distribute modes see [Configure the distribution mode for load balancer](https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-distribution-mode)

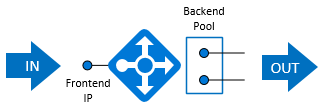


Figure 7 - Load Balancer

The design shown here uses separate load balancers to distribute traffic among instances of the inbound firewall, outbound firewall and application backend virtual machines using the default 5-tuple distribution mode.

## VM-Series Firewall

The inbound and outbound firewalls each consist of multiple VM-Series virtual machines within a common availability set. The name, size, sku and authentication type (password or public key) can be customized for each firewall VM by overriding the default parameter values passed to the **AzureDeployApp.json** template. The default VM configuration use 2 instances of the Standard\_D3 virtual machine which has 4 cores and 14GB of memory. License options include Bring Your Own License (BYOL) and Hourly or Monthly Pay as You Go (PAYG) options which can be selected with template parameters. The admin user name and password should always be set explicitly by overriding the default template parameters.

The name of the availability set and the number of VM within the availability set can also be customized in the same way.

Each virtual machine in the firewall has a dedicated management interface attached to a publicly accessible endpoint. The IP address of each endpoint is assigned dynamically but the DNS name can be explicitly by overriding the default template parameters.

## Network Interface

Each virtual machine in the VM-Series firewall has 3 attached network interface cards (NIC). Each NIC has a unique IP address within its respective subnet. The tables below describe the subnet and function of each NIC.

### Inbound firewall network interfaces

|  |  |
| --- | --- |
| Subnet | Attached to |
| Management subnet | Management interface |
| Untrust subnet | Backend of the Inbound load balancer |
| Trust subnet | Frontend of the backend load balancer |

### Outbound firewall network interfaces

|  |  |
| --- | --- |
| Subnet | Attached to |
| Management subnet | Management interface |
| Trust subnet | Backend of the outbound load balancer |
| Untrust subnet | Out to the internet |

## Application Backend

The application backend consists of one or more virtual machines within a common availability set. The name, size, and authentication type (password or public key) can be customized for each backend VM by overriding the default parameter values passed to the **AzureDeployApp.json** template. The default VM configuration use 2 instances of the Standard\_D1\_V2 virtual machine which has 1 core and 3.5GB of memory. The admin user name and password should always be set explicitly by overriding the default template parameters.

The backend deployment can be completely customized by modifying or replacing the deployBackend.json template.

## Storage

Each resource group includes a separate storage account to storing the virtual hard disks (VHDs) used by the virtual machines within the group. The name and type of storage account can be customized with parameters. The storage account name must be globally unique. The default storage type is Standard LRS which stores 3 redundant copies of data within the same local datacenter.

# Template Deployment

## Deployment Process

You can deploy the solution described in this document with the ARM templates located at: <https://github.com/fullscale180/PAN>. Use the templates to gain a better understanding of the VM-Series firewall in Azure or customized them to suite your own deployment scenario.

The templates in the repository use a nested design where top-level templates invoke a secondary set of templates to deploy the sub parts of the overall solution. There are two top-level templates in the repository: **AzureDeployInfra.json** which is used to deploy the virtual network and outbound firewall and **AzureDeployApp.json** which is used to deploy an application instance and inbound firewall in the same virtual network. The top-level templates use the secondary templates to deploy parts of the overall solution. All templates in the solution are listed in the table below (top-level templates in bold).

|  |  |
| --- | --- |
| Template | Function |
| **AzureDeployInfra.json** | Deploy the virtual network and outbound firewall |
| **AzureDeployApp.json** | Deploy an application instance and inbound firewall |
| deployStorage.json | Deploy a storage account |
| deployVnet.json | Deploy a virtual network |
| deployNsg.json | Deploy a network security group |
| deployFirewall.json | Deploy an instance of a VM-Series firewall |
| deployBackend.json | Deploy the application backend |
| deployLoadBalancer.json | Deploy the load balancer |
| deployVm-password-bs-yes.json | Deploy a virtual machine with password auth and bootstrap disk |
| deployVm-password-bs-no.json | Deploy a virtual machine with password auth and no disk |

### Web-based Deployment

The simplest way to deploy the solution is to use the Deploy to Azure buttons on the repository landing page. The deploy buttons will initiate a web based deployment where the deployment parameters



The landing page has two deployment buttons. Use the first button to invoke **AzureDeployInfra.json** and the second button to invoke **AzureDeployApp.json.** When using the web-based deployment, you’ll be prompted for parameter values on the Custom Deployment page.

When deploying **AzureDeployInfra.json**, choose the subscription, resource group name and location in the Basics section then provide a unique name for the storage account and outbound firewall management endpoint in the setting section. Remember the name you chose for the infrastructure resource group in this section. It will be needed again when deploying the application in the next section. You can accept the defaults for the remainder of the settings then accept the terms and click Purchase to launch the deployment.

When deploying **AzureDeployApp.json**, again you’ll choose the subscription, resource group name and location in the Basics section. The subscription and location should be the same as those used when deploying the infrastructure. The resource group can be different if you want to isolate the application resources from other parts of the solution. Each application must have a unique application prefix setting. The prefix use used when naming application resources in order to ensure uniqueness. The Virtual Network RG setting must match the name of resource group used when deploying **AzureDeployInfra.json.**

You can use **AzureDeployApp.json** again to deploy another application in the same virtual network sharing the same outbound firewall. Each application must have a unique application prefix and must use the same value for the Virtual Network RG setting.

You can also use PowerShell to deploy the solution as shown below.

### PowerShell Deployment

You can also use the PowerShell script **AzureDeploy.ps1** from the repository <https://github.com/fullscale180/PAN> to launch a deployment. The script will automate the following deployment steps:

1. Login in to your Azure account

Login-AzureRmAccount

1. Select an active subscription subscription (provide *SubscriptionName* below).

Get-AzureRmSubscription -SubscriptionName *<SubscriptionName>* | Select-AzureRMSubscription

1. Set variables and parameters for infrastructure deployment. Choose values in *italics* below

$location = "*westus2*"

$deployName = "AzureDeploy"

$infaRGName = "INFRA"

$templateFile = "azureDeployInfra.json"

$parameterObject = @{"storageName" = "*mypanstorage*"; "mgmtPublicIPDns" = "*mypanmgmt*"}

1. Create new infrastructure resource group

$rg = New-AzureRMResourceGroup -name $infaRGName -location $location

1. Deploy infrastructure to new resource group

New-AzureRMResourceGroupDeployment -ResourceGroupName $infaRGName -Name $deployName -TemplateFile $templateFile -TemplateParameterObject $parameterObject

1. Set variables and parameters for application deployment. Choose appPrefix below

|  |
| --- |
|  |

$appRGName = "APP1"

$templateFile = "azureDeployApp.json"

$parameterObject = @{ "appPrefix" = "*a1*"; "virtualNetworkRG" = $infaRGName}

1. Create new application resource group

$rg = New-AzureRMResourceGroup -name $appRGName -location $location

1. Deploy application to new resource group

New-AzureRMResourceGroupDeployment -ResourceGroupName $appRGName -Name $deployName -TemplateFile $templateFile -TemplateParameterObject $parameterObject

|  |  |
| --- | --- |
|  |  |

Repeat steps 6, 7, & 8 with a unique resource group as appPrefix for each application