Contents

- Application Gateway documentation
- Overview
  - About application gateways
  - Application Gateway v2
  - Well-Architected review of Application Gateway
- Quickstarts
  - Create Application Gateway - Portal
  - Create Application Gateway - PowerShell
  - Create Application Gateway - Azure CLI
  - Create Application Gateway - Bicep
  - Create Application Gateway - ARM template
  - Create Application Gateway - Terraform
- Tutorials
  - Secure with SSL
  - Host multiple sites
  - Route by URL
  - Redirect web traffic
  - Autoscaling and zone redundant
  - Ingress Controller add-on for AKS (Greenfield)
  - Ingress Controller add-on for AKS (Brownfield)
- Samples
  - Azure PowerShell
  - Azure CLI
  - Resource Manager templates
- Concepts
  - Support for working remotely
  - Basics
    - Application Gateway features
    - How Application Gateway works

- Application Gateway components
- Configuration
  - Overview
  - Infrastructure
  - Front-end IP address
  - Listeners
  - Request routing rules
  - HTTP settings
- High traffic support
- Autoscaling v2 SKU
- Routing
  - Multiple site hosting
  - URL routing
  - Redirection
  - Rewrite HTTP headers and URL
- Security
  - Security baseline
  - Private Link
- SSL
  - SSL termination and end to end SSL
  - SSL policy overview
  - Mutual authentication
  - Using Key Vault
- Ingress for AKS
- Health monitoring
  - Monitor Azure Application Gateway
  - Health probe
  - Diagnostic logs and backend health
  - Metrics
- WebSocket support
- Understanding pricing
- FAQ

How-to guides
- Host single site
  - Azure PowerShell
  - Azure CLI
- Configure internal load balancer
  - Azure portal
  - Azure PowerShell
- Configure SSL
  - SSL termination
    - SSL termination - PowerShell
    - SSL termination - Azure CLI
    - Configure Key Vault - PowerShell
    - Configure Key Vault - Portal
  - End-to-end SSL
    - End-to-end SSL - Portal
    - End-to-end SSL - PowerShell
  - Mutual authentication
    - Mutual authentication - Portal
    - Mutual authentication - PowerShell
  - Manage certificates
    - Certificates for the backend
    - Renew certificates
    - Generate self-signed certificates for backend
    - Trusted client certificates
  - SSL policy
    - Configure SSL policy
    - Listener specific SSL policy
- Ingress for AKS
  - Ingress for AKS via Helm (Brownfield)
  - Ingress for AKS via Helm (Greenfield)
  - Migrate from Helm deployment to AKS add-on
  - Disable and re-enable AKS Ingress Controller add-on

- Enable cookie affinity
- Enable multiple namespace support
- Use private IP for internal routing
- Add health probes to AKS pods
- Use Application Gateway to expose AKS service over HTTP/HTTPS
- Upgrade ingress controller using Helm
- Use LetsEncrypt.org with Application Gateway
- Expose WebSocket to Application Gateway
- Autoscale AKS pods with Application Gateway metrics

Route by URL
- Azure PowerShell
- Azure CLI

Host multiple sites
- Azure PowerShell
- Azure CLI

Redirect traffic
- External traffic
  - Azure PowerShell
  - Azure CLI
- HTTP to HTTPS
  - Azure portal
  - Azure PowerShell
  - Azure CLI
- Redirect internal traffic
  - Azure PowerShell
  - Azure CLI
- Redirect web traffic using Azure PowerShell

Rewrite HTTP headers and URL
- Header rewrite
  - Azure portal
  - Azure PowerShell
  - Create and rewrite HTTP headers

- URL rewrite
  - Azure portal
- Configure App Service
- Configure custom probes
  - Portal
  - Classic PowerShell
  - Resource Manager PowerShell
- Troubleshoot
  - ILB with an App Service Environment
  - App service issues
  - Session affinity issues
  - Bad Gateway (502) errors
  - HTTP response codes
  - Mutual authentication
  - Ingress for AKS
  - Resource Health
  - Use Log Analytics
  - Backend health issues
  - Key Vault errors
  - Disabled listeners
- Proxy buffer configuration
- Custom error pages
- Migrate from v1 to v2
- Configure alerts
- Classic to Resource Manager
- Configure Private Link

Reference

- Monitoring data
- Ingress for AKS annotations
- Azure CLI
- Azure PowerShell
- .NET

- Java
- Node.js
- Python
- REST
- Azure Resource Manager
- Resource Manager template

Resources

- Author templates
- Azure Roadmap
- Community templates
- Pricing
- Regional availability
- Stack Overflow

What is Azure Application Gateway?

8/22/2022 • 2 minutes to read • Edit Online

Azure Application Gateway is a web traffic load balancer that enables you to manage traffic to your web applications. Traditional load balancers operate at the transport layer (OSI layer 4 - TCP and UDP) and route traffic based on source IP address and port, to a destination IP address and port.

Application Gateway can make routing decisions based on additional attributes of an HTTP request, for example URI path or host headers. For example, you can route traffic based on the incoming URL. So if `/images` is in the incoming URL, you can route traffic to a specific set of servers (known as a pool) configured for images. If `/video` is in the URL, that traffic is routed to another pool that's optimized for videos.

This type of routing is known as application layer (OSI layer 7) load balancing. Azure Application Gateway can do URL-based routing and more.

> **NOTE**
>
> Azure provides a suite of fully managed load-balancing solutions for your scenarios.
>
> - If you are looking to do DNS based global routing and do **not** have requirements for Transport Layer Security (TLS) protocol termination ("SSL offload"), per-HTTP/HTTPS request or application-layer processing, review Traffic Manager.
> - If you need to optimize global routing of your web traffic and optimize top-tier end-user performance and reliability through quick global failover, see Front Door.
> - To do transport layer load balancing, review Load Balancer.
>
> Your end-to-end scenarios may benefit from combining these solutions as needed. For an Azure load-balancing options comparison, see Overview of load-balancing options in Azure.

Features

To learn about Application Gateway features, see Azure Application Gateway features.

Pricing and SLA

For Application Gateway pricing information, see Application Gateway pricing.

For Application Gateway SLA information, see Application Gateway SLA.

What's new

To learn what's new with Azure Application Gateway, see Azure updates.

Next steps

Depending on your requirements and environment, you can create a test Application Gateway using either the Azure portal, Azure PowerShell, or Azure CLI.

- Quickstart: Direct web traffic with Azure Application Gateway - Azure portal
- Quickstart: Direct web traffic with Azure Application Gateway - Azure PowerShell
- Quickstart: Direct web traffic with Azure Application Gateway - Azure CLI
- Learn module: Introduction to Azure Application Gateway

What is Azure Application Gateway v2?

8/22/2022 • 5 minutes to read • Edit Online

Application Gateway is available under a Standard_v2 SKU. Web Application Firewall (WAF) is available under a WAF_v2 SKU. The v2 SKU offers performance enhancements and adds support for critical new features like autoscaling, zone redundancy, and support for static VIPs. Existing features under the Standard and WAF SKU continue to be supported in the new v2 SKU, with a few exceptions listed in comparison section.

The new v2 SKU includes the following enhancements:

- **Autoscaling**: Application Gateway or WAF deployments under the autoscaling SKU can scale out or in based on changing traffic load patterns. Autoscaling also removes the requirement to choose a deployment size or instance count during provisioning. This SKU offers true elasticity. In the Standard_v2 and WAF_v2 SKU, Application Gateway can operate both in fixed capacity (autoscaling disabled) and in autoscaling enabled mode. Fixed capacity mode is useful for scenarios with consistent and predictable workloads. Autoscaling mode is beneficial in applications that see variance in application traffic.

- **Zone redundancy**: An Application Gateway or WAF deployment can span multiple Availability Zones, removing the need to provision separate Application Gateway instances in each zone with a Traffic Manager. You can choose a single zone or multiple zones where Application Gateway instances are deployed, which makes it more resilient to zone failure. The backend pool for applications can be similarly distributed across availability zones.

  Zone redundancy is available only where Azure Zones are available. In other regions, all other features are supported. For more information, see Regions and Availability Zones in Azure

- **Static VIP**: Application Gateway v2 SKU supports the static VIP type exclusively. This ensures that the VIP associated with the application gateway doesn't change for the lifecycle of the deployment, even after a restart. There isn't a static VIP in v1, so you must use the application gateway URL instead of the IP address for domain name routing to App Services via the application gateway.

- **Header Rewrite**: Application Gateway allows you to add, remove, or update HTTP request and response headers with v2 SKU. For more information, see Rewrite HTTP headers with Application Gateway

- **Key Vault Integration**: Application Gateway v2 supports integration with Key Vault for server certificates that are attached to HTTPS enabled listeners. For more information, see TLS termination with Key Vault certificates.

- **Mutual Authentication (mTLS)**: Application Gateway v2 supports authentication of client requests. For more information, see Overview of mutual authentication with Application Gateway.

- **Azure Kubernetes Service Ingress Controller**: The Application Gateway v2 Ingress Controller allows the Azure Application Gateway to be used as the ingress for an Azure Kubernetes Service (AKS) known as AKS Cluster. For more information, see What is Application Gateway Ingress Controller?.

- **Private link**: The v2 SKU offers private connectivity from other virtual networks in other regions and subscriptions through the use of private endpoints.

- **Performance enhancements**: The v2 SKU offers up to 5X better TLS offload performance as compared to the Standard/WAF SKU.

- **Faster deployment and update time** The v2 SKU provides faster deployment and update time as compared to Standard/WAF SKU. This also includes WAF configuration changes.

Unsupported regions

The Standard_v2 and WAF_v2 SKU is not currently available in the following regions:

- UK North
- UK South2
- South Africa West
- China East
- China North
- US DOD East
- US DOD Central
- US Gov Central
- Germany Northeast
- Germany Central
- Qatar Central

Pricing

With the v2 SKU, the pricing model is driven by consumption and is no longer attached to instance counts or sizes. The v2 SKU pricing has two components:

- **Fixed price** - This is hourly (or partial hour) price to provision a Standard_v2 or WAF_v2 Gateway. Please note that 0 additional minimum instances still ensures high availability of the service which is always included with fixed price.
- **Capacity Unit price** - This is a consumption-based cost that is charged in addition to the fixed cost. Capacity unit charge is also computed hourly or partial hourly. There are three dimensions to capacity unit - compute unit, persistent connections, and throughput. Compute unit is a measure of processor capacity

consumed. Factors affecting compute unit are TLS connections/sec, URL Rewrite computations, and WAF rule processing. Persistent connection is a measure of established TCP connections to the application gateway in a given billing interval. Throughput is average Megabits/sec processed by the system in a given billing interval. The billing is done at a Capacity Unit level for anything above the reserved instance count.

Each capacity unit is composed of at most: 1 compute unit, 2500 persistent connections, and 2.22-Mbps throughput.

To learn more, see Understanding pricing.

Feature comparison between v1 SKU and v2 SKU

The following table compares the features available with each SKU.

FEATURE	V1 SKU	V2 SKU
Autoscaling		✓
Zone redundancy		✓
Static VIP		✓
Azure Kubernetes Service (AKS) Ingress controller		✓
Azure Key Vault integration		✓
Rewrite HTTP(S) headers		✓
URL-based routing	✓	✓
Multiple-site hosting	✓	✓
Mutual Authentication (mTLS)		✓
Private Link support		✓
Traffic redirection	✓	✓
Web Application Firewall (WAF)	✓	✓
WAF custom rules		✓
WAF policy associations		✓
Transport Layer Security (TLS)/Secure Sockets Layer (SSL) termination	✓	✓
End-to-end TLS encryption	✓	✓
Session affinity	✓	✓
Custom error pages	✓	✓

FEATURE	V1 SKU	V2 SKU
WebSocket support	✓	✓
HTTP/2 support	✓	✓
Connection draining	✓	✓
Proxy NTML authentication	✓	

NOTE

The autoscaling v2 SKU now supports default health probes to automatically monitor the health of all resources in its back-end pool and highlight those backend members that are considered unhealthy. The default health probe is automatically configured for backends that don't have any custom probe configuration. To learn more, see health probes in application gateway.

Differences from v1 SKU

This section describes features and limitations of the v2 SKU that differ from the v1 SKU.

DIFFERENCE	DETAILS
Authentication certificate	Not supported. For more information, see Overview of end to end TLS with Application Gateway.
Mixing Standard_v2 and Standard Application Gateway on the same subnet	Not supported
User-Defined Route (UDR) on Application Gateway subnet	Supported (specific scenarios). In preview. For more information about supported scenarios, see Application Gateway configuration overview.
NSG for Inbound port range	- 65200 to 65535 for Standard_v2 SKU - 65503 to 65534 for Standard SKU. For more information, see the FAQ.
Performance logs in Azure diagnostics	Not supported. Azure metrics should be used.
Billing	Billing scheduled to start on July 1, 2019.
FIPS mode	These are currently not supported.
ILB only mode	This is currently not supported. Public and ILB mode together is supported.
Net watcher integration	Not supported.
Microsoft Defender for Cloud integration	Not yet available.

Migrate from v1 to v2

An Azure PowerShell script is available in the PowerShell gallery to help you migrate from your v1 Application Gateway/WAF to the v2 Autoscaling SKU. This script helps you copy the configuration from your v1 gateway. Traffic migration is still your responsibility. For more information, see Migrate Azure Application Gateway from v1 to v2.

Next steps

Depending on your requirements and environment, you can create a test Application Gateway using either the Azure portal, Azure PowerShell, or Azure CLI.

- Tutorial: Create an application gateway that improves web application access
- Learn module: Introduction to Azure Application Gateway

Quickstart: Direct web traffic with Azure Application Gateway - Azure portal

8/22/2022 • 9 minutes to read • Edit Online

In this quickstart, you use the Azure portal to create an Azure Application Gateway and test it to make sure it works correctly. You will assign listeners to ports, create rules, and add resources to a backend pool. For the sake of simplicity, a simple setup is used with a public front-end IP address, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines (VMs) in the backend pool.

For more information about the components of an application gateway, see Application gateway components.

You can also complete this quickstart using Azure PowerShell or Azure CLI.

Prerequisites

An Azure account with an active subscription is required. If you don't already have an account, you can create an account for free.

Sign in to the Azure portal with your Azure account.

Create an application gateway

You'll create the application gateway using the tabs on the **Create an application gateway** page.

1. On the Azure portal menu or from the **Home** page, select **Create a resource**. The **New** window appears.
2. Select **Networking** and then select **Application Gateway** in the **Featured** list.

Basics tab

1. On the **Basics** tab, enter these values for the following application gateway settings:

   - **Resource group**: Select **myResourceGroupAG** for the resource group. If it doesn't exist, select **Create new** to create it.

   - **Application gateway name**: Enter *myAppGateway* for the name of the application gateway.

2. For Azure to communicate between the resources that you create, it needs a virtual network. You can either create a new virtual network or use an existing one. In this example, you'll create a new virtual network at the same time that you create the application gateway. Application Gateway instances are created in separate subnets. You create two subnets in this example: one for the application gateway, and another for the backend servers.

> **NOTE**
>
> Virtual network service endpoint policies are currently not supported in an Application Gateway subnet.

Under **Configure virtual network**, create a new virtual network by selecting **Create new**. In the **Create virtual network** window that opens, enter the following values to create the virtual network and two subnets:

- **Name**: Enter *myVNet* for the name of the virtual network.
- **Subnet name** (Application Gateway subnet): The **Subnets** grid will show a subnet named

Default. Change the name of this subnet to *myAGSubnet*.
The application gateway subnet can contain only application gateways. No other resources are allowed.

- **Subnet name** (backend server subnet): In the second row of the **Subnets** grid, enter *myBackendSubnet* in the **Subnet name** column.
- **Address range** (backend server subnet): In the second row of the **Subnets** Grid, enter an address range that doesn't overlap with the address range of *myAGSubnet*. For example, if the address range of *myAGSubnet* is 10.0.0.0/24, enter *10.0.1.0/24* for the address range of *myBackendSubnet*.

Select **OK** to close the **Create virtual network** window and save the virtual network settings.

3. On the **Basics** tab, accept the default values for the other settings and then select **Next: Frontends**.

Frontends tab

1. On the **Frontends** tab, verify **Frontend IP address type** is set to **Public**.
   You can configure the Frontend IP to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP.

> **NOTE**
> For the Application Gateway v2 SKU, there must be a **Public** frontend IP configuration. You can still have both a Public and a Private frontend IP configuration, but Private only frontend IP configuration (Only ILB mode) is currently not enabled for the v2 SKU.

2. Select **Add new** for the **Public IP address** and enter *myAGPublicIPAddress* for the public IP address name, and then select **OK**.

3. Select **Next: Backends**.

Backends tab

The backend pool is used to route requests to the backend servers that serve the request. Backend pools can be composed of NICs, virtual machine scale sets, public IP addresses, internal IP addresses, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you'll create an empty backend pool with your application gateway and then add backend targets to the backend pool.

1. On the **Backends** tab, select **Add a backend pool**.

2. In the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:

   - **Name**: Enter *myBackendPool* for the name of the backend pool.
   - **Add backend pool without targets**: Select **Yes** to create a backend pool with no targets. You'll add backend targets after creating the application gateway.

3. In the **Add a backend pool** window, select **Add** to save the backend pool configuration and return to the **Backends** tab.

4. On the **Backends** tab, select **Next: Configuration**.

Configuration tab

On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule.

1. Select **Add a routing rule** in the **Routing rules** column.

2. In the **Add a routing rule** window that opens, enter *myRoutingRule* for the **Rule name**.

3. A routing rule requires a listener. On the **Listener** tab within the **Add a routing rule** window, enter the following values for the listener:

   - **Listener name**: Enter *myListener* for the name of the listener.

   - **Frontend IP**: Select **Public** to choose the public IP you created for the frontend.

     Accept the default values for the other settings on the **Listener** tab, then select the **Backend targets** tab to configure the rest of the routing rule.

4. On the **Backend targets** tab, select **myBackendPool** for the **Backend target**.

5. For the **HTTP setting**, select **Add new** to add a new HTTP setting. The HTTP setting will determine the behavior of the routing rule. In the **Add an HTTP setting** window that opens, enter *myHTTPSetting* for the **HTTP setting name** and *80* for the **Backend port**. Accept the default values for the other settings in the **Add an HTTP setting** window, then select **Add** to return to the **Add a routing rule** window.

6. On the **Add a routing rule** window, select **Add** to save the routing rule and return to the **Configuration** tab.

7. Select **Next: Tags** and then **Next: Review + create**.

Review + create tab

Review the settings on the **Review + create** tab, and then select **Create** to create the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

Add backend targets

In this example, you'll use virtual machines as the target backend. You can either use existing virtual machines or

create new ones. You'll create two virtual machines as backend servers for the application gateway.

To do this, you'll:

1. Create two new VMs, *myVM* and *myVM2*, to be used as backend servers.
2. Install IIS on the virtual machines to verify that the application gateway was created successfully.
3. Add the backend servers to the backend pool.

Create a virtual machine

1. On the Azure portal menu or from the **Home** page, select **Create a resource**. The **New** window appears.

2. Select **Windows Server 2016 Datacenter** in the **Popular** list. The **Create a virtual machine** page appears.
   Application Gateway can route traffic to any type of virtual machine used in its backend pool. In this example, you use a Windows Server 2016 Datacenter virtual machine.

3. Enter these values in the **Basics** tab for the following virtual machine settings:

   - **Resource group**: Select **myResourceGroupAG** for the resource group name.
   - **Virtual machine name**: Enter *myVM* for the name of the virtual machine.
   - **Region**: Select the same region where you created the application gateway.
   - **Username**: Type a name for the administrator user name.
   - **Password**: Type a password.
   - **Public inbound ports**: None.
4. Accept the other defaults and then select **Next: Disks**.

5. Accept the **Disks** tab defaults and then select **Next: Networking**.

6. On the **Networking** tab, verify that **myVNet** is selected for the **Virtual network** and the **Subnet** is set to **myBackendSubnet**. Accept the other defaults and then select **Next: Management**.
   Application Gateway can communicate with instances outside of the virtual network that it is in, but you need to ensure there's IP connectivity.

7. On the **Management** tab, set **Boot diagnostics** to **Disable**. Accept the other defaults and then select **Review + create**.

8. On the **Review + create** tab, review the settings, correct any validation errors, and then select **Create**.

9. Wait for the virtual machine creation to complete before continuing.

Install IIS for testing

In this example, you install IIS on the virtual machines to verify Azure created the application gateway successfully.

1. Open Azure PowerShell.

   Select **Cloud Shell** from the top navigation bar of the Azure portal and then select **PowerShell** from the drop-down list.

2. Run the following command to install IIS on the virtual machine. Change the *Location* parameter if necessary:

```
Set-AzVMExtension `
  -ResourceGroupName myResourceGroupAG `
  -ExtensionName IIS `
  -VMName myVM `
  -Publisher Microsoft.Compute `
  -ExtensionType CustomScriptExtension `
  -TypeHandlerVersion 1.4 `
  -SettingString '{"commandToExecute":"powershell Add-WindowsFeature Web-Server; powershell Add-
Content -Path \"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"}' `
  -Location EastUS
```

3. Create a second virtual machine and install IIS by using the steps that you previously completed. Use *myVM2* for the virtual machine name and for the **VMName** setting of the **Set-AzVMExtension** cmdlet.

Add backend servers to backend pool

1. On the Azure portal menu, select **All resources** or search for and select *All resources*. Then select **myAppGateway**.

2. Select **Backend pools** from the left menu.

3. Select **myBackendPool**.

4. Under **Backend targets**, **Target type**, select **Virtual machine** from the drop-down list.

5. Under **Target**, select the **myVM** and **myVM2** virtual machines and their associated network interfaces from the drop-down lists.

6. Select **Save**.

7. Wait for the deployment to complete before proceeding to the next step.

Test the application gateway

Although IIS isn't required to create the application gateway, you installed it in this quickstart to verify if Azure successfully created the application gateway.

Use IIS to test the application gateway:

1. Find the public IP address for the application gateway on its **Overview** page.

   Or, you can select **All resources**, enter *myAGPublicIPAddress* in the search box, and then select it in the search results. Azure displays the public IP address on the **Overview** page.

2. Copy the public IP address, and then paste it into the address bar of your browser to browse that IP address.

3. Check the response. A valid response verifies that the application gateway was successfully created and can successfully connect with the backend.

Refresh the browser multiple times and you should see connections to both myVM and myVM2.

Clean up resources

When you no longer need the resources that you created with the application gateway, delete the resource group. When you delete the resource group, you also remove the application gateway and all the related resources.

To delete the resource group:

1. On the Azure portal menu, select **Resource groups** or search for and select *Resource groups*.
2. On the **Resource groups** page, search for **myResourceGroupAG** in the list, then select it.
3. On the **Resource group page**, select **Delete resource group**.
4. Enter *myResourceGroupAG* under **TYPE THE RESOURCE GROUP NAME** and then select **Delete**

Next steps

Tutorial: Configure an application gateway with TLS termination using the Azure portal

Quickstart: Direct web traffic with Azure Application Gateway using Azure PowerShell

8/22/2022 • 7 minutes to read • Edit Online

In this quickstart, you use Azure PowerShell to create an application gateway. Then you test it to make sure it works correctly.

The application gateway directs application web traffic to specific resources in a backend pool. You assign listeners to ports, create rules, and add resources to a backend pool. For the sake of simplicity, this article uses a simple setup with a public front-end IP address, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool.

You can also complete this quickstart using Azure CLI or the Azure portal.

Prerequisites

- An Azure account with an active subscription. Create an account for free.
- Azure PowerShell version 1.0.0 or later (if you run Azure PowerShell locally).

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	Azure CLI Copy Try It
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	Launch Cloud Shell

OPTION	EXAMPLE/LINK
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Connect to Azure

To connect with Azure, run `Connect-AzAccount`.

Create a resource group

In Azure, you allocate related resources to a resource group. You can either use an existing resource group or create a new one.

To create a new resource group, use the `New-AzResourceGroup` cmdlet:

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

For Azure to communicate between the resources that you create, it needs a virtual network. The application gateway subnet can contain only application gateways. No other resources are allowed. You can either create a new subnet for Application Gateway or use an existing one. You create two subnets in this example: one for the application gateway, and another for the backend servers. You can configure the Frontend IP address of the Application Gateway to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP address.

1. Create the subnet configurations using `New-AzVirtualNetworkSubnetConfig`.
2. Create the virtual network with the subnet configurations using `New-AzVirtualNetwork`.
3. Create the public IP address using `New-AzPublicIpAddress`.

> **NOTE**
> Virtual network service endpoint policies are currently not supported in an Application Gateway subnet.

```
$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.21.0.0/24
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.21.1.0/24
New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.21.0.0/16 `
  -Subnet $agSubnetConfig, $backendSubnetConfig
New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Static `
  -Sku Standard
```

Create an application gateway

Create the IP configurations and frontend port

1. Use `New-AzApplicationGatewayIPConfiguration` to create the configuration that associates the subnet you created with the application gateway.
2. Use `New-AzApplicationGatewayFrontendIPConfig` to create the configuration that assigns the public IP address that you previously created for the application gateway.
3. Use `New-AzApplicationGatewayFrontendPort` to assign port 80 to access the application gateway.

```
$vnet   = Get-AzVirtualNetwork -ResourceGroupName myResourceGroupAG -Name myVNet
$subnet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name myAGSubnet
$pip    = Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet
$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip
$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the backend pool

1. Use `New-AzApplicationGatewayBackendAddressPool` to create the backend pool for the application gateway. The backend pool is empty for now. When you create the backend server NICs in the next section, you'll add them to the backend pool.
2. Configure the settings for the backend pool with `New-AzApplicationGatewayBackendHttpSetting`.

```
$backendPool = New-AzApplicationGatewayBackendAddressPool `
  -Name myAGBackendPool
$poolSettings = New-AzApplicationGatewayBackendHttpSetting `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 30
```

Create the listener and add a rule

Azure requires a listener to enable the application gateway for routing traffic appropriately to the backend pool. Azure also requires a rule for the listener to know which backend pool to use for incoming traffic.

1. Create a listener using `New-AzApplicationGatewayHttpListener` with the frontend configuration and frontend port that you previously created.
2. Use `New-AzApplicationGatewayRequestRoutingRule` to create a rule named *rule1*.

```
$defaultlistener = New-AzApplicationGatewayHttpListener `
  -Name myAGListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport
$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultlistener `
  -BackendAddressPool $backendPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you've created the necessary supporting resources, create the application gateway:

1. Use `New-AzApplicationGatewaySku` to specify parameters for the application gateway.
2. Use `New-AzApplicationGateway` to create the application gateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_v2 `
  -Tier Standard_v2 `
  -Capacity 2
New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $backendPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultlistener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku
```

Backend servers

Now that you have created the Application Gateway, create the backend virtual machines which will host the websites. A backend can be composed of NICs, virtual machine scale sets, public IP address, internal IP address, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service.

In this example, you create two virtual machines to use as backend servers for the application gateway. You also install IIS on the virtual machines to verify that Azure successfully created the application gateway.

Create two virtual machines

1. Get the recently created Application Gateway backend pool configuration with `Get-AzApplicationGatewayBackendAddressPool`.
2. Create a network interface with `New-AzNetworkInterface`.
3. Create a virtual machine configuration with `New-AzVMConfig`.
4. Create the virtual machine with `New-AzVM`.

When you run the following code sample to create the virtual machines, Azure prompts you for credentials.

Enter a user name and a password:

```
$appgw = Get-AzApplicationGateway -ResourceGroupName myResourceGroupAG -Name myAppGateway
$backendPool = Get-AzApplicationGatewayBackendAddressPool -Name myAGBackendPool -ApplicationGateway $appgw
$vnet   = Get-AzVirtualNetwork -ResourceGroupName myResourceGroupAG -Name myVNet
$subnet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vnet -Name myBackendSubnet
$cred = Get-Credential
for ($i=1; $i -le 2; $i++)
{
  $nic = New-AzNetworkInterface `
    -Name myNic$i `
    -ResourceGroupName myResourceGroupAG `
    -Location EastUS `
    -Subnet $subnet `
    -ApplicationGatewayBackendAddressPool $backendpool
  $vm = New-AzVMConfig `
    -VMName myVM$i `
    -VMSize Standard_DS2_v2
  Set-AzVMOperatingSystem `
    -VM $vm `
    -Windows `
    -ComputerName myVM$i `
    -Credential $cred
  Set-AzVMSourceImage `
    -VM $vm `
    -PublisherName MicrosoftWindowsServer `
    -Offer WindowsServer `
    -Skus 2016-Datacenter `
    -Version latest
  Add-AzVMNetworkInterface `
    -VM $vm `
    -Id $nic.Id
  Set-AzVMBootDiagnostic `
    -VM $vm `
    -Disable
  New-AzVM -ResourceGroupName myResourceGroupAG -Location EastUS -VM $vm
  Set-AzVMExtension `
    -ResourceGroupName myResourceGroupAG `
    -ExtensionName IIS `
    -VMName myVM$i `
    -Publisher Microsoft.Compute `
    -ExtensionType CustomScriptExtension `
    -TypeHandlerVersion 1.4 `
    -SettingString '{"commandToExecute":"powershell Add-WindowsFeature Web-Server; powershell Add-Content -
Path \"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"}' `
    -Location EastUS
}
```

Test the application gateway

Although IIS isn't required to create the application gateway, you installed it in this quickstart to verify if Azure successfully created the application gateway.

Use IIS to test the application gateway:

1. Run `Get-AzPublicIPAddress` to get the public IP address of the application gateway.
2. Copy and paste the public IP address into the address bar of your browser. When you refresh the browser, you should see the name of the virtual machine. A valid response verifies that the application gateway was successfully created and it can successfully connect with the backend.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Clean up resources

When you no longer need the resources that you created with the application gateway, delete the resource group. When you delete the resource group, you also delete the application gateway and all its related resources.

To delete the resource group, call the `Remove-AzResourceGroup` cmdlet:

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Manage web traffic with an application gateway using Azure PowerShell

Quickstart: Direct web traffic with Azure Application Gateway - Azure CLI

8/22/2022 • 6 minutes to read • Edit Online

In this quickstart, you use Azure CLI to create an application gateway. Then you test it to make sure it works correctly.

The application gateway directs application web traffic to specific resources in a backend pool. You assign listeners to ports, create rules, and add resources to a backend pool. For the sake of simplicity, this article uses a simple setup with a public front-end IP address, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool.

You can also complete this quickstart using Azure PowerShell or the Azure portal.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This article requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create resource group

In Azure, you allocate related resources to a resource group. Create a resource group by using `az group create`.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

For Azure to communicate between the resources that you create, it needs a virtual network. The application gateway subnet can contain only application gateways. No other resources are allowed. You can either create a new subnet for Application Gateway or use an existing one. In this example, you create two subnets: one for the application gateway, and another for the backend servers. You can configure the Frontend IP of the Application Gateway to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP address.

To create the virtual network and subnet, use `az network vnet create`. Run `az network public-ip create` to create the public IP address.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.21.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.21.0.0/24
az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet   \
  --address-prefix 10.21.1.0/24
az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard
```

Create the backend servers

A backend can have NICs, virtual machine scale sets, public IP addresses, internal IP addresses, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you create two virtual machines to use as backend servers for the application gateway. You also install NGINX on the virtual machines to test the application gateway.

Create two virtual machines

Install the NGINX web server on the virtual machines to verify the application gateway was successfully created. You can use a cloud-init configuration file to install NGINX and run a "Hello World" Node.js app on a Linux virtual machine. For more information about cloud-init, see Cloud-init support for virtual machines in Azure.

In your Azure Cloud Shell, copy and paste the following configuration into a file named *cloud-init.txt*. Enter *editor cloud-init.txt* to create the file.

```
#cloud-config
package_upgrade: true
packages:
  - nginx
  - nodejs
  - npm
write_files:
  - owner: www-data:www-data
  - path: /etc/nginx/sites-available/default
    content: |
      server {
        listen 80;
        location / {
          proxy_pass http://localhost:3000;
          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection keep-alive;
          proxy_set_header Host $host;
          proxy_cache_bypass $http_upgrade;
        }
      }
  - owner: azureuser:azureuser
  - path: /home/azureuser/myapp/index.js
    content: |
      var express = require('express')
      var app = express()
      var os = require('os');
      app.get('/', function (req, res) {
        res.send('Hello World from host ' + os.hostname() + '!')
      })
      app.listen(3000, function () {
        console.log('Hello world app listening on port 3000!')
      })
runcmd:
  - service nginx restart
  - cd "/home/azureuser/myapp"
  - npm init
  - npm install express -y
  - nodejs index.js
```

Create the network interfaces with `az network nic create`. To create the virtual machines, you use `az vm create`.

```
for i in `seq 1 2`; do
  az network nic create \
    --resource-group myResourceGroupAG \
    --name myNic$i \
    --vnet-name myVNet \
    --subnet myBackendSubnet
  az vm create \
    --resource-group myResourceGroupAG \
    --name myVM$i \
    --nics myNic$i \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys \
    --custom-data cloud-init.txt
done
```

Create the application gateway

Create an application gateway using `az network application-gateway create`. When you create an application gateway with the Azure CLI, you specify configuration information, such as capacity, SKU, and HTTP settings.

Azure then adds the private IP addresses of the network interfaces as servers in the backend pool of the application gateway.

```
address1=$(az network nic show --name myNic1 --resource-group myResourceGroupAG | grep
"\"privateIpAddress\":" | grep -oE '[^ ]+$' | tr -d '",')
address2=$(az network nic show --name myNic2 --resource-group myResourceGroupAG | grep
"\"privateIpAddress\":" | grep -oE '[^ ]+$' | tr -d '",')
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --capacity 2 \
  --sku Standard_v2 \
  --public-ip-address myAGPublicIPAddress \
  --vnet-name myVNet \
  --subnet myAGSubnet \
  --servers "$address1" "$address2" \
  --priority 100
```

It can take up to 30 minutes for Azure to create the application gateway. After it's created, you can view the following settings in the **Settings** section of the **Application gateway** page:

- **appGatewayBackendPool**: Located on the **Backend pools** page. It specifies the required backend pool.
- **appGatewayBackendHttpSettings**: Located on the **HTTP settings** page. It specifies that the application gateway uses port 80 and the HTTP protocol for communication.
- **appGatewayHttpListener**: Located on the **Listeners page**. It specifies the default listener associated with **appGatewayBackendPool**.
- **appGatewayFrontendIP**: Located on the **Frontend IP configurations** page. It assigns *myAGPublicIPAddress* to **appGatewayHttpListener**.
- **rule1**: Located on the **Rules** page. It specifies the default routing rule that's associated with **appGatewayHttpListener**.

Test the application gateway

Although Azure doesn't require an NGINX web server to create the application gateway, you installed it in this quickstart to verify whether Azure successfully created the application gateway. To get the public IP address of the new application gateway, use `az network public-ip show`.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

Copy and paste the public IP address into the address bar of your browser.

When you refresh the browser, you should see the name of the second VM. This indicates the application gateway was successfully created and can connect with the backend.

Clean up resources

When you no longer need the resources that you created with the application gateway, use the `az group delete` command to delete the resource group. When you delete the resource group, you also delete the application gateway and all its related resources.

```
az group delete --name myResourceGroupAG
```

Next steps

Manage web traffic with an application gateway using the Azure CLI

Quickstart: Direct web traffic with Azure Application Gateway - Bicep

8/22/2022 • 4 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Azure Application Gateway. Then you test the application gateway to make sure it works correctly.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a simple setup with a public front-end IP address, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool.

The Bicep file used in this quickstart is from Azure Quickstart Templates

```
@description('Admin username for the backend servers')
param adminUsername string

@description('Password for the admin account on the backend servers')
@secure()
param adminPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machine.')
param vmSize string = 'Standard_B2ms'

var virtualMachineName = 'myVM'
var virtualNetworkName = 'myVNet'
var networkInterfaceName = 'net-int'
var ipconfigName = 'ipconfig'
var publicIPAddressName = 'public_ip'
var nsgName = 'vm-nsg'
var applicationGateWayName = 'myAppGateway'
var virtualNetworkPrefix = '10.0.0.0/16'
var subnetPrefix = '10.0.0.0/24'
var backendSubnetPrefix = '10.0.1.0/24'

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = [for i in range(0, 2): {
  name: '${nsgName}${i + 1}'
  location: location
  properties: {
    securityRules: [
      {
        name: 'RDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
```

```
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}]

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = [for i in range(0, 3): {
  name: '${publicIPAddressName}${i}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}]

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        virtualNetworkPrefix
      ]
    }
    subnets: [
      {
        name: 'myAGSubnet'
        properties: {
          addressPrefix: subnetPrefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
      {
        name: 'myBackendSubnet'
        properties: {
          addressPrefix: backendSubnetPrefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource virtualMachine 'Microsoft.Compute/virtualMachines@2021-11-01' = [for i in range(0, 2): {
  name: '${virtualMachineName}${i + 1}'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2016-Datacenter'
        version: 'latest'
```

```
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: '${virtualMachineName}${i + 1}'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: resourceId('Microsoft.Network/networkInterfaces', '${networkInterfaceName}${i + 1}')
        }
      ]
    }
  }
  dependsOn: [
    networkInterface
  ]
}]

resource virtualMachine_IIS 'Microsoft.Compute/virtualMachines/extensions@2021-11-01' = [for i in range(0,
2): {
  name: '${virtualMachineName}${(i + 1)}/IIS'
  location: location
  properties: {
    autoUpgradeMinorVersion: true
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    typeHandlerVersion: '1.4'
    settings: {
      commandToExecute: 'powershell Add-WindowsFeature Web-Server; powershell Add-Content -Path
"C:\\inetpub\\wwwroot\\Default.htm" -Value $($env:computername)'
    }
  }
  dependsOn: [
    virtualMachine
  ]
}]

resource applicationGateWay 'Microsoft.Network/applicationGateways@2021-05-01' = {
  name: applicationGateWayName
  location: location
  properties: {
    sku: {
      name: 'Standard_v2'
      tier: 'Standard_v2'
    }
    gatewayIPConfigurations: [
      {
        name: 'appGatewayIpConfig'
        properties: {
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName, 'myAGSubnet')
          }
        }
```

```
      }
    ]
    frontendIPConfigurations: [
      {
        name: 'appGwPublicFrontendIp'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddressName}0')
          }
        }
      }
    ]
    frontendPorts: [
      {
        name: 'port_80'
        properties: {
          port: 80
        }
      }
    ]
    backendAddressPools: [
      {
        name: 'myBackendPool'
        properties: {}
      }
    ]
    backendHttpSettingsCollection: [
      {
        name: 'myHTTPSetting'
        properties: {
          port: 80
          protocol: 'Http'
          cookieBasedAffinity: 'Disabled'
          pickHostNameFromBackendAddress: false
          requestTimeout: 20
        }
      }
    ]
    httpListeners: [
      {
        name: 'myListener'
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendIPConfigurations',
applicationGateWayName, 'appGwPublicFrontendIp')
          }
          frontendPort: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendPorts', applicationGateWayName,
'port_80')
          }
          protocol: 'Http'
          requireServerNameIndication: false
        }
      }
    ]
    requestRoutingRules: [
      {
        name: 'myRoutingRule'
        properties: {
          ruleType: 'Basic'
          httpListener: {
            id: resourceId('Microsoft.Network/applicationGateways/httpListeners', applicationGateWayName,
'myListener')
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateWayName, 'myBackendPool')
          }
```

```
          backendHttpSettings: {
            id: resourceId('Microsoft.Network/applicationGateways/backendHttpSettingsCollection',
applicationGateWayName, 'myHTTPSetting')
          }
        }
      }
    ]
    enableHttp2: false
    autoscaleConfiguration: {
      minCapacity: 0
      maxCapacity: 10
    }
  }
  dependsOn: [
    virtualNetwork
    publicIPAddress[0]
  ]
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = [for i in range(0, 2): {
  name: '${networkInterfaceName}${i + 1}'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: '${ipconfigName}${i + 1}'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddressName}${i + 1}')
          }
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
'myBackendSubnet')
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
          applicationGatewayBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateWayName, 'myBackendPool')
            }
          ]
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: resourceId('Microsoft.Network/networkSecurityGroups', '${nsgName}${i + 1}')
    }
  }
  dependsOn: [
    publicIPAddress
    applicationGateWay
    nsg
  ]
}]
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/applicationgateways
- Microsoft.Network/publicIPAddresses : one for the application gateway, and two for the virtual machines.
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks

- **Microsoft.Compute/virtualMachines** : two virtual machines
- **Microsoft.Network/networkInterfaces** : two for the virtual machines
- **Microsoft.Compute/virtualMachine/extensions** : to configure IIS and the web pages

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name myResourceGroupAG --location eastus
az deployment group create --resource-group myResourceGroupAG --template-file main.bicep --parameters
adminUsername=<admin-username>
```

> **NOTE**
>
> Replace **<admin-username>** with the admin username for the backend servers. You'll also be prompted to enter **adminPassword**.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group myResourceGroupAG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name myResourceGroupAG
```

Next steps

Manage web traffic with an application gateway using the Azure CLI

Quickstart: Direct web traffic with Azure Application Gateway - ARM template

8/22/2022 • 5 minutes to read • Edit Online

In this quickstart, you use an Azure Resource Manager template (ARM template) to create an Azure Application Gateway. Then you test the application gateway to make sure it works correctly.

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

You can also complete this quickstart using the Azure portal, Azure PowerShell, or Azure CLI.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Deploy to Azure

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the template

For the sake of simplicity, this template creates a simple setup with a public front-end IP, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool.

The template used in this quickstart is from Azure Quickstart Templates

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.5.6.12127",
      "templateHash": "5093932345902341724"
    }
  },
  "parameters": {
    "adminUsername": {
```

```
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Admin username for the backend servers"
      }
    },
    "adminPassword": {
      "type": "secureString",
      "metadata": {
        "description": "Password for the admin account on the backend servers"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    },
    "vmSize": {
      "type": "string",
      "defaultValue": "Standard_B2ms",
      "metadata": {
        "description": "Size of the virtual machine."
      }
    }
  },
  "variables": {
    "virtualMachineName": "myVM",
    "virtualNetworkName": "myVNet",
    "networkInterfaceName": "net-int",
    "ipconfigName": "ipconfig",
    "publicIPAddressName": "public_ip",
    "nsgName": "vm-nsg",
    "applicationGateWayName": "myAppGateway",
    "virtualNetworkPrefix": "10.0.0.0/16",
    "subnetPrefix": "10.0.0.0/24",
    "backendSubnetPrefix": "10.0.1.0/24"
  },
  "resources": [
    {
      "copy": {
        "name": "nsg",
        "count": "[length(range(0, 2))]"
      },
      "type": "Microsoft.Network/networkSecurityGroups",
      "apiVersion": "2021-05-01",
      "name": "[format('{0}{1}', variables('nsgName'), add(range(0, 2)[copyIndex()], 1))]",
      "location": "[parameters('location')]",
      "properties": {
        "securityRules": [
          {
            "name": "RDP",
            "properties": {
              "protocol": "Tcp",
              "sourcePortRange": "*",
              "destinationPortRange": "3389",
              "sourceAddressPrefix": "*",
              "destinationAddressPrefix": "*",
              "access": "Allow",
              "priority": 300,
              "direction": "Inbound"
            }
          }
        ]
      }
    },
    {
      "copy": {
```

```
        "name": "publicIPAddress",
        "count": "[length(range(0, 3))]"
      },
      "type": "Microsoft.Network/publicIPAddresses",
      "apiVersion": "2021-05-01",
      "name": "[format('{0}{1}', variables('publicIPAddressName'), range(0, 3)[copyIndex()])]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard"
      },
      "properties": {
        "publicIPAddressVersion": "IPv4",
        "publicIPAllocationMethod": "Static",
        "idleTimeoutInMinutes": 4
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks",
      "apiVersion": "2021-05-01",
      "name": "[variables('virtualNetworkName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[variables('virtualNetworkPrefix')]"
          ]
        },
        "subnets": [
          {
            "name": "myAGSubnet",
            "properties": {
              "addressPrefix": "[variables('subnetPrefix')]",
              "privateEndpointNetworkPolicies": "Enabled",
              "privateLinkServiceNetworkPolicies": "Enabled"
            }
          },
          {
            "name": "myBackendSubnet",
            "properties": {
              "addressPrefix": "[variables('backendSubnetPrefix')]",
              "privateEndpointNetworkPolicies": "Enabled",
              "privateLinkServiceNetworkPolicies": "Enabled"
            }
          }
        ],
        "enableDdosProtection": false,
        "enableVmProtection": false
      }
    },
    {
      "copy": {
        "name": "virtualMachine",
        "count": "[length(range(0, 2))]"
      },
      "type": "Microsoft.Compute/virtualMachines",
      "apiVersion": "2021-11-01",
      "name": "[format('{0}{1}', variables('virtualMachineName'), add(range(0, 2)[copyIndex()], 1))]",
      "location": "[parameters('location')]",
      "properties": {
        "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "2016-Datacenter",
            "version": "latest"
          },
```

```
          "osDisk": {
            "osType": "Windows",
            "createOption": "FromImage",
            "caching": "ReadWrite",
            "managedDisk": {
              "storageAccountType": "StandardSSD_LRS"
            },
            "diskSizeGB": 127
          }
        },
        "osProfile": {
          "computerName": "[format('{0}{1}', variables('virtualMachineName'), add(range(0, 2)[copyIndex()],
1))]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPassword')]",
          "windowsConfiguration": {
            "provisionVMAgent": true,
            "enableAutomaticUpdates": true
          },
          "allowExtensionOperations": true
        },
        "networkProfile": {
          "networkInterfaces": [
            {
              "id": "[resourceId('Microsoft.Network/networkInterfaces', format('{0}{1}',
variables('networkInterfaceName'), add(range(0, 2)[copyIndex()], 1)))]"
            }
          ]
        }
      },
      "dependsOn": [
        "networkInterface"
      ]
    },
    {
      "copy": {
        "name": "virtualMachine_IIS",
        "count": "[length(range(0, 2))]"
      },
      "type": "Microsoft.Compute/virtualMachines/extensions",
      "apiVersion": "2021-11-01",
      "name": "[format('{0}{1}/IIS', variables('virtualMachineName'), add(range(0, 2)[copyIndex()], 1))]",
      "location": "[parameters('location')]",
      "properties": {
        "autoUpgradeMinorVersion": true,
        "publisher": "Microsoft.Compute",
        "type": "CustomScriptExtension",
        "typeHandlerVersion": "1.4",
        "settings": {
          "commandToExecute": "powershell Add-WindowsFeature Web-Server; powershell Add-Content -Path
\"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"
        }
      },
      "dependsOn": [
        "virtualMachine"
      ]
    },
    {
      "type": "Microsoft.Network/applicationGateways",
      "apiVersion": "2021-05-01",
      "name": "[variables('applicationGateWayName')]",
      "location": "[parameters('location')]",
      "properties": {
        "sku": {
          "name": "Standard_v2",
          "tier": "Standard_v2"
        },
        "gatewayIPConfigurations": [
          {
```

```
            "name": "appGatewayIpConfig",
            "properties": {
              "subnet": {
                "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets',
variables('virtualNetworkName'), 'myAGSubnet')]"
              }
            }
          }
        ],
        "frontendIPConfigurations": [
          {
            "name": "appGwPublicFrontendIp",
            "properties": {
              "privateIPAllocationMethod": "Dynamic",
              "publicIPAddress": {
                "id": "[resourceId('Microsoft.Network/publicIPAddresses', format('{0}0',
variables('publicIPAddressName')))]"
              }
            }
          }
        ],
        "frontendPorts": [
          {
            "name": "port_80",
            "properties": {
              "port": 80
            }
          }
        ],
        "backendAddressPools": [
          {
            "name": "myBackendPool",
            "properties": {}
          }
        ],
        "backendHttpSettingsCollection": [
          {
            "name": "myHTTPSetting",
            "properties": {
              "port": 80,
              "protocol": "Http",
              "cookieBasedAffinity": "Disabled",
              "pickHostNameFromBackendAddress": false,
              "requestTimeout": 20
            }
          }
        ],
        "httpListeners": [
          {
            "name": "myListener",
            "properties": {
              "frontendIPConfiguration": {
                "id": "[resourceId('Microsoft.Network/applicationGateways/frontendIPConfigurations',
variables('applicationGateWayName'), 'appGwPublicFrontendIp')]"
              },
              "frontendPort": {
                "id": "[resourceId('Microsoft.Network/applicationGateways/frontendPorts',
variables('applicationGateWayName'), 'port_80')]"
              },
              "protocol": "Http",
              "requireServerNameIndication": false
            }
          }
        ],
        "requestRoutingRules": [
          {
            "name": "myRoutingRule",
            "properties": {
              "ruleType": "Basic",
```

```
              "httpListener": {
                "id": "[resourceId('Microsoft.Network/applicationGateways/httpListeners',
variables('applicationGateWayName'), 'myListener')]"
              },
              "backendAddressPool": {
                "id": "[resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
variables('applicationGateWayName'), 'myBackendPool')]"
              },
              "backendHttpSettings": {
                "id": "[resourceId('Microsoft.Network/applicationGateways/backendHttpSettingsCollection',
variables('applicationGateWayName'), 'myHTTPSetting')]"
              }
            }
          }
        ],
        "enableHttp2": false,
        "autoscaleConfiguration": {
          "minCapacity": 0,
          "maxCapacity": 10
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/publicIPAddresses', format('{0}{1}',
variables('publicIPAddressName'), range(0, 3)[0]))]",
        "[resourceId('Microsoft.Network/virtualNetworks', variables('virtualNetworkName'))]"
      ]
    },
    {
      "copy": {
        "name": "networkInterface",
        "count": "[length(range(0, 2))]"
      },
      "type": "Microsoft.Network/networkInterfaces",
      "apiVersion": "2021-05-01",
      "name": "[format('{0}{1}', variables('networkInterfaceName'), add(range(0, 2)[copyIndex()], 1))]",
      "location": "[parameters('location')]",
      "properties": {
        "ipConfigurations": [
          {
            "name": "[format('{0}{1}', variables('ipconfigName'), add(range(0, 2)[copyIndex()], 1))]",
            "properties": {
              "privateIPAllocationMethod": "Dynamic",
              "publicIPAddress": {
                "id": "[resourceId('Microsoft.Network/publicIPAddresses', format('{0}{1}',
variables('publicIPAddressName'), add(range(0, 2)[copyIndex()], 1)))]"
              },
              "subnet": {
                "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets',
variables('virtualNetworkName'), 'myBackendSubnet')]"
              },
              "primary": true,
              "privateIPAddressVersion": "IPv4",
              "applicationGatewayBackendAddressPools": [
                {
                  "id": "[resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
variables('applicationGateWayName'), 'myBackendPool')]"
                }
              ]
            }
          }
        ],
        "enableAcceleratedNetworking": false,
        "enableIPForwarding": false,
        "networkSecurityGroup": {
          "id": "[resourceId('Microsoft.Network/networkSecurityGroups', format('{0}{1}',
variables('nsgName'), add(range(0, 2)[copyIndex()], 1)))]"
        }
      },
      "dependsOn": [
```

```
        "[resourceId('Microsoft.Network/applicationGateways', variables('applicationGateWayName'))]",
        "nsg",
        "publicIPAddress"
      ]
    }
  ]
}
```

Multiple Azure resources are defined in the template:

- **Microsoft.Network/applicationgateways**
- **Microsoft.Network/publicIPAddresses** : one for the application gateway, and two for the virtual machines.
- **Microsoft.Network/networkSecurityGroups**
- **Microsoft.Network/virtualNetworks**
- **Microsoft.Compute/virtualMachines** : two virtual machines
- **Microsoft.Network/networkInterfaces** : two for the virtual machines
- **Microsoft.Compute/virtualMachine/extensions** : to configure IIS and the web pages

Deploy the template

Deploy the ARM template to Azure:

1. Select **Deploy to Azure** to sign in to Azure and open the template. The template creates an application gateway, the network infrastructure, and two virtual machines in the backend pool running IIS.

   Deploy to Azure

2. Select or create your resource group, type the virtual machine administrator user name and password.

3. Select **Review + Create** and then select **Create**.

   The deployment can take 20 minutes or longer to complete.

Validate the deployment

Although IIS isn't required to create the application gateway, it's installed to verify if Azure successfully created the application gateway. Use IIS to test the application gateway:

1. Find the public IP address for the application gateway on its **Overview** page.

   Or, you can select **All resources**, enter *myAGPublicIPAddress* in the search box, and then select it in the search results. Azure displays the public IP address on the **Overview** page.

2. Copy the public IP address, and then paste it into the address bar of your browser to browse that IP address.

3. Check the response. A valid response verifies that the application gateway was successfully created and can successfully connect with the backend.

Refresh the browser multiple times and you should see connections to both myVM1 and myVM2.

Clean up resources

When you no longer need the resources that you created with the application gateway, delete the resource group. This removes the application gateway and all the related resources.

To delete the resource group, call the `Remove-AzResourceGroup` cmdlet:

```
Remove-AzResourceGroup -Name <your resource group name>
```

Next steps

Manage web traffic with an application gateway using the Azure CLI

Tutorial: Configure an Application Gateway with TLS termination using the Azure portal

8/22/2022 • 9 minutes to read • Edit Online

You can use the Azure portal to configure an application gateway with a certificate for TLS termination that uses virtual machines for backend servers.

In this tutorial, you learn how to:

- Create a self-signed certificate
- Create an application gateway with the certificate
- Create the virtual machines used as backend servers
- Test the application gateway

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisites

- An Azure subscription

Create a self-signed certificate

In this section, you use New-SelfSignedCertificate to create a self-signed certificate. You upload the certificate to the Azure portal when you create the listener for the application gateway.

On your local computer, open a Windows PowerShell window as an administrator. Run the following command to create the certificate:

```
New-SelfSignedCertificate `
  -certstorelocation cert:\localmachine\my `
  -dnsname www.contoso.com
```

You should see something like this response:

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject
----------                                -------
E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630  CN=www.contoso.com
```

Use Export-PfxCertificate with the Thumbprint that was returned to export a pfx file from the certificate. Make sure your password is 4 - 12 characters long:

```
$pwd = ConvertTo-SecureString -String <your password> -Force -AsPlainText
Export-PfxCertificate `
  -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `
  -FilePath c:\appgwcert.pfx `
  -Password $pwd
```

Sign in to Azure

Sign in to the Azure portal.

Create an application gateway

1. From the Azure portal menu, select **+ Create a resource** > **Networking** > **Application Gateway**, or search for *Application Gateway* in the portal search box.

2. Select **Create**.

Basics tab

1. On the **Basics** tab, enter or select these values:

   - **Resource group**: Select **myResourceGroupAG** for the resource group. If it doesn't exist, select **Create new** to create it.

   - **Application gateway name**: Enter *myAppGateway* for the name of the application gateway.

2. For Azure to communicate between the resources that you create, it needs a virtual network. You can either create a new virtual network or use an existing one. In this example, you'll create a new virtual network at the same time that you create the application gateway. Application Gateway instances are created in separate subnets. You create two subnets in this example: one for the application gateway, and another for the backend servers.

   Under **Configure virtual network**, create a new virtual network by selecting **Create new**. In the **Create virtual network** window that opens, enter the following values to create the virtual network and two subnets:

   - **Name**: Enter *myVNet* for the name of the virtual network.
   - **Subnet name** (Application Gateway subnet): The **Subnets** grid will show a subnet named *Default*. Change the name of this subnet to *myAGSubnet*.
     The application gateway subnet can contain only application gateways. No other resources are allowed.
   - **Subnet name** (backend server subnet): In the second row of the **Subnets** grid, enter

myBackendSubnet in the **Subnet name** column.

- **Address range** (backend server subnet): In the second row of the **Subnets** Grid, enter an address range that doesn't overlap with the address range of *myAGSubnet*. For example, if the address range of *myAGSubnet* is 10.0.0.0/24, enter *10.0.1.0/24* for the address range of *myBackendSubnet*.

Select **OK** to close the **Create virtual network** window and save the virtual network settings.

3. On the **Basics** tab, accept the default values for the other settings and then select **Next: Frontends**.

Frontends tab

1. On the **Frontends** tab, verify **Frontend IP address type** is set to **Public**.
   You can configure the Frontend IP to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP.

   > **NOTE**
   > For the Application Gateway v2 SKU, you can only choose **Public** frontend IP configuration. Private frontend IP configuration is currently not enabled for this v2 SKU.

2. Choose **Add new** for the **Public IP address** and enter *myAGPublicIPAddress* for the public IP address name, and then select **OK**.

3. Select **Next: Backends**.

Backends tab

The backend pool is used to route requests to the backend servers that serve the request. Backend pools can be composed of NICs, virtual machine scale sets, public IPs, internal IPs, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you'll create an empty backend pool with your application gateway and then add backend targets to the backend pool.

1. On the **Backends** tab, select **Add a backend pool**.

2. In the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:

   - **Name**: Enter *myBackendPool* for the name of the backend pool.
   - **Add backend pool without targets**: Select **Yes** to create a backend pool with no targets. You'll add backend targets after creating the application gateway.

3. In the **Add a backend pool** window, select **Add** to save the backend pool configuration and return to the **Backends** tab.

4. On the **Backends** tab, select **Next: Configuration**.

Configuration tab

On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule.

1. Select **Add a routing rule** in the **Routing rules** column.

2. In the **Add a routing rule** window that opens, enter *myRoutingRule* for the **Rule name**.

3. A routing rule requires a listener. On the **Listener** tab within the **Add a routing rule** window, enter the following values for the listener:

   - **Listener name**: Enter *myListener* for the name of the listener.
   - **Frontend IP**: Select **Public** to choose the public IP you created for the frontend.
   - **Protocol**: Select **HTTPS**.
   - **Port**: Verify 443 is entered for the port.

   Under **HTTPS Settings**:

   - **Choose a certificate** - Select **Upload a certificate**.

   - **PFX certificate file** - Browse to and select the c:\appgwcert.pfx file that you create earlier.

   - **Certificate name** - Type *mycert1* for the name of the certificate.

   - **Password** - Type the password you used to create the certificate.

     Accept the default values for the other settings on the **Listener** tab, then select the **Backend targets** tab to configure the rest of the routing rule.

4. On the **Backend targets** tab, select **myBackendPool** for the **Backend target**.

5. For the **HTTP setting**, select **Add new** to create a new HTTP setting. The HTTP setting will determine the behavior of the routing rule. In the **Add a HTTP setting** window that opens, enter *myHTTPSetting* for the **HTTP setting name**. Accept the default values for the other settings in the **Add a HTTP setting** window, then select **Add** to return to the **Add a routing rule** window.

6. On the **Add a routing rule** window, select **Add** to save the routing rule and return to the **Configuration** tab.

7. Select **Next: Tags** and then **Next: Review + create**.

Review + create tab

Review the settings on the **Review + create** tab, and then select **Create** to create the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

Add backend targets

In this example, you'll use virtual machines as the target backend. You can either use existing virtual machines or create new ones. You'll create two virtual machines that Azure uses as backend servers for the application gateway.

To do this, you'll:

1. Create two new VMs, *myVM* and *myVM2*, to be used as backend servers.
2. Install IIS on the virtual machines to verify that the application gateway was created successfully.
3. Add the backend servers to the backend pool.

Create a virtual machine

1. From the Azure portal menu, select **+ Create a resource** > **Compute** > **Windows Server 2016 Datacenter**, or search for *Windows Server* in the portal search box and select **Windows Server 2016 Datacenter**.

2. Select **Create**.

   Application Gateway can route traffic to any type of virtual machine used in its backend pool. In this example, you use a Windows Server 2016 Datacenter.

3. Enter these values in the **Basics** tab for the following virtual machine settings:

   - **Resource group**: Select **myResourceGroupAG** for the resource group name.
   - **Virtual machine name**: Enter *myVM* for the name of the virtual machine.
   - **Username**: Enter a name for the administrator user name.
   - **Password**: Enter a password for the administrator account.
4. Accept the other defaults and then select **Next: Disks**.

5. Accept the **Disks** tab defaults and then select **Next: Networking**.

6. On the **Networking** tab, verify that **myVNet** is selected for the **Virtual network** and the **Subnet** is set to **myBackendSubnet**. Accept the other defaults and then select **Next: Management**.

   Application Gateway can communicate with instances outside of the virtual network that it is in, but you need to ensure there's IP connectivity.

7. On the **Management** tab, set **Boot diagnostics** to **Disable**. Accept the other defaults and then select **Review + create**.

8. On the **Review + create** tab, review the settings, correct any validation errors, and then select **Create**.

9. Wait for the deployment to complete before continuing.

Install IIS for testing

In this example, you install IIS on the virtual machines only to verify Azure created the application gateway successfully.

1. Open Azure PowerShell. To do so, select **Cloud Shell** from the top navigation bar of the Azure portal and then select **PowerShell** from the drop-down list.

2. Change the location setting for your environment, and then run the following command to install IIS on the virtual machine:

```
        Set-AzVMExtension `
          -ResourceGroupName myResourceGroupAG `
          -ExtensionName IIS `
          -VMName myVM `
          -Publisher Microsoft.Compute `
          -ExtensionType CustomScriptExtension `
          -TypeHandlerVersion 1.4 `
          -SettingString '{"commandToExecute":"powershell Add-WindowsFeature Web-Server; powershell
Add-Content -Path \"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"}' `
          -Location <location>
```

3. Create a second virtual machine and install IIS by using the steps that you previously completed. Use *myVM2* for the virtual machine name and for the **VMName** setting of the **Set-AzVMExtension** cmdlet.

Add backend servers to backend pool

1. Select **All resources**, and then select **myAppGateway**.
2. Select **Backend pools** from the left menu.
3. Select **myBackendPool**.
4. Under **Target type**, select **Virtual machine** from the drop-down list.
5. Under **Target**, select the network interface under **myVM** from the drop-down list.
6. Repeat to add the network interface for **myVM2**.

7. Select **Save**.
8. Wait for the deployment to complete before proceeding to the next step.

Test the application gateway

1. Select **All resources**, and then select **myAGPublicIPAddress**.

2. In the address bar of your browser, type *https://<your application gateway ip address>*.

   To accept the security warning if you used a self-signed certificate, select **Details** (or **Advanced** on Chrome) and then go on to the webpage:

   Your secured IIS website is then displayed as in the following example:

Clean up resources

When no longer needed, delete the resource group and all related resources. To do so, select the resource group and select **Delete resource group**.

Next steps

In this tutorial, you:

- Created a self-signed certificate
- Created an application gateway with the certificate

To learn more about Application Gateway TLS support, see end to end TLS with Application Gateway and Application Gateway TLS policy.

To learn how to create and configure an Application Gateway to host multiple web sites using the Azure portal, advance to the next tutorial.

Host multiple sites

Tutorial: Create and configure an application gateway to host multiple web sites using the Azure portal

8/22/2022 • 10 minutes to read • Edit Online

You can use the Azure portal to configure the hosting of multiple web sites when you create an application gateway. In this tutorial, you define backend address pools using virtual machines. You then configure listeners and rules based on two domains to make sure web traffic arrives at the appropriate servers in the pools. This tutorial uses examples of `www.contoso.com` and `www.fabrikam.com`.

In this tutorial, you learn how to:

- Create an application gateway
- Create virtual machines for backend servers
- Create backend pools with the backend servers
- Create listeners
- Create routing rules
- Edit hosts file for name resolution

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisites

- An Azure subscription

Sign in to Azure

Sign in to the Azure portal.

Create an application gateway

1. From the Azure portal menu, select **+ Create a resource** > **Networking** > **Application Gateway**, or search for *Application Gateway* in the portal search box.

2. Select **Create**.

Basics tab

1. On the **Basics** tab, enter these values:

   - **Resource group**: Select **myResourceGroupAG** for the resource group. If it doesn't exist, select **Create new** to create it.

   - **Application gateway name**: Enter *myAppGateway* for the name of the application gateway.

2. For Azure to communicate between the resources that you create, it needs a virtual network. You can either create a new virtual network or use an existing one. In this example, you'll create a new virtual network at the same time that you create the application gateway. Application gateway instances are

created in separate subnets. You create two subnets in this example: one for the application gateway, and another for the backend servers.

Under **Configure virtual network**, select **Create new** to create a new virtual network. In the **Create virtual network** window that opens, enter the following values to create the virtual network and two subnets:

- **Name**: Enter *myVNet* for the name of the virtual network.
- **Subnet name** (application gateway subnet): The **Subnets** grid will show a subnet named *Default*. Change the name of this subnet to *myAGSubnet*.
  The application gateway subnet can contain only application gateways. No other resources are allowed.
- **Subnet name** (backend server subnet): In the second row of the **Subnets** grid, enter *myBackendSubnet* in the **Subnet name** column.
- **Address range** (backend server subnet): In the second row of the **Subnets** Grid, enter an address range that doesn't overlap with the address range of *myAGSubnet*. For example, if the address range of *myAGSubnet* is 10.0.0.0/24, enter *10.0.1.0/24* for the address range of *myBackendSubnet*.

Select **OK** to close the **Create virtual network** window and save the virtual network settings.

3. On the **Basics** tab, accept the default values for the other settings and then select **Next: Frontends**.

Frontends tab

1. On the **Frontends** tab, verify **Frontend IP address type** is set to **Public**.
   You can configure the Frontend IP to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP.

> **NOTE**
>
> For the application gateway v2 SKU, you can only choose **Public** frontend IP configuration. Private frontend IP configuration is currently not enabled for this v2 SKU.

2. Select **Add new** for the **Public IP address** and enter *myAGPublicIPAddress* for the public IP address name, and then select **OK**.

3. Select **Next: Backends**.

Backends tab

The backend pool is used to route requests to the backend servers that serve the request. Backend pools can be NICs, virtual machine scale sets, public IPs, internal IPs, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you'll create an empty backend pool with your application gateway and then add backend targets to the backend pool.

1. On the **Backends** tab, select **Add a backend pool**.
2. In the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:
   - **Name**: Enter *contosoPool* for the name of the backend pool.
   - **Add backend pool without targets**: Select **Yes** to create a backend pool with no targets. You'll add backend targets after creating the application gateway.
3. In the **Add a backend pool** window, select **Add** to save the backend pool configuration and return to the **Backends** tab.
4. Now add another backend pool called *fabrikamPool* the same way that you added the previous pool.
5. Select **Add**.

6. On the **Backends** tab, select **Next: Configuration**.

Configuration tab

On the **Configuration** tab, you'll connect the frontend and backend pools you created using a routing rule.

1. Select **Add a routing rule** in the **Routing rules** column.
2. In the **Add a routing rule** window that opens, enter *contosoRule* for the **Rule name**.
3. A routing rule requires a listener. On the **Listener** tab within the **Add a routing rule** window, enter the following values for the listener:
   - **Rule name**: *contosoRule*.
   - **Listener name**: *contosoListener*.
   - **Frontend IP**: Select **Public** to choose the public IP you created for the frontend.

   Under **Additional settings**:

   - **Listener type**: Multiple sites
   - **Host name**: `www.contoso.com`

   Accept the default values for the other settings on the **Listener** tab, then select the **Backend targets** tab to configure the rest of the routing rule.

4. On the **Backend targets** tab, select **contosoPool** for the **Backend target**.

5. For the **HTTP setting**, select **Add new** to create a new HTTP setting. The HTTP setting will determine the behavior of the routing rule. In the **Add an HTTP setting** window that opens, enter *contosoHTTPSetting* for the **HTTP setting name**. Accept the default values for the other settings in the **Add an HTTP setting** window, then select **Add** to return to the **Add a routing rule** window.

6. On the **Add a routing rule** window, select **Add** to save the routing rule and return to the **Configuration** tab.

7. Select **Add a routing rule** and add a similar rule, listener, backend target, and HTTP setting for Fabrikam.

8. Select **Next: Tags** and then **Next: Review + create**.

Review + create tab

Review the settings on the **Review + create** tab, and then select **Create** to create the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway.

Wait until the deployment finishes successfully before moving on to the next section.

Add backend targets

In this example, you'll use virtual machines as the target backend. You can either use existing virtual machines or create new ones. You'll create two virtual machines that Azure uses as backend servers for the application gateway.

To add backend targets, you'll:

1. Create two new VMs, *contosoVM* and *fabrikamVM*, to be used as backend servers.
2. Install IIS on the virtual machines to verify that the application gateway was created successfully.
3. Add the backend servers to the backend pools.

Create a virtual machine

1. On the Azure portal, select **Create a resource**. The **New** window appears.

2. Select **Windows Server 2016 Datacenter** in the **Popular** list. The **Create a virtual machine** page appears.
   Application Gateway can route traffic to any type of virtual machine used in its backend pool. In this example, you use a Windows Server 2016 Datacenter.

3. Enter these values in the **Basics** tab for the following virtual machine settings:

- **Subscription**: Select your subscription.
- **Resource group**: Select **myResourceGroupAG** for the resource group name.
- **Virtual machine name**: Enter *contosoVM* for the name of the virtual machine.
- **Region**: Select the same region that you used before.
- **Username**: Enter a name for the administrator user name.
- **Password**: Enter a password for the administrator.

4. Accept the other defaults and then select **Next: Disks**.

5. Accept the **Disks** tab defaults and then select **Next: Networking**.

6. On the **Networking** tab, verify that **myVNet** is selected for the **Virtual network** and the **Subnet** is set to **myBackendSubnet**. Accept the other defaults and then select **Next: Management**.
   Application Gateway can communicate with instances outside of the virtual network that it is in, but you need to ensure there's IP connectivity.

7. On the **Management** tab, set **Boot diagnostics** to **Disable**. Accept the other defaults and then select **Review + create**.

8. On the **Review + create** tab, review the settings, correct any validation errors, and then select **Create**.

9. Wait for the virtual machine creation to complete before continuing.

Install IIS for testing

In this example, you install IIS on the virtual machines only to verify Azure created the application gateway successfully.

1. Open Azure PowerShell. To do so, select **Cloud Shell** from the top navigation bar of the Azure portal and then select **PowerShell** from the drop-down list.

2. Run the following command to install IIS on the virtual machine, substituting your resource group region for <location>:

```
Set-AzVMExtension `
  -ResourceGroupName myResourceGroupAG `
  -ExtensionName IIS `
  -VMName contosoVM `
  -Publisher Microsoft.Compute `
  -ExtensionType CustomScriptExtension `
  -TypeHandlerVersion 1.4 `
  -SettingString '{"commandToExecute":"powershell Add-WindowsFeature Web-Server; powershell Add-
Content -Path \"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"}' `
  -Location <location>
```

3. Create a second virtual machine and install IIS using the steps that you previously completed. Use *fabrikamVM* for the virtual machine name and for the **VMName** setting of the **Set-AzVMExtension** cmdlet.

Add backend servers to backend pools

1. Select **All resources**, and then select **myAppGateway**.
2. Select **Backend pools** from the left menu.
3. Select **contosoPool**.
4. Under **Target type**, select **Virtual machine** from the drop-down list.
5. Under **Target**, select the **contosoVM** virtual machine's network interface from the drop-down list.

6. Select **Save**.
7. Repeat to add the *fabrikamVM* and interface to the *fabrikamPool*.

Wait for the deployment to complete before proceeding to the next step.

Edit your hosts file for name resolution

After the application gateway is created with its public IP address, you can get the IP address, and use it to edit your hosts file to resolve `www.contoso.com` and `www.fabrikam.com`. In a production environment, you could create a `CNAME` in DNS for name resolution.

1. Select **All resources**, and then select **myAGPublicIPAddress**.

2. Copy the IP address and use it as the value for new entries your `hosts` file.

3. On your local machine, open an administrative command prompt, and navigate to `c:\Windows\System32\drivers\etc`.

4. Open the `hosts` file, and add the following entries, where `x.x.x.x` is the application gateway's public IP address:

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
# 127.0.0.1       localhost
# ::1             localhost
x.x.x.x www.contoso.com
x.x.x.x www.fabrikam.com
```

5. Save the file.

6. Run the following commands to load and display the changes to your hosts file:

```
ipconfig/registerdns
ipconfig/displaydns
```

Test the application gateway

1. Type a domain name into the address bar of your browser. For example, `http://www.contoso.com`.

2. Change the address to the other domain and you should see something like the following example:

Clean up resources

When you no longer need the resources that you created with the application gateway, delete the resource group. When you delete the resource group, you also delete the application gateway and all its related resources.

To remove the resource group:

1. On the left menu of the Azure portal, select **Resource groups**.
2. On the **Resource groups** page, search for **myResourceGroupAG** in the list, then select it.
3. On the **myResourceGroupAG** page, select **Delete resource group**.
4. Enter *myResourceGroupAG* for **TYPE THE RESOURCE GROUP NAME** and then select **Delete**.

To restore the hosts file:

1. Delete the `www.contoso.com` and `www.fabrikam.com` lines from the `hosts` file.
2. Run `ipconfig/registerdns` and `ipconfig/flushdns` from the command prompt.

Next steps

In this tutorial, you:

- Created an application gateway with listeners and rules based on two domains
- Tested the application gateway after editing the host files of backend servers

To learn more about hosting multiple sites, see application gateway multiple site hosting.

To learn how to create and configure an application gateway with path-based routing rules using the Azure portal, advance to the next tutorial.

Route by URL

Tutorial: Create an application gateway with path-based routing rules using the Azure portal

8/22/2022 • 6 minutes to read • Edit Online

You can use the Azure portal to configure URL path-based routing rules when you create an application gateway. In this tutorial, you create backend pools using virtual machines. You then create routing rules that make sure web traffic arrives at the appropriate servers in the pools.

In this article, you learn how to:

- Create an application gateway
- Create virtual machines for backend servers
- Create backend pools with the backend servers
- Create a backend listener
- Create a path-based routing rule

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- An Azure subscription

Create virtual machines

In this example, you create three virtual machines to be used as backend servers for the application gateway. You also install IIS on the virtual machines to verify that the application gateway works as expected.

1. Sign in to the Azure portal at https://portal.azure.com.
2. On the Azure portal, select **Create a resource**.
3. Select **Windows Server 2016 Datacenter** in the Popular list.
4. Enter these values for the virtual machine:

- **Subscription** - Select your subscription.
- **Resource group**, select **Create new**, and then type *myResourceGroupAG*.
- **Virtual machine name**: *myVM1*
- **Region**: *(US) East US*
- **Username**: Type a user name
- **Password**: Type a password

5. Select **Next:Disks**.
6. Select **Next:Networking**
7. For **Virtual network**, select **Create new** and then type these values for the virtual network:
   - *myVNet* - for the name of the virtual network.
   - *10.0.0.0/16* - for the virtual network address space.
   - *myBackendSubnet* for the first subnet name
   - *10.0.1.0/24* - for the subnet address space.
   - *myAGSubnet* - for the second subnet name.
   - *10.0.0.0/24* - for the subnet address space.
8. Select **OK**.
9. Ensure that under **Subnet**, **myBackendSubnet** is selected for the subnet, and then select **Next: Management**.
10. Select **Disable** to disable boot diagnostics.
11. Select **Review + Create**, review the settings on the summary page, and then select **Create**.
12. Create two more virtual machines, *myVM2* and *myVM3* and place them in the *MyVNet* virtual network and the *myBackendSubnet* subnet.

Install IIS

1. Open the interactive shell and make sure that it's set to **PowerShell**.

2. Run the following command to install IIS on the virtual machine:

```
     $publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-
powershell-samples/master/application-gateway/iis/appgatewayurl.ps1");  "commandToExecute" =
"powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

    Set-AzVMExtension `
     -ResourceGroupName myResourceGroupAG `
     -Location eastus `
     -ExtensionName IIS `
     -VMName myVM1 `
     -Publisher Microsoft.Compute `
     -ExtensionType CustomScriptExtension `
     -TypeHandlerVersion 1.4 `
     -Settings $publicSettings
```

3. Install IIS on the other virtual machines using the steps that you just finished. Use *myVM2* and *myVM3* for VMName values in Set-AzVMExtension.

Create an application gateway

1. Select **Create a resource** on the left menu of the Azure portal.

2. Select **Networking** and then select **Application Gateway** in the **Featured** list.

Basics tab

1. On the **Basics** tab, enter these values for the following application gateway settings:

   - **Subscription**: Select your subscription.
   - **Resource group**: Select **myResourceGroupAG** for the resource group.
   - **Application gateway name**: Type *myAppGateway* for the name of the application gateway.
   - **Region** - Select **East US**.
2. Under **Configure virtual network**, select **myVNet** for the name of the virtual network.

3. Select **myAGSubnet** for the subnet.

4. Accept the default values for the other settings and then select **Next: Frontends**.

Home > Create a resource >

Create application gateway ...

1 Basics | 2 Frontends | 3 Backends | 4 Configuration | 5 Tags | 6 Review + create

An application gateway is a web traffic load balancer that enables you to manage traffic to your web application. Learn more about application gateway

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Azure Subscription
Resource group *	(New) MyResourceGroupAG Create new

Instance details

Application gateway name *	myAppGateway
Region *	East US
Tier	Standard V2
Enable autoscaling	Yes / No
Minimum instance count *	0
Maximum instance count	10
Availability zone	None
HTTP2	Disabled / Enabled

Configure virtual network

Virtual network *	myVNet Create new
Subnet *	myAGSubnet (10.0.0.0/24) Manage subnet configuration

Previous | Next : Frontends >

Frontends tab

1. On the **Frontends** tab, verify **Frontend IP address type** is set to **Public**.

> **NOTE**
>
> For the Application Gateway v2 SKU, you can only choose **Public** frontend IP configuration. Private frontend IP configuration is currently not enabled for this v2 SKU.

2. Select **Add new** for the **Public IP address** and enter *myAGPublicIPAddress* for the public IP address name, and then select **OK**.

3. Select **Next: Backends**.

Backends tab

The backend pool is used to route requests to the backend servers that serve the request. Backend pools can be composed of NICs, virtual machine scale sets, public IPs, internal IPs, fully qualified domain names (FQDN), and

multi-tenant back-ends like Azure App Service.

1. On the **Backends** tab, select **Add a backend pool**.
2. In the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:
   - **Name**: Enter *myBackendPool* for the name of the backend pool.
3. Under **Target type**, select **Virtual machine** from the drop-down list.
4. Under **Target** select the network interface for **myVM1**.
5. Select **Add**.
6. Repeat to add an *Images* backend pool with *myVM2* as the target, and a *Video* backend pool with *myVM3* as the target.
7. Select **Add** to save the backend pool configuration and return to the **Backends** tab.
8. On the **Backends** tab, select **Next: Configuration**.

Configuration tab

On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule.

1. Select **Add a routing rule** in the **Routing rules** column.
2. In the **Add a routing rule** window that opens, enter *myRoutingRule* for the **Rule name**.
3. A routing rule requires a listener. On the **Listener** tab within the **Add a routing rule** window, type the following values for the listener:
   - **Listener name**: Enter *myListener* for the name of the listener.
   - **Frontend IP**: Select **Public** to choose the public IP you created for the frontend.
   - **Port**: Type *8080*

     Accept the default values for the other settings on the **Listener** tab, then select the **Backend targets** tab to configure the rest of the routing rule.
4. On the **Backend targets** tab, select **myBackendPool** for the **Backend target**.
5. For the **HTTP setting**, select **Add new** to create a new HTTP setting. The HTTP setting will determine the behavior of the routing rule.
6. In the **Add an HTTP setting** window that opens, enter *myHTTPSetting* for the **HTTP setting name**. Accept the default values for the other settings in the **Add an HTTP setting** window, then select **Add** to return to the **Add a routing rule** window.
7. Under **Path-based routing**, select **Add multiple targets to create a path-based rule**.
8. For **Path**, type */images/**.
9. For **Target name**, type *Images*.
10. For **HTTP setting**, select **myHTTPSetting**
11. For **Backend target**, select **Images**.
12. Select **Add** to save the path rule and return to the **Add a routing rule** tab.
13. Repeat to add another rule for Video.

14. Select **Add** to add the routing rule and return to the **Configuration** tab.

15. Select **Next: Tags** and then **Next: Review + create**.

> **NOTE**
>
> You do not need to add a custom /* path rule to handle default cases. This is automatically handled by the default backend pool.

> **NOTE**
>
> Wildcard delimiter * is only honored at the end of the rule. For more information and supported path based rules examples, see URL Path Based Routing overview.

Review + create tab

Review the settings on the **Review + create** tab, and then select **Create** to create the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

Test the application gateway

1. Select **All resources**, and then select **myAppGateway**.

2. Copy the public IP address, and then paste it into the address bar of your browser. Such as, http://52.188.72.175:8080.

The listener on port 8080 routes this request to the default backend pool.

3. Change the URL to *http://<ip-address>:8080/images/test.htm*, replacing <ip-address> with the public IP address of **myAppGateway**, and you should see something like the following example:

The listener on port 8080 routes this request to the *Images* backend pool.

4. Change the URL to *http://<ip-address>:8080/video/test.htm*, replacing <ip-address> with the public IP address of **myAppGateway**, and you should see something like the following example:

The listener on port 8080 routes this request to the *Video* backend pool.

Clean up resources

When no longer needed, delete the resource group and all related resources. To do so, select the resource group and select **Delete resource group**.

Next steps

In this tutorial, you created an application gateway with a path-based routing rule.

To learn more about path-based routing in Application Gateways, see URL path-based routing overview

To learn how to create and configure an Application Gateway to redirect web traffic using the Azure CLI, advance to the next tutorial.

Redirect web traffic

Tutorial: Create an application gateway with URL path-based redirection using the Azure CLI

8/22/2022 • 6 minutes to read • Edit Online

You can use the Azure CLI to configure URL path-based routing rules when you create an application gateway. In this tutorial, you create backend pools using virtual machine scale sets. You then create URL routing rules that make sure web traffic is redirected to the appropriate backend pool.

In this tutorial, you learn how to:

- Set up the network
- Create an application gateway
- Add listeners and routing rules
- Create virtual machine scale sets for backend pools

The following example shows site traffic coming from both ports 8080 and 8081 and being directed to the same backend pools:

If you prefer, you can complete this tutorial using Azure PowerShell.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet named *myBackendSubnet* that's needed by the backend servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24

az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard
```

Create an application gateway

Use az network application-gateway create to create the application gateway named myAppGateway. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myPublicIPAddress* that you previously created.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_v2 \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 80 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Add backend pools and ports

You can add backend address pools named *imagesBackendPool* and *videoBackendPool* to your application gateway by using az network application-gateway address-pool create. You add the frontend ports for the pools using az network application-gateway frontend-port create.

```
az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name imagesBackendPool

az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name videoBackendPool

az network application-gateway frontend-port create \
  --port 8080 \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name bport

az network application-gateway frontend-port create \
  --port 8081 \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name rport
```

Add listeners and rules

Add listeners

Add the backend listeners named *backendListener* and *redirectedListener* that are needed to route traffic using az network application-gateway http-listener create.

```
az network application-gateway http-listener create \
  --name backendListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port bport \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway

az network application-gateway http-listener create \
  --name redirectedListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port rport \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway
```

Add the default URL path map

URL path maps make sure specific URLs are routed to specific backend pools. You can create URL path maps named *imagePathRule* and *videoPathRule* using az network application-gateway url-path-map create and az network application-gateway url-path-map rule create

```
az network application-gateway url-path-map create \
  --gateway-name myAppGateway \
  --name urlpathmap \
  --paths /images/* \
  --resource-group myResourceGroupAG \
  --address-pool imagesBackendPool \
  --default-address-pool appGatewayBackendPool \
  --default-http-settings appGatewayBackendHttpSettings \
  --http-settings appGatewayBackendHttpSettings \
  --rule-name imagePathRule

az network application-gateway url-path-map rule create \
  --gateway-name myAppGateway \
  --name videoPathRule \
  --resource-group myResourceGroupAG \
  --path-map-name urlpathmap \
  --paths /video/* \
  --address-pool videoBackendPool
```

Add redirection configuration

You can configure redirection for the listener using az network application-gateway redirect-config create.

```
az network application-gateway redirect-config create \
  --gateway-name myAppGateway \
  --name redirectConfig \
  --resource-group myResourceGroupAG \
  --type Found \
  --include-path true \
  --include-query-string true \
  --target-listener backendListener
```

Add the redirection URL path map

```
az network application-gateway url-path-map create \
  --gateway-name myAppGateway \
  --name redirectpathmap \
  --paths /images/* \
  --resource-group myResourceGroupAG \
  --redirect-config redirectConfig \
  --rule-name redirectPathRule
```

Add routing rules

The routing rules associate the URL path maps with the listeners that you created. You can add the rules named *defaultRule* and *redirectedRule* using az network application-gateway rule create.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name defaultRule \
  --resource-group myResourceGroupAG \
  --http-listener backendListener \
  --rule-type PathBasedRouting \
  --url-path-map urlpathmap \
  --address-pool appGatewayBackendPool

az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name redirectedRule \
  --resource-group myResourceGroupAG \
  --http-listener redirectedListener \
  --rule-type PathBasedRouting \
  --url-path-map redirectpathmap \
  --address-pool appGatewayBackendPool
```

Create virtual machine scale sets

In this example, you create three virtual machine scale sets that support the three backend pools that you created. The scale sets that you create are named *myvmss1*, *myvmss2*, and *myvmss3*. Each scale set contains two virtual machine instances on which you install NGINX.

Replace <azure-user> and <password> with a user name and password of your choice.

```
for i in `seq 1 3`; do
  if [ $i -eq 1 ]
  then
    poolName="appGatewayBackendPool"
  fi
  if [ $i -eq 2 ]
  then
    poolName="imagesBackendPool"
  fi
  if [ $i -eq 3 ]
  then
    poolName="videoBackendPool"
  fi

  az vmss create \
    --name myvmss$i \
    --resource-group myResourceGroupAG \
    --image UbuntuLTS \
    --admin-username <azure-user> \
    --admin-password <password> \
    --instance-count 2 \
    --vnet-name myVNet \
    --subnet myBackendSubnet \
    --vm-sku Standard_DS2 \
    --upgrade-policy-mode Automatic \
    --app-gateway myAppGateway \
    --backend-pool-name $poolName
done
```

Install NGINX

```
for i in `seq 1 3`; do
  az vmss extension set \
    --publisher Microsoft.Azure.Extensions \
    --version 2.0 \
    --name CustomScript \
    --resource-group myResourceGroupAG \
    --vmss-name myvmss$i \
    --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-samples/master/application-gateway/iis/install_nginx.sh"], "commandToExecute": "./install_nginx.sh" }'

done
```

Test the application gateway

To get the public IP address of the application gateway, use az network public-ip show. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://40.121.222.19`, `http://40.121.222.19:8080/images/test.htm`, `http://40.121.222.19:8080/video/test.htm`, or `http://40.121.222.19:8081/images/test.htm`.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

Change the URL to http://<ip-address>:8080/images/test.html, replacing your IP address for <ip-address>, and you should see something like the following example:

Change the URL to http://<ip-address>:8080/video/test.html, replacing your IP address for <ip-address>, and you should see something like the following example:

Now, change the URL to http://<ip-address>:8081/images/test.htm, replacing your IP address for <ip-address>, and you should see traffic redirected back to the images backend pool at http://<ip-address>:8080/images.

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroupAG
```

Next steps

Learn more about what you can do with application gateway

Tutorial: Create an application gateway that improves web application access

8/22/2022 • 4 minutes to read • Edit Online

If you're an IT admin concerned with improving web application access, you can optimize your application gateway to scale based on customer demand and span multiple availability zones. This tutorial helps you configure Azure Application Gateway features that do that: autoscaling, zone redundancy, and reserved VIPs (static IP). You'll use Azure PowerShell cmdlets and the Azure Resource Manager deployment model to solve the problem.

In this tutorial, you learn how to:

- Create a self-signed certificate
- Create an autoscale virtual network
- Create a reserved public IP
- Set up your application gateway infrastructure
- Specify autoscale
- Create the application gateway
- Test the application gateway

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This tutorial requires that you run an administrative Azure PowerShell session locally. You must have Azure PowerShell module version 1.0.0 or later installed. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. After you verify the PowerShell version, run `Connect-AzAccount` to create a connection with Azure.

Sign in to Azure

```
Connect-AzAccount
Select-AzSubscription -Subscription "<sub name>"
```

Create a resource group

Create a resource group in one of the available locations.

```
$location = "East US 2"
$rg = "AppGW-rg"

#Create a new Resource Group
New-AzResourceGroup -Name $rg -Location $location
```

Create a self-signed certificate

For production use, you should import a valid certificate signed by trusted provider. For this tutorial, you create a self-signed certificate using New-SelfSignedCertificate. You can use Export-PfxCertificate with the Thumbprint that was returned to export a pfx file from the certificate.

```
New-SelfSignedCertificate `
  -certstorelocation cert:\localmachine\my `
  -dnsname www.contoso.com
```

You should see something like this result:

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject
----------                                -------
E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630  CN=www.contoso.com
```

Use the thumbprint to create the pfx file. Replace *<password>* with a password of your choice:

```
$pwd = ConvertTo-SecureString -String "<password>" -Force -AsPlainText

Export-PfxCertificate `
  -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `
  -FilePath c:\appgwcert.pfx `
  -Password $pwd
```

Create a virtual network

Create a virtual network with one dedicated subnet for an autoscaling application gateway. Currently only one autoscaling application gateway can be deployed in each dedicated subnet.

```
#Create VNet with two subnets
$sub1 = New-AzVirtualNetworkSubnetConfig -Name "AppGwSubnet" -AddressPrefix "10.0.0.0/24"
$sub2 = New-AzVirtualNetworkSubnetConfig -Name "BackendSubnet" -AddressPrefix "10.0.1.0/24"
$vnet = New-AzvirtualNetwork -Name "AutoscaleVNet" -ResourceGroupName $rg `
       -Location $location -AddressPrefix "10.0.0.0/16" -Subnet $sub1, $sub2
```

Create a reserved public IP

Specify the allocation method of PublicIPAddress as **Static**. An autoscaling application gateway VIP can only be static. Dynamic IPs are not supported. Only the standard PublicIpAddress SKU is supported.

```
#Create static public IP
$pip = New-AzPublicIpAddress -ResourceGroupName $rg -name "AppGwVIP" `
       -location $location -AllocationMethod Static -Sku Standard -Zone 1,2,3
```

Retrieve details

Retrieve details of the resource group, subnet, and IP in a local object to create the IP configuration details for the application gateway.

```
$publicip = Get-AzPublicIpAddress -ResourceGroupName $rg -name "AppGwVIP"
$vnet = Get-AzvirtualNetwork -Name "AutoscaleVNet" -ResourceGroupName $rg
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name "AppGwSubnet" -VirtualNetwork $vnet
```

Create web apps

Configure two web apps for the backend pool. Replace *<site1-name>* and *<site-2-name>* with unique names in the `azurewebsites.net` domain.

```
New-AzAppServicePlan -ResourceGroupName $rg -Name "ASP-01"  -Location $location -Tier Basic `
   -NumberofWorkers 2 -WorkerSize Small
New-AzWebApp -ResourceGroupName $rg -Name <site1-name> -Location $location -AppServicePlan ASP-01
New-AzWebApp -ResourceGroupName $rg -Name <site2-name> -Location $location -AppServicePlan ASP-01
```

Configure the infrastructure

Configure the IP config, front-end IP config, back-end pool, HTTP settings, certificate, port, listener, and rule in an identical format to the existing Standard application gateway. The new SKU follows the same object model as the Standard SKU.

Replace your two web app FQDNs (for example: `mywebapp.azurewebsites.net`) in the $pool variable definition.

```
$ipconfig = New-AzApplicationGatewayIPConfiguration -Name "IPConfig" -Subnet $gwSubnet
$fip = New-AzApplicationGatewayFrontendIPConfig -Name "FrontendIPCOnfig" -PublicIPAddress $publicip
$pool = New-AzApplicationGatewayBackendAddressPool -Name "Pool1" `
       -BackendIPAddresses <your first web app FQDN>, <your second web app FQDN>
$fp01 = New-AzApplicationGatewayFrontendPort -Name "SSLPort" -Port 443
$fp02 = New-AzApplicationGatewayFrontendPort -Name "HTTPPort" -Port 80

$securepfxpwd = ConvertTo-SecureString -String "Azure123456!" -AsPlainText -Force
$sslCert01 = New-AzApplicationGatewaySslCertificate -Name "SSLCert" -Password $securepfxpwd `
            -CertificateFile "c:\appgwcert.pfx"
$listener01 = New-AzApplicationGatewayHttpListener -Name "SSLListener" `
             -Protocol Https -FrontendIPConfiguration $fip -FrontendPort $fp01 -SslCertificate $sslCert01
$listener02 = New-AzApplicationGatewayHttpListener -Name "HTTPListener" `
             -Protocol Http -FrontendIPConfiguration $fip -FrontendPort $fp02

$setting = New-AzApplicationGatewayBackendHttpSettings -Name "BackendHttpSetting1" `
          -Port 80 -Protocol Http -CookieBasedAffinity Disabled -PickHostNameFromBackendAddress
$rule01 = New-AzApplicationGatewayRequestRoutingRule -Name "Rule1" -RuleType basic `
         -BackendHttpSettings $setting -HttpListener $listener01 -BackendAddressPool $pool
$rule02 = New-AzApplicationGatewayRequestRoutingRule -Name "Rule2" -RuleType basic `
         -BackendHttpSettings $setting -HttpListener $listener02 -BackendAddressPool $pool
```

Specify autoscale

Now you can specify the autoscale configuration for the application gateway.

```
$autoscaleConfig = New-AzApplicationGatewayAutoscaleConfiguration -MinCapacity 2
$sku = New-AzApplicationGatewaySku -Name Standard_v2 -Tier Standard_v2
```

In this mode, the application gateway autoscales based on the application traffic pattern.

Create the application gateway

Create the application gateway and include redundancy zones and the autoscale configuration.

```
$appgw = New-AzApplicationGateway -Name "AutoscalingAppGw" -Zone 1,2,3 `
  -ResourceGroupName $rg -Location $location -BackendAddressPools $pool `
  -BackendHttpSettingsCollection $setting -GatewayIpConfigurations $ipconfig `
  -FrontendIpConfigurations $fip -FrontendPorts $fp01, $fp02 `
  -HttpListeners $listener01, $listener02 -RequestRoutingRules $rule01, $rule02 `
  -Sku $sku -sslCertificates $sslCert01 -AutoscaleConfiguration $autoscaleConfig
```

Test the application gateway

Use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address or DNS name, and then paste it into the address bar of your browser.

```
$pip = Get-AzPublicIPAddress -ResourceGroupName $rg -Name AppGwVIP
$pip.IpAddress
```

Clean up resources

First explore the resources that were created with the application gateway. Then, when they're no longer needed, you can use the `Remove-AzResourceGroup` command to remove the resource group, application gateway, and all related resources.

```
Remove-AzResourceGroup -Name $rg
```

Next steps

Create an application gateway with URL path-based routing rules

Tutorial: Enable the ingress controller add-on for a new AKS cluster with a new application gateway instance

8/22/2022 • 5 minutes to read • Edit Online

You can use the Azure CLI to enable the application gateway ingress controller (AGIC) add-on for a new Azure Kubernetes Services (AKS) cluster.

In this tutorial, you'll create an AKS cluster with the AGIC add-on enabled. Creating the cluster will automatically create an Azure application gateway instance to use. You'll then deploy a sample application that will use the add-on to expose the application through application gateway.

The add-on provides a much faster way to deploy AGIC for your AKS cluster than previously through Helm. It also offers a fully managed experience.

In this tutorial, you learn how to:

- Create a resource group.
- Create a new AKS cluster with the AGIC add-on enabled.
- Deploy a sample application by using AGIC for ingress on the AKS cluster.
- Check that the application is reachable through application gateway.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Create a resource group

In Azure, you allocate related resources to a resource group. Create a resource group by using az group create. The following example creates a resource group named **myResourceGroup** in the **East US** location (region):

```
az group create --name myResourceGroup --location eastus
```

Deploy an AKS cluster with the add-on enabled

You'll now deploy a new AKS cluster with the AGIC add-on enabled. If you don't provide an existing application gateway instance to use in this process, you'll automatically create and set up a new application gateway instance to serve traffic to the AKS cluster.

> **NOTE**
>
> The application gateway ingress controller add-on supports *only* application gateway v2 SKUs (Standard and WAF), and *not* the application gateway v1 SKUs. When you're deploying a new application gateway instance through the AGIC add-on, you can deploy only an application gateway Standard_v2 SKU. If you want to enable the add-on for an application gateway WAF_v2 SKU, use either of these methods:
>
> - Enable WAF on application gateway through the portal.
> - Create the WAF_v2 application gateway instance first, and then follow instructions on how to enable the AGIC add-on with an existing AKS cluster and existing application gateway instance.

In the following example, you'll deploy a new AKS cluster named *myCluster* by using Azure CNI and managed identities. The AGIC add-on will be enabled in the resource group that you created, **myResourceGroup**.

Deploying a new AKS cluster with the AGIC add-on enabled without specifying an existing application gateway instance will automatically create a Standard_v2 SKU application gateway instance. You'll need to specify a name and subnet address space for the new application gateway instance. The address space must be from 10.224.0.0/12 prefix used by the AKS virtual network without overlapping with 10.224.0.0/16 prefix used by the AKS subnet. In this tutorial, use *myApplicationGateway* for the application gateway name and *10.225.0.0/16* for its subnet address space.

```
az aks create -n myCluster -g myResourceGroup --network-plugin azure --enable-managed-identity -a ingress-
appgw --appgw-name myApplicationGateway --appgw-subnet-cidr "10.225.0.0/16" --generate-ssh-keys
```

To configure more parameters for the above command, see az aks create.

> **NOTE**
>
> The AKS cluster that you created will appear in the resource group that you created, **myResourceGroup**. However, the automatically created application gateway instance will be in the node resource group, where the agent pools are. The node resource group is named **MC_resource-group-name_cluster-name_location** by default, but can be modified.

Deploy a sample application by using AGIC

You'll now deploy a sample application to the AKS cluster that you created. The application will use the AGIC add-on for ingress and connect the application gateway instance to the AKS cluster.

First, get credentials to the AKS cluster by running the `az aks get-credentials` command:

```
az aks get-credentials -n myCluster -g myResourceGroup
```

Now that you have credentials, run the following command to set up a sample application that uses AGIC for ingress to the cluster. AGIC will update the application gateway instance that you set up earlier with corresponding routing rules to the sample application you're deploying.

```
kubectl apply -f https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-ingress/master/docs/examples/aspnetapp.yaml
```

Check that the application is reachable

Now that the application gateway instance is set up to serve traffic to the AKS cluster, let's verify that your application is reachable. First, get the IP address of the ingress:

```
kubectl get ingress
```

Check that the sample application that you created is running by either:

- Visiting the IP address of the application gateway instance that you got from running the preceding command.
- Using `curl`.

Application gateway might take a minute to get the update. If application gateway is still in an **Updating** state on the portal, let it finish before you try to reach the IP address.

Clean up resources

When you no longer need them, delete all resources created in this tutorial by deleting **myResourceGroup** and **MC_myResourceGroup_myCluster_eastus** resource groups:

```
az group delete --name myResourceGroup
az group delete --name MC_myResourceGroup_myCluster_eastus
```

Next steps

In this tutorial, you:

- Created new AKS cluster with the AGIC add-on enabled
- Deployed a sample application by using AGIC for ingress on the AKS cluster

To learn more about AGIC, see What is Application Gateway Ingress Controller? and Disable and re-enable AGIC add-on for your AKS cluster

To learn how to enable application gateway ingress controller add-on for an existing AKS cluster with an existing application gateway, advance to the next tutorial.

Enable AGIC for existing AKS and application gateway

Tutorial: Enable application gateway ingress controller add-on for an existing AKS cluster with an existing application gateway

8/22/2022 • 6 minutes to read • Edit Online

You can use Azure CLI or portal to enable the application gateway ingress controller (AGIC) add-on for an existing Azure Kubernetes Services (AKS) cluster. In this tutorial, you'll learn how to use AGIC add-on to expose your Kubernetes application in an existing AKS cluster through an existing application gateway deployed in separate virtual networks. You'll start by creating an AKS cluster in one virtual network and an application gateway in a separate virtual network to simulate existing resources. You'll then enable the AGIC add-on, peer the two virtual networks together, and deploy a sample application that will be exposed through the application gateway using the AGIC add-on. If you're enabling the AGIC add-on for an existing application gateway and existing AKS cluster in the same virtual network, then you can skip the peering step below. The add-on provides a much faster way of deploying AGIC for your AKS cluster than through Helm and also offers a fully managed experience.

In this tutorial, you learn how to:

- Create a resource group
- Create a new AKS cluster
- Create a new application gateway
- Enable the AGIC add-on in the existing AKS cluster through Azure CLI
- Enable the AGIC add-on in the existing AKS cluster through Azure portal
- Peer the application gateway virtual network with the AKS cluster virtual network
- Deploy a sample application using AGIC for ingress on the AKS cluster
- Check that the application is reachable through application gateway

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Create a resource group

In Azure, you allocate related resources to a resource group. Create a resource group by using az group create. The following example creates a resource group named **myResourceGroup** in the **East US** location (region):

```
az group create --name myResourceGroup --location eastus
```

Deploy a new AKS cluster

You'll now deploy a new AKS cluster, to simulate having an existing AKS cluster that you want to enable the AGIC add-on for.

In the following example, you'll be deploying a new AKS cluster named **myCluster** using Azure CNI and Managed Identities in the resource group you created, **myResourceGroup**.

```
az aks create -n myCluster -g myResourceGroup --network-plugin azure --enable-managed-identity --generate-ssh-keys
```

To configure more parameters for the above command, see az aks create.

> **NOTE**
> A node resource group will be created with the name **MC_resource-group-name_cluster-name_location**.

Deploy a new application gateway

You'll now deploy a new application gateway, to simulate having an existing application gateway that you want to use to load balance traffic to your AKS cluster, **myCluster**. The name of the application gateway will be **myApplicationGateway**, but you'll need to first create a public IP resource, named **myPublicIp**, and a new virtual network called **myVnet** with address space 10.0.0.0/16, and a subnet with address space 10.0.0.0/24 called **mySubnet**, and deploy your application gateway in **mySubnet** using **myPublicIp**.

Caution

When you use an AKS cluster and application gateway in separate virtual networks, the address spaces of the two virtual networks must not overlap. The default address space that an AKS cluster deploys in is 10.224.0.0/12.

```
az network public-ip create -n myPublicIp -g myResourceGroup --allocation-method Static --sku Standard
az network vnet create -n myVnet -g myResourceGroup --address-prefix 10.0.0.0/16 --subnet-name mySubnet --subnet-prefix 10.0.0.0/24
az network application-gateway create -n myApplicationGateway -l eastus -g myResourceGroup --sku Standard_v2 --public-ip-address myPublicIp --vnet-name myVnet --subnet mySubnet --priority 100
```

> **NOTE**
> application gateway ingress controller (AGIC) add-on **only** supports application gateway v2 SKUs (Standard and WAF), and **not** the application gateway v1 SKUs.

Enable the AGIC add-on in existing AKS cluster through Azure CLI

If you'd like to continue using Azure CLI, you can continue to enable the AGIC add-on in the AKS cluster you created, **myCluster**, and specify the AGIC add-on to use the existing application gateway you created,

myApplicationGateway.

```
appgwId=$(az network application-gateway show -n myApplicationGateway -g myResourceGroup -o tsv --query
"id")
az aks enable-addons -n myCluster -g myResourceGroup -a ingress-appgw --appgw-id $appgwId
```

Enable the AGIC add-on in existing AKS cluster through Azure portal

If you'd like to use Azure portal to enable AGIC add-on, go to (https://aka.ms/azure/portal/aks/agic) and navigate to your AKS cluster through the portal link. From there, go to the Networking tab within your AKS cluster. You'll see an application gateway ingress controller section, which allows you to enable/disable the ingress controller add-on using the Azure portal. Select the box next to **Enable ingress controller**, and then select the application gateway you created, **myApplicationGateway** from the dropdown menu. Select **Save**.

Peer the two virtual networks together

Since you deployed the AKS cluster in its own virtual network and the Application gateway in another virtual network, you'll need to peer the two virtual networks together in order for traffic to flow from the Application gateway to the pods in the cluster. Peering the two virtual networks requires running the Azure CLI command two separate times, to ensure that the connection is bi-directional. The first command will create a peering connection from the Application gateway virtual network to the AKS virtual network; the second command will create a peering connection in the other direction.

```
nodeResourceGroup=$(az aks show -n myCluster -g myResourceGroup -o tsv --query "nodeResourceGroup")
aksVnetName=$(az network vnet list -g $nodeResourceGroup -o tsv --query "[0].name")

aksVnetId=$(az network vnet show -n $aksVnetName -g $nodeResourceGroup -o tsv --query "id")
az network vnet peering create -n AppGWtoAKSVnetPeering -g myResourceGroup --vnet-name myVnet --remote-vnet
$aksVnetId --allow-vnet-access

appGWVnetId=$(az network vnet show -n myVnet -g myResourceGroup -o tsv --query "id")
az network vnet peering create -n AKStoAppGWVnetPeering -g $nodeResourceGroup --vnet-name $aksVnetName --
remote-vnet $appGWVnetId --allow-vnet-access
```

Deploy a sample application using AGIC

You'll now deploy a sample application to the AKS cluster you created that will use the AGIC add-on for Ingress and connect the application gateway to the AKS cluster. First, you'll get credentials to the AKS cluster you deployed by running the `az aks get-credentials` command.

```
az aks get-credentials -n myCluster -g myResourceGroup
```

Once you have the credentials to the cluster you created, run the following command to set up a sample application that uses AGIC for Ingress to the cluster. AGIC will update the application gateway you set up earlier with corresponding routing rules to the new sample application you deployed.

```
kubectl apply -f https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-ingress/master/docs/examples/aspnetapp.yaml
```

Check that the application is reachable

Now that the application gateway is set up to serve traffic to the AKS cluster, let's verify that your application is reachable. You'll first get the IP address of the Ingress.

```
kubectl get ingress
```

Check that the sample application you created is up and running by either visiting the IP address of the application gateway that you got from running the above command or check with `curl`. It may take application gateway a minute to get the update, so if the application gateway is still in an "Updating" state on Azure portal, then let it finish before trying to reach the IP address.

Clean up resources

When no longer needed, delete all resources created in this tutorial by deleting **myResourceGroup** and **MC_myResourceGroup_myCluster_eastus** resource groups:

```
az group delete --name myResourceGroup
az group delete --name MC_myResourceGroup_myCluster_eastus
```

Next steps

Learn more about disabling the AGIC add-on

Azure PowerShell examples for Azure Application Gateway (AG)

8/22/2022 • 2 minutes to read • Edit Online

The following table includes links to Azure PowerShell script examples for Azure Application Gateway.

EXAMPLE	DESCRIPTION
Manage web traffic	Creates an Application Gateway and all related resources.
Restrict web traffic	Creates an Application Gateway that restricts traffic using OWASP rules.
WAF v2 custom rules	Creates an Application Gateway Web Application Firewall v2 with custom rules.

Azure CLI examples for Azure Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

The following table includes links to Azure CLI script examples for Azure Application Gateway.

EXAMPLE	DESCRIPTION
Manage web traffic	Creates an application gateway and all related resources.
Restrict web traffic	Creates an application gateway that restricts traffic using OWASP rules.

Azure Resource Manager templates for Azure Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

The following table includes links to Azure Resource Manager templates for Azure Application Gateway.

EXAMPLE	DESCRIPTION
Application Gateway v2 with Web Application Firewall	Creates an Application Gateway v2 with Web Application Firewall v2.

Working remotely using Azure networking services

8/22/2022 • 7 minutes to read • Edit Online

> **NOTE**
> This article describes how you can leverage Azure networking services, Microsoft network, and the Azure partner ecosystem to work remotely and mitigate network issues that you might be facing because of the COVID-19 crisis.

This article describes the options that are available to organizations to set up remote access for their users or to supplement their existing solutions with additional capacity during periods of peak utilization. Network architects are faced with the following challenges:

- Address an increase in network utilization.
- Provide reliable-secure connectivity to more employees of their company and customers.
- Provide connectivity to remote locations across the globe.

Not all networks (for example, private WAN and corporate core networks) experience congestion from peak remote worker load. The bottlenecks are commonly reported only in home broadband networks and VPN gateways of on-premises networks of corporations.

Network planners can help ease the bottlenecks and alleviate the network congestion by keeping in mind that different traffic types need different network treatment priorities and by some smart load redirection/distribution. For example, real-time tele-medecine traffic of doctor-patient interaction is of high importance and delay/jitter sensitive. Whereas, replication of the same traffic between storages is not delay sensitive. The former traffic must be routed via the most optimal network path with higher quality of service; whereas it is acceptable to route the later traffic via sub-optimal route.

Sharing our best practices - Azure network is designed for elasticity and high-availability

Azure is designed to withstand sudden changes in the utilization of the resources and can greatly help during periods of peak utilization. Also, Microsoft maintains and operates one of the worlds' largest networks. Microsoft's network has been designed for high availability that can withstand different types of failure: from a single network element failure to failure of an entire region.

The Microsoft network is designed to meet the requirements and provide optimal performance for different types of network traffic including delay sensitive multimedia traffic for Skype and Teams, CDN, real-time big data analysis, Azure storage, Bing, and Xbox. To provide optimal performance for different types of traffic, the Microsoft network attracts all the traffic that is destined to- or wanting to transit through- its resources as close as possible to the origin of the traffic.

> **NOTE**
> Using the Azure networking features described below leverages the traffic attraction behavior of the Microsoft global network to provide a better customer networking experience. The traffic attraction behavior of the Microsoft network helps off loading traffic as soon as possible from the first/last mile networks that may experience congestion during periods of peak utilization.

Enable employees to work remotely

Azure VPN gateway supports both Point-to-Site (P2S) and Site-to-Site (S2S) VPN connections. Using the Azure VPN gateway you can scale your employee's connections to securely access both your Azure deployed resources and your on-premises resources. For more information, see How to enable users to work remotely.

If you are using Secure Sockets Tunneling Protocol (SSTP), the number of concurrent connections is limited to 128. To get a higher number of connections, we suggest transitioning to OpenVPN or IKEv2. For more information, see Transition to OpenVPN protocol or IKEv2 from SSTP.

To access your resources deployed in Azure, remote developers could use Azure Bastion solution, instead of VPN connection to get secure shell access (RDP or SSH) without requiring public IPs on the VMs being accessed. For more information, see Work remotely using Azure Bastion.

For aggregating large-scale VPN connection, to support any-to-any connections between resources in different on-premises global locations, in different regional hub and spoke virtual networks, and to optimize utilization of multiple home broadband networks you can use Azure Virtual WAN. For more information, see Struggling to cater to work from home needs? Here is where Azure Virtual WAN can help.

Another way to support a remote workforce is to deploy a Virtual Desktop Infrastructure (VDI) hosted in your Azure virtual network, secured with an Azure Firewall. For example, Azure Virtual Desktop (AVD) is a desktop and app virtualization service that runs in Azure. With Azure Virtual Desktop, you can set up a scalable and flexible environment in your Azure subscription without the need to run any additional gateway servers. You are only responsible for the AVD virtual machines in your virtual network. For more information, see Azure Firewall remote work support.

Azure also has a rich set of eco system partners. Our partners Network Virtual Appliances on Azure can also help scale VPN connectivity. For more information, see Network Virtual Appliance (NVA) considerations for remote work.

Extend employees' connection to access globally distributed resources

The following Azure services can help enable employees to access your globally distributed resources. Your resources could be in any of the Azure regions, on-premises networks, or even in other public or private clouds.

- **Azure virtual network peering**: If you deploy your resources in more than one Azure regions and/or if you aggregate the connectivity of remotely working employees using multiple virtual networks, you can establish connectivity between the multiple Azure virtual networks using virtual network peering. For more information, see Virtual network peering.
- **Azure VPN-based solution**: For your remote employees connected to Azure via P2S or S2S VPN, you can enable access to on-premises networks by configuring S2S VPN between your on-premises networks and Azure VPN gateway. For more information, see Create a Site-to-Site connection.
- **ExpressRoute**: Using ExpressRoute private peering you can enable private connectivity between your Azure deployments and on-premises infrastructure or your infrastructure in a co-location facility. ExpressRoute, via Microsoft peering, also permits accessing public endpoints in Microsoft from your on-premises network. ExpressRoute connections do not go over the public Internet. They offer secure connectivity, reliability, higher throughput, with lower and consistent latencies than typical connections over the Internet. For more information, see ExpressRoute overview. Leveraging your existing network provider that is already part of our ExpressRoute partner ecosystem can help reduce the time to get large bandwidth connections to Microsoft. Using ExpressRoute Direct you can directly connect your on-premises network to the Microsoft backbone. ExpressRoute Direct offers two different line-rate options of dual 10 Gbps or 100 Gbps.

- **Azure Virtual WAN**: Azure Virtual WAN allows seamless interoperability between your VPN connections and ExpressRoute circuits. As mentioned earlier, Azure Virtual WAN also support any-to-any connections between resources in different on-prem global locations, in different regional hub and spoke virtual networks

Scale customer connectivity to frontend resources

During times when more people go online, many corporate websites experience increased customer traffic. Azure Application Gateway can help manage this increased frontend workload. For more information, see Application Gateway high traffic support.

Microsoft support for multi-cloud traffic

For your deployments in other public clouds, Microsoft can provide global connectivity. Azure Virtual WAN, VPN or ExpressRoute can help in this regard. To extend connectivity from Azure to other clouds, you can configure S2S VPN between the two clouds. You can also establish connectivity from Azure to other public clouds using ExpressRoute. Oracle cloud is part of ExpressRoute partner ecosystem. You can set up a direct interconnection between Azure and Oracle Cloud Infrastructure. Most service providers that are part of the ExpressRoute partner ecosystem also offer private connectivity to other public clouds. Leveraging these service providers, you can establish private connectivity between your deployments in Azure and other clouds via ExpressRoute.

Next steps

The following articles discuss how different Azure networking features can be used to scale users to work remotely:

ARTICLE	DESCRIPTION
How to enable users to work remotely	Review available options to set up remote access for users or to supplement their existing solutions with additional capacity for your organization.
Struggling to cater to work from home needs? Here is where Azure Virtual WAN can help	Use Azure Virtual WAN to address the remote connectivity needs of your organization.
Application Gateway high traffic support	Use Application Gateway with Web Application Firewall (WAF) for a scalable and secure way to manage traffic to your web applications.
Network Virtual Appliance (NVA) considerations for remote work	Review guidance about leveraging NVAs in Azure to provide remote access solutions.
Transition to OpenVPN protocol or IKEv2 from SSTP	Overcome the 128 concurrent connection limit of SSTP by transitioning to OpenVPN protocol or IKEv2.
Working remotely using Azure Bastion	Provide secure and seamless RDP/SSH connectivity to virtual machines within the Azure virtual network, directly in the Azure portal, without the use of a public IP address.
Using Azure ExpressRoute to create hybrid connectivity to support remote users	Use ExpressRoute for hybrid connectivity to enable users in your organization to work remotely.
Azure Firewall remote work support	Protect your Azure virtual network resources using Azure Firewall.

Azure Application Gateway features

8/22/2022 • 8 minutes to read • Edit Online

Azure Application Gateway is a web traffic load balancer that enables you to manage traffic to your web applications.

Application Gateway includes the following features:

Secure Sockets Layer (SSL/TLS) termination

Application gateway supports SSL/TLS termination at the gateway, after which traffic typically flows unencrypted to the backend servers. This feature allows web servers to be unburdened from costly encryption and decryption overhead. But sometimes unencrypted communication to the servers isn't an acceptable option. This can be because of security requirements, compliance requirements, or the application may only accept a secure connection. For these applications, application gateway supports end to end SSL/TLS encryption.

For more information, see Overview of SSL termination and end to end SSL with Application Gateway

Autoscaling

Application Gateway Standard_v2 supports autoscaling and can scale up or down based on changing traffic load patterns. Autoscaling also removes the requirement to choose a deployment size or instance count during provisioning.

For more information about the Application Gateway Standard_v2 features, see What is Azure Application Gateway v2?.

Zone redundancy

A Standard_v2 Application Gateway can span multiple Availability Zones, offering better fault resiliency and removing the need to provision separate Application Gateways in each zone.

Static VIP

The application gateway Standard_v2 SKU supports static VIP type exclusively. This ensures that the VIP associated with application gateway doesn't change even over the lifetime of the Application Gateway.

Web Application Firewall

Web Application Firewall (WAF) is a service that provides centralized protection of your web applications from common exploits and vulnerabilities. WAF is based on rules from the OWASP (Open Web Application Security

Project) core rule sets 3.1 (WAF_v2 only), 3.0, and 2.2.9.

Web applications are increasingly targets of malicious attacks that exploit common known vulnerabilities. Common among these exploits are SQL injection attacks, cross site scripting attacks to name a few. Preventing such attacks in application code can be challenging and may require rigorous maintenance, patching and monitoring at many layers of the application topology. A centralized web application firewall helps make security management much simpler and gives better assurance to application administrators against threats or intrusions. A WAF solution can also react to a security threat faster by patching a known vulnerability at a central location versus securing each of individual web applications. Existing application gateways can be converted to a Web Application Firewall enabled application gateway easily.

For more information, see What is Azure Web Application Firewall?.

Ingress Controller for AKS

Application Gateway Ingress Controller (AGIC) allows you to use Application Gateway as the ingress for an Azure Kubernetes Service (AKS) cluster.

The ingress controller runs as a pod within the AKS cluster and consumes Kubernetes Ingress Resources and converts them to an Application Gateway configuration, which allows the gateway to load-balance traffic to the Kubernetes pods. The ingress controller only supports Application Gateway Standard_v2 and WAF_v2 SKUs.

For more information, see Application Gateway Ingress Controller (AGIC).

URL-based routing

URL Path Based Routing allows you to route traffic to back-end server pools based on URL Paths of the request. One of the scenarios is to route requests for different content types to different pool.

For example, requests for `http://contoso.com/video/*` are routed to VideoServerPool, and `http://contoso.com/images/*` are routed to ImageServerPool. DefaultServerPool is selected if none of the path patterns match.

For more information, see URL Path Based Routing overview.

Multiple-site hosting

With Application Gateway, you can configure routing based on host name or domain name for more than one web application on the same application gateway. It allows you to configure a more efficient topology for your deployments by adding up to 100+ websites to one application gateway. Each website can be directed to its own backend pool. For example, three domains, contoso.com, fabrikam.com, and adatum.com, point to the IP address of the application gateway. You'd create three multi-site listeners and configure each listener for the respective port and protocol setting.

Requests for `http://contoso.com` are routed to ContosoServerPool, `http://fabrikam.com` are routed to FabrikamServerPool, and so on.

Similarly, two subdomains of the same parent domain can be hosted on the same application gateway deployment. Examples of using subdomains could include `http://blog.contoso.com` and `http://app.contoso.com` hosted on a single application gateway deployment. For more information, see Application Gateway multiple site hosting.

You can also define wildcard host names in a multi-site listener and up to 5 host names per listener. To learn more, see wildcard host names in listener.

Redirection

A common scenario for many web applications is to support automatic HTTP to HTTPS redirection to ensure all communication between an application and its users occurs over an encrypted path.

In the past, you may have used techniques such as dedicated pool creation whose sole purpose is to redirect requests it receives on HTTP to HTTPS. Application gateway supports the ability to redirect traffic on the Application Gateway. This simplifies application configuration, optimizes the resource usage, and supports new redirection scenarios, including global and path-based redirection. Application Gateway redirection support isn't limited to HTTP to HTTPS redirection alone. This is a generic redirection mechanism, so you can redirect from and to any port you define using rules. It also supports redirection to an external site as well.

Application Gateway redirection support offers the following capabilities:

- Global redirection from one port to another port on the Gateway. This enables HTTP to HTTPS redirection on a site.
- Path-based redirection. This type of redirection enables HTTP to HTTPS redirection only on a specific site area, for example a shopping cart area denoted by `/cart/*`.
- Redirect to an external site.

For more information, see Application Gateway redirect overview.

Session affinity

The cookie-based session affinity feature is useful when you want to keep a user session on the same server. By using gateway-managed cookies, the Application Gateway can direct subsequent traffic from a user session to the same server for processing. This is important in cases where session state is saved locally on the server for a user session.

For more information, see How an application gateway works.

Websocket and HTTP/2 traffic

Application Gateway provides native support for the WebSocket and HTTP/2 protocols. There's no user-configurable setting to selectively enable or disable WebSocket support.

The WebSocket and HTTP/2 protocols enable full duplex communication between a server and a client over a long running TCP connection. This allows for a more interactive communication between the web server and the client, which can be bidirectional without the need for polling as required in HTTP-based implementations. These protocols have low overhead, unlike HTTP, and can reuse the same TCP connection for multiple request/responses resulting in a more efficient resource utilization. These protocols are designed to work over traditional HTTP ports of 80 and 443.

For more information, see WebSocket support and HTTP/2 support.

Connection draining

Connection draining helps you achieve graceful removal of backend pool members during planned service updates. This setting is enabled via the backend http setting and can be applied to all members of a backend pool during rule creation. Once enabled, Application Gateway ensures all deregistering instances of a backend pool don't receive any new request while allowing existing requests to complete within a configured time limit. This applies to both backend instances that are explicitly removed from the backend pool by a user configuration change, and backend instances that are reported as unhealthy as determined by the health probes. The only exception to this are requests bound for deregistering instances, which have been deregistered explicitly, because of gateway-managed session affinity and continues to be proxied to the deregistering instances.

For more information, see Application Gateway Configuration Overview.

Custom error pages

Application Gateway allows you to create custom error pages instead of displaying default error pages. You can use your own branding and layout using a custom error page.

For more information, see Custom Errors.

Rewrite HTTP headers and URL

HTTP headers allow the client and server to pass additional information with the request or the response. Rewriting these HTTP headers helps you accomplish several important scenarios, such as:

- Adding security-related header fields like HSTS/ X-XSS-Protection.
- Removing response header fields that can reveal sensitive information.
- Stripping port information from X-Forwarded-For headers.

Application Gateway and WAF v2 SKU supports the capability to add, remove, or update HTTP request and response headers, while the request and response packets move between the client and back-end pools. You can also rewrite URLs, query string parameters and host name. With URL rewrite and URL path-based routing, you can choose to either route requests to one of the backend pools based on the original path or the rewritten path, using the re-evaluate path map option.

It also provides you with the capability to add conditions to ensure the specified headers or URL are rewritten only when certain conditions are met. These conditions are based on the request and response information.

For more information, see Rewrite HTTP headers and URL.

Sizing

Application Gateway Standard_v2 can be configured for autoscaling or fixed size deployments. The v2 SKU doesn't offer different instance sizes. For more information on v2 performance and pricing, see Autoscaling V2 and Understanding pricing.

The Application Gateway Standard (v1) is offered in three sizes: **Small**, **Medium**, and **Large**. Small instance sizes are intended for development and testing scenarios.

For a complete list of application gateway limits, see Application Gateway service limits.

The following table shows an average performance throughput for each application gateway v1 instance with SSL offload enabled:

AVERAGE BACK-END PAGE RESPONSE SIZE	SMALL	MEDIUM	LARGE
6 KB	7.5 Mbps	13 Mbps	50 Mbps
100 KB	35 Mbps	100 Mbps	200 Mbps

> **NOTE**
>
> These values are approximate values for an application gateway throughput. The actual throughput depends on various environment details, such as average page size, location of back-end instances, and processing time to serve a page. For exact performance numbers, you should run your own tests. These values are only provided for capacity planning guidance.

Version feature comparison

For an Application Gateway v1-v2 feature comparison, see What is Azure Application Gateway v2?.

Next steps

- Learn how an application gateway works

How an application gateway works

8/22/2022 • 4 minutes to read • Edit Online

This article explains how an application gateway accepts incoming requests and routes them to the backend.

How an application gateway accepts a request

1. Before a client sends a request to an application gateway, it resolves the domain name of the application gateway by using a Domain Name System (DNS) server. Azure controls the DNS entry because all application gateways are in the azure.com domain.

2. The Azure DNS returns the IP address to the client, which is the frontend IP address of the application gateway.

3. The application gateway accepts incoming traffic on one or more listeners. A listener is a logical entity that checks for connection requests. It's configured with a frontend IP address, protocol, and port number for connections from clients to the application gateway.

4. If a web application firewall (WAF) is in use, the application gateway checks the request headers and the body, if present, against WAF rules. This action determines if the request is valid request or a security threat. If the request is valid, it's routed to the backend. If the request isn't valid and WAF is in Prevention mode, it's blocked as a security threat. If it's in Detection mode, the request is evaluated and logged, but still forwarded to the backend server.

Azure Application Gateway can be used as an internal application load balancer or as an internet-facing application load balancer. An internet-facing application gateway uses public IP addresses. The DNS name of an internet-facing application gateway is publicly resolvable to its public IP address. As a result, internet-facing application gateways can route client requests from the internet.

Internal application gateways use only private IP addresses. If you are using a Custom or Private DNS zone, the domain name should be internally resolvable to the private IP address of the Application Gateway. Therefore, internal load-balancers can only route requests from clients with access to a virtual network for the application gateway.

How an application gateway routes a request

If a request is valid and not blocked by WAF, the application gateway evaluates the request routing rule that's associated with the listener. This action determines which backend pool to route the request to.

Based on the request routing rule, the application gateway determines whether to route all requests on the listener to a specific backend pool, route requests to different backend pools based on the URL path, or redirect requests to another port or external site.

> **NOTE**
>
> Rules are processed in the order they're listed in the portal for v1 SKU.

When the application gateway selects the backend pool, it sends the request to one of the healthy backend servers in the pool (y.y.y.y). The health of the server is determined by a health probe. If the backend pool contains multiple servers, the application gateway uses a round-robin algorithm to route the requests between healthy servers. This load balances the requests on the servers.

After the application gateway determines the backend server, it opens a new TCP session with the backend server based on HTTP settings. HTTP settings specify the protocol, port, and other routing-related settings that are required to establish a new session with the backend server.

The port and protocol used in HTTP settings determine whether the traffic between the application gateway and backend servers is encrypted (thus accomplishing end-to-end TLS) or is unencrypted.

When an application gateway sends the original request to the backend server, it honors any custom configuration made in the HTTP settings related to overriding the hostname, path, and protocol. This action maintains cookie-based session affinity, connection draining, host-name selection from the backend, and so on.

> **NOTE**
>
> If the backend pool:
>
> - **Is a public endpoint**, the application gateway uses its frontend public IP to reach the server. If there isn't a frontend public IP address, one is assigned for the outbound external connectivity.
> - **Contains an internally resolvable FQDN or a private IP address**, the application gateway routes the request to the backend server by using its instance private IP addresses.
> - **Contains an external endpoint or an externally resolvable FQDN**, the application gateway routes the request to the backend server by using its frontend public IP address. The DNS resolution is based on a private DNS zone or custom DNS server, if configured, or it uses the default Azure-provided DNS. If there isn't a frontend public IP address, one is assigned for the outbound external connectivity.

Modifications to the request

Application gateway inserts six additional headers to all requests before it forwards the requests to the backend. These headers are x-forwarded-for, x-forwarded-port, x-forwarded-proto, x-original-host, x-original-url, and x-appgw-trace-id. The format for x-forwarded-for header is a comma-separated list of IP:port.

The valid values for x-forwarded-proto are HTTP or HTTPS. X-forwarded-port specifies the port where the request reached the application gateway. X-original-host header contains the original host header with which the request arrived. This header is useful in Azure website integration, where the incoming host header is modified before traffic is routed to the backend. If session affinity is enabled as an option, then it adds a gateway-managed affinity cookie.

X-appgw-trace-id is a unique guid generated by application gateway for each client request and presented in the forwarded request to the backend pool member. The guid consists of 32 alphanumeric characters presented without dashes (for example: ac882cd65a2712a0fe1289ec2bb6aee7). This guid can be used to correlate a request received by application gateway and initiated to a backend pool member via the transactionId property in Diagnostic Logs.

You can configure application gateway to modify request and response headers and URL by using Rewrite HTTP

headers and URL or to modify the URI path by using a path-override setting. However, unless configured to do so, all incoming requests are proxied to the backend.

Next steps

Learn about application gateway components

Application gateway components

8/22/2022 • 8 minutes to read • Edit Online

An application gateway serves as the single point of contact for clients. It distributes incoming application traffic across multiple backend pools, which include Azure VMs, virtual machine scale sets, Azure App Service, and on-premises/external servers. To distribute traffic, an application gateway uses several components described in this article.

Frontend IP addresses

A frontend IP address is the IP address associated with an application gateway. You can configure an application gateway to have a public IP address, a private IP address, or both. An application gateway supports one public or one private IP address. Your virtual network and public IP address must be in the same location as your application gateway. After it's created, a frontend IP address is associated with a listener.

Static versus dynamic public IP address

The Azure Application Gateway V2 SKU can be configured to support either both static internal IP address and static public IP address, or only static public IP address. It cannot be configured to support only static internal IP address.

The V1 SKU can be configured to support static or dynamic internal IP address and dynamic public IP address. The dynamic IP address of Application Gateway does not change on a running gateway. It can change only when you stop or start the Gateway. It does not change on system failures, updates, Azure host updates etc.

The DNS name associated with an application gateway doesn't change over the lifecycle of the gateway. As a result, you should use a CNAME alias and point it to the DNS address of the application gateway.

Listeners

A listener is a logical entity that checks for incoming connection requests. A listener accepts a request if the protocol, port, hostname, and IP address associated with the request match the same elements associated with the listener configuration.

Before you use an application gateway, you must add at least one listener. There can be multiple listeners attached to an application gateway, and they can be used for the same protocol.

After a listener detects incoming requests from clients, the application gateway routes these requests to members in the backend pool configured in the rule.

Listeners support the following ports and protocols.

Ports

A port is where a listener listens for the client request. You can configure ports for v1 and v2 SKUs as per below.

SKU	SUPPORTED PORT RANGE	EXCEPTION(S)
V2	1 to 64999	22
V1	1 to 65502	3389

Protocols

Application Gateway supports four protocols: HTTP, HTTPS, HTTP/2, and WebSocket:

> **NOTE**
> HTTP/2 protocol support is available to clients connecting to application gateway listeners only. The communication to backend server pools is always over HTTP/1.1. By default, HTTP/2 support is disabled. You can choose to enable it.

- Specify between the HTTP and HTTPS protocols in the listener configuration.
- Support for WebSockets and HTTP/2 protocols is provided natively, and WebSocket support is enabled by default. There's no user-configurable setting to selectively enable or disable WebSocket support. Use WebSockets with both HTTP and HTTPS listeners.

Use an HTTPS listener for TLS termination. An HTTPS listener offloads the encryption and decryption work to your application gateway, so your web servers aren't burdened by the overhead.

Custom error pages

Application Gateway lets you create custom error pages instead of displaying default error pages. You can use your own branding and layout using a custom error page. Application Gateway displays a custom error page when a request can't reach the backend.

For more information, see Custom error pages for your application gateway.

Types of listeners

There are two types of listeners:

- **Basic**. This type of listener listens to a single domain site, where it has a single DNS mapping to the IP address of the application gateway. This listener configuration is required when you host a single site behind an application gateway.

- **Multi-site**. This listener configuration is required when you want to configure routing based on host name or domain name for more than one web application on the same application gateway. It allows you to configure a more efficient topology for your deployments by adding up to 100+ websites to one

application gateway. Each website can be directed to its own backend pool. For example, three domains, contoso.com, fabrikam.com, and adatum.com, point to the IP address of the application gateway. You'd create three multi-site listeners and configure each listener for the respective port and protocol setting.

You can also define wildcard host names in a multi-site listener and up to 5 host names per listener. To learn more, see wildcard host names in listener.

For more information on how to configure a multi-site listener, see Multiple-site hosting in Application Gateway using Azure portal.

After you create a listener, you associate it with a request routing rule. This rule determines how the request received on the listener should be routed to the backend. The request routing rule also contains the backend pool to be routed to and the HTTP setting where the backend port, protocol, etc. are mentioned.

Request routing rules

A request routing rule is a key component of an application gateway because it determines how to route traffic on the listener. The rule binds the listener, the back-end server pool, and the backend HTTP settings.

When a listener accepts a request, the request routing rule forwards the request to the backend or redirects it elsewhere. If the request is forwarded to the backend, the request routing rule defines which backend server pool to forward it to. The request routing rule also determines if the headers in the request are to be rewritten. One listener can be attached to one rule.

There are two types of request routing rules:

- **Basic**. All requests on the associated listener (for example, blog.contoso.com/*) are forwarded to the associated backend pool by using the associated HTTP setting.
- **Path-based**. This routing rule lets you route the requests on the associated listener to a specific backend pool, based on the URL in the request. If the path of the URL in a request matches the path pattern in a path-based rule, the rule routes that request. It applies the path pattern only to the URL path, not to its query parameters. If the URL path on a listener request doesn't match any of the path-based rules, it routes the request to the default backend pool and HTTP settings.

For more information, see URL-based routing.

Redirection support

The request routing rule also allows you to redirect traffic on the application gateway. This is a generic redirection mechanism, so you can redirect to and from any port you define by using rules.

You can choose the redirection target to be another listener (which can help enable automatic HTTP to HTTPS redirection) or an external site. You can also choose to have the redirection be temporary or permanent, or to append the URI path and query string to the redirected URL.

For more information, see Redirect traffic on your application gateway.

Rewrite HTTP headers and URL

By using rewrite rules, you can add, remove, or update HTTP(S) request and response headers as well as URL path and query string parameters as the request and response packets move between the client and backend pools via the application gateway.

The headers and URL parameters can be set to static values or to other headers and server variables. This helps with important use cases, such as extracting client IP addresses, removing sensitive information about the backend, adding more security, and so on.

For more information, see Rewrite HTTP headers and URL on your application gateway.

HTTP settings

An application gateway routes traffic to the backend servers (specified in the request routing rule that include HTTP settings) by using the port number, protocol, and other settings detailed in this component.

The port and protocol used in the HTTP settings determine whether the traffic between the application gateway and backend servers is encrypted (providing end-to-end TLS) or unencrypted.

This component is also used to:

- Determine whether a user session is to be kept on the same server by using the cookie-based session affinity.
- Gracefully remove backend pool members by using connection draining.
- Associate a custom probe to monitor the backend health, set the request timeout interval, override host name and path in the request, and provide one-click ease to specify settings for the App Service backend.

Backend pools

A backend pool routes request to backend servers, which serve the request. Backend pools can contain:

- NICs
- Virtual machine scale sets
- Public IP addresses
- Internal IP addresses
- FQDN
- Multitenant backends (such as App Service)

Application Gateway backend pool members aren't tied to an availability set. An application gateway can communicate with instances outside of the virtual network that it's in. As a result, the members of the backend pools can be across clusters, across datacenters, or outside Azure, as long as there's IP connectivity.

If you use internal IPs as backend pool members, you must use virtual network peering or a VPN gateway. Virtual network peering is supported and beneficial for load-balancing traffic in other virtual networks.

An application gateway can also communicate with on-premises servers when they're connected by Azure ExpressRoute or VPN tunnels if traffic is allowed.

You can create different backend pools for different types of requests. For example, create one backend pool for general requests, and then another backend pool for requests to the microservices for your application.

Health probes

By default, an application gateway monitors the health of all resources in its backend pool and automatically removes unhealthy ones. It then monitors unhealthy instances and adds them back to the healthy backend pool when they become available and respond to health probes.

In addition to using default health probe monitoring, you can also customize the health probe to suit your application's requirements. Custom probes allow more granular control over the health monitoring. When using custom probes, you can configure a custom hostname, URL path, probe interval, and how many failed responses to accept before marking the back-end pool instance as unhealthy, custom status codes and response body match, etc. We recommend that you configure custom probes to monitor the health of each backend pool.

For more information, see Monitor the health of your application gateway.

Next steps

Create an application gateway:

- In the Azure portal
- By using Azure PowerShell
- By using the Azure CLI

Application Gateway configuration overview

8/22/2022 • 2 minutes to read • Edit Online

Azure Application Gateway consists of several components that you can configure in various ways for different scenarios. This article shows you how to configure each component.

This image illustrates an application that has three listeners. The first two are multi-site listeners for `http://acme.com/*` and `http://fabrikam.com/*`, respectively. Both listen on port 80. The third is a basic listener that has end-to-end Transport Layer Security (TLS) termination, previously known as Secure Sockets Layer (SSL) termination.

Infrastructure

The Application Gateway infrastructure includes the virtual network, subnets, network security groups, and user defined routes.

For more information, see Application Gateway infrastructure configuration.

Front-end IP address

You can configure the application gateway to have a public IP address, a private IP address, or both. A public IP is required when you host a back end that clients must access over the Internet via an Internet-facing virtual IP (VIP).

For more information, see Application Gateway front-end IP address configuration.

Listeners

A listener is a logical entity that checks for incoming connection requests by using the port, protocol, host, and IP address. When you configure the listener, you must enter values for these that match the corresponding values in the incoming request on the gateway.

For more information, see Application Gateway listener configuration.

Request routing rules

When you create an application gateway by using the Azure portal, you create a default rule (*rule1*). This rule binds the default listener (*appGatewayHttpListener*) with the default back-end pool (*appGatewayBackendPool*) and the default back-end HTTP settings (*appGatewayBackendHttpSettings*). After you create the gateway, you can edit the settings of the default rule or create new rules.

For more information, see Application Gateway request routing rules.

HTTP settings

The application gateway routes traffic to the back-end servers by using the configuration that you specify here. After you create an HTTP setting, you must associate it with one or more request-routing rules.

For more information, see Application Gateway HTTP settings configuration.

Back-end pool

You can point a back-end pool to four types of backend members: a specific virtual machine, a virtual machine scale set, an IP address/FQDN, or an app service.

After you create a back-end pool, you must associate it with one or more request-routing rules. You must also configure health probes for each back-end pool on your application gateway. When a request-routing rule condition is met, the application gateway forwards the traffic to the healthy servers (as determined by the health probes) in the corresponding back-end pool.

Health probes

An application gateway monitors the health of all resources in its back end by default. But we strongly recommend that you create a custom probe for each back-end HTTP setting to get greater control over health monitoring. To learn how to configure a custom probe, see Custom health probe settings.

> **NOTE**
>
> After you create a custom health probe, you need to associate it to a back-end HTTP setting. A custom probe won't monitor the health of the back-end pool unless the corresponding HTTP setting is explicitly associated with a listener using a rule.

Next steps

Now that you know about Application Gateway components, you can:

- Create an application gateway in the Azure portal
- Create an application gateway using PowerShell
- Create an application gateway using the Azure CLI

Application Gateway infrastructure configuration

8/22/2022 • 7 minutes to read • Edit Online

The application gateway infrastructure includes the virtual network, subnets, network security groups, and user defined routes.

Virtual network and dedicated subnet

An application gateway is a dedicated deployment in your virtual network. Within your virtual network, a dedicated subnet is required for the application gateway. You can have multiple instances of a given application gateway deployment in a subnet. You can also deploy other application gateways in the subnet. But you can't deploy any other resource in the application gateway subnet. You can't mix v1 and v2 Azure Application Gateway SKUs on the same subnet.

> **NOTE**
>
> Virtual network service endpoint policies are currently not supported in an Application Gateway subnet.

Size of the subnet

Application Gateway uses one private IP address per instance, plus another private IP address if a private front-end IP is configured.

Azure also reserves five IP addresses in each subnet for internal use: the first four and the last IP addresses. For example, consider 15 application gateway instances with no private front-end IP. You need at least 20 IP addresses for this subnet: five for internal use and 15 for the application gateway instances.

Consider a subnet that has 27 application gateway instances and an IP address for a private front-end IP. In this case, you need 33 IP addresses: 27 for the application gateway instances, one for the private front end, and five for internal use.

Application Gateway (Standard or WAF) SKU can support up to 32 instances (32 instance IP addresses + 1 private frontend IP configuration + 5 Azure reserved) – so a minimum subnet size of /26 is recommended

Application Gateway (Standard_v2 or WAF_v2 SKU) can support up to 125 instances (125 instance IP addresses + 1 private frontend IP configuration + 5 Azure reserved). A minimum subnet size of /24 is recommended.

To determine the available capacity of a subnet that has existing Application Gateways provisioned, take the size of the subnet and subtract the five reserved IP addresses of the subnet reserved by the platform. Next, take each gateway and subtract the the max-instance count. For each gateway that has a private frontend IP configuration, subtract one additional IP address per gateway as well.

For example, here's how to calculate the available addressing for a subnet with three gateways of varying sizes:

- Gateway 1: Maximum of 10 instances; utilizes a private frontend IP configuration
- Gateway 2: Maximum of 2 instances; no private frontend IP configuration
- Gateway 3: Maximum of 15 instances; utilizes a private frontend IP configuration
- Subnet Size: /24

Subnet Size /24 = 255 IP addresses - 5 reserved from the platform = 250 available addresses. 250 - Gateway 1 (10) - 1 private frontend IP configuration = 239 239 - Gateway 2 (2) = 237 237 - Gateway 3 (15) - 1 private frontend IP configuration = 221

> **IMPORTANT**
> Although a /24 subnet is not required per Application Gateway v2 SKU deployment, it is highly recommended. This is to ensure that Application Gateway v2 has sufficient space for autoscaling expansion and maintenance upgrades. You should ensure that the Application Gateway v2 subnet has sufficient address space to accommodate the number of instances required to serve your maximum expected traffic. If you specify the maximum instance count, then the subnet should have capacity for at least that many addresses. For capacity planning around instance count, see instance count details.

> **TIP**
> IP addresses are allocated from the beginning of the defined subnet space for gateway instances. As instances are created and removed due to creation of gateways or scaling events, it can become difficult to understand what the next available address is in the subnet. To be able to determine the next address to use for a future gateway and have a contiguous addressing theme for frontend IPs, consider assigning frontend IP addresses from the upper half of the defined subset space. For example, if my subnet address space is 10.5.5.0/24, consider setting the private frontend IP configuration of your gateways starting with 10.5.5.254 and then following with 10.5.5.253, 10.5.5.252, 10.5.5.251, and so forth for future gateways.

> **TIP**
> It is possible to change the subnet of an existing Application Gateway within the same virtual network. You can do this using Azure PowerShell or Azure CLI. For more information, see Frequently asked questions about Application Gateway

Network security groups

Network security groups (NSGs) are supported on Application Gateway. But there are some restrictions:

- You must allow incoming Internet traffic on TCP ports 65503-65534 for the Application Gateway v1 SKU, and TCP ports 65200-65535 for the v2 SKU with the destination subnet as **Any** and source as **GatewayManager** service tag. This port range is required for Azure infrastructure communication. These ports are protected (locked down) by Azure certificates. External entities, including the customers of those gateways, can't communicate on these endpoints.
- Outbound Internet connectivity can't be blocked. Default outbound rules in the NSG allow Internet connectivity. We recommend that you:
  - Don't remove the default outbound rules.
  - Don't create other outbound rules that deny any outbound connectivity.
- Traffic from the **AzureLoadBalancer** tag with the destination subnet as **Any** must be allowed.

Allow access to a few source IPs

For this scenario, use NSGs on the Application Gateway subnet. Put the following restrictions on the subnet in this order of priority:

1. Allow incoming traffic from a source IP or IP range with the destination as the entire Application Gateway subnet address range and destination port as your inbound access port, for example, port 80 for HTTP access.
2. Allow incoming requests from source as **GatewayManager** service tag and destination as **Any** and destination ports as 65503-65534 for the Application Gateway v1 SKU, and ports 65200-65535 for v2 SKU for back-end health status communication. This port range is required for Azure infrastructure communication. These ports are protected (locked down) by Azure certificates. Without appropriate certificates in place, external entities can't initiate changes on those endpoints.

3. Allow incoming Azure Load Balancer probes (*AzureLoadBalancer* tag) on the network security group.
4. Allow expected inbound traffic to match your listener configuration (i.e. if you have listeners configured for port 80, you will want an allow inbound rule for port 80)
5. Block all other incoming traffic by using a deny-all rule.
6. Allow outbound traffic to the Internet for all destinations.

Supported user-defined routes

> **IMPORTANT**
>
> Using UDRs on the Application Gateway subnet might cause the health status in the back-end health view to appear as **Unknown**. It also might cause generation of Application Gateway logs and metrics to fail. We recommend that you don't use UDRs on the Application Gateway subnet so that you can view the back-end health, logs, and metrics.

- **v1**

  For the v1 SKU, user-defined routes (UDRs) are supported on the Application Gateway subnet, as long as they don't alter end-to-end request/response communication. For example, you can set up a UDR in the Application Gateway subnet to point to a firewall appliance for packet inspection. But you must make sure that the packet can reach its intended destination after inspection. Failure to do so might result in incorrect health-probe or traffic-routing behavior. This includes learned routes or default 0.0.0.0/0 routes that are propagated by Azure ExpressRoute or VPN gateways in the virtual network.

- **v2**

  For the v2 SKU, there are supported and unsupported scenarios:

  v2 supported scenarios

  > **WARNING**
  >
  > An incorrect configuration of the route table could result in asymmetrical routing in Application Gateway v2. Ensure that all management/control plane traffic is sent directly to the Internet and not through a virtual appliance. Logging, metrics, and CRL checks could also be affected.

  Scenario 1: UDR to disable Border Gateway Protocol (BGP) Route Propagation to the Application Gateway subnet

  Sometimes the default gateway route (0.0.0.0/0) is advertised via the ExpressRoute or VPN gateways associated with the Application Gateway virtual network. This breaks management plane traffic, which requires a direct path to the Internet. In such scenarios, a UDR can be used to disable BGP route propagation.

  To disable BGP route propagation, use the following steps:

  1. Create a Route Table resource in Azure.
  2. Disable the **Virtual network gateway route propagation** parameter.
  3. Associate the Route Table to the appropriate subnet.

  Enabling the UDR for this scenario shouldn't break any existing setups.

  Scenario 2: UDR to direct 0.0.0.0/0 to the Internet

  You can create a UDR to send 0.0.0.0/0 traffic directly to the Internet.

  Scenario 3: UDR for Azure Kubernetes Service with kubenet

If you're using kubenet with Azure Kubernetes Service (AKS) and Application Gateway Ingress Controller (AGIC), you'll need a route table to allow traffic sent to the pods from Application Gateway to be routed to the correct node. This won't be necessary if you use Azure CNI.

To use the route table to allow kubenet to work, follow the steps below:

1. Go to the resource group created by AKS (the name of the resource group should begin with "MC_")
2. Find the route table created by AKS in that resource group. The route table should be populated with the following information:
   - Address prefix should be the IP range of the pods you want to reach in AKS.
   - Next hop type should be Virtual Appliance.
   - Next hop address should be the IP address of the node hosting the pods.
3. Associate this route table to the Application Gateway subnet.

v2 unsupported scenarios

Scenario 1: UDR for Virtual Appliances

Any scenario where 0.0.0.0/0 needs to be redirected through any virtual appliance, a hub/spoke virtual network, or on-premises (forced tunneling) isn't supported for V2.

Next steps

- Learn about front-end IP address configuration.

Application Gateway front-end IP address configuration

8/22/2022 • 2 minutes to read • Edit Online

You can configure the application gateway to have a public IP address, a private IP address, or both. A public IP address is required when you host a back end that clients must access over the Internet via an Internet-facing virtual IP (VIP).

Public and private IP address support

Application Gateway V2 currently does not support only private IP mode. It supports the following combinations:

- Private IP address and public IP address
- Public IP address only

For more information, see Frequently asked questions about Application Gateway.

A public IP address isn't required for an internal endpoint that's not exposed to the Internet. That's known as an *internal load-balancer* (ILB) endpoint or private frontend IP. An application gateway ILB is useful for internal line-of-business applications that aren't exposed to the Internet. It's also useful for services and tiers in a multi-tier application within a security boundary that aren't exposed to the Internet but that require round-robin load distribution, session stickiness, or TLS termination.

Only one public IP address and one private IP address is supported. You choose the front-end IP when you create the application gateway.

- For a public IP address, you can create a new public IP address or use an existing public IP in the same location as the application gateway. For more information, see static vs. dynamic public IP address.
- For a private IP address, you can specify a private IP address from the subnet where the application gateway is created. For Application Gateway v2 sku deployments, a static IP address must be defined when adding a private IP address to the gateway. For Application Gateway v1 sku deployments, if you don't specify an IP address, an available IP address is automatically selected from the subnet. The IP address type that you select (static or dynamic) can't be changed later. For more information, see Create an application gateway with an internal load balancer.

A front-end IP address is associated to a *listener*, which checks for incoming requests on the front-end IP.

Next steps

- Learn about listener configuration

Application Gateway listener configuration

8/22/2022 • 4 minutes to read • Edit Online

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

A listener is a logical entity that checks for incoming connection requests by using the port, protocol, host, and IP address. When you configure the listener, you must enter values for these that match the corresponding values in the incoming request on the gateway.

When you create an application gateway by using the Azure portal, you also create a default listener by choosing the protocol and port for the listener. You can choose whether to enable HTTP2 support on the listener. After you create the application gateway, you can edit the settings of that default listener (*appGatewayHttpListener*) or create new listeners.

Listener type

When you create a new listener, you choose between *basic* and *multi-site*.

- If you want all of your requests (for any domain) to be accepted and forwarded to backend pools, choose basic. Learn how to create an application gateway with a basic listener.
- If you want to forward requests to different backend pools based on the *host* header or host names, choose multi-site listener, where you must also specify a host name that matches with the incoming request. This is because Application Gateway relies on HTTP 1.1 host headers to host more than one website on the same public IP address and port. To learn more, see hosting multiple sites using Application Gateway.

Order of processing listeners

For the v1 SKU, requests are matched according to the order of the rules and the type of listener. If a rule with basic listener comes first in the order, it's processed first and will accept any request for that port and IP combination. To avoid this, configure the rules with multi-site listeners first and push the rule with the basic listener to the last in the list.

For the v2 SKU, multi-site listeners are processed before basic listeners.

Front-end IP address

Choose the front-end IP address that you plan to associate with this listener. The listener will listen to incoming requests on this IP.

Front-end port

Choose the front-end port. Select an existing port or create a new one. Choose any value from the allowed range of ports. You can use not only well-known ports, such as 80 and 443, but any allowed custom port that's suitable. A port can be used for public-facing listeners or private-facing listeners.

Protocol

Choose HTTP or HTTPS:

- If you choose HTTP, the traffic between the client and the application gateway is unencrypted.
- Choose HTTPS if you want TLS termination or end-to-end TLS encryption. The traffic between the client and the application gateway is encrypted. And the TLS connection terminates at the application gateway. If you want end-to-end TLS encryption, you must choose HTTPS and configure the **back-end HTTP** setting. This ensures that traffic is re-encrypted when it travels from the application gateway to the back end.

To configure TLS termination, a TLS/SSL certificate must be added to the listener. This allows the Application Gateway to decrypt incoming traffic and encrypt response traffic to the client. The certificate provided to the Application Gateway must be in Personal Information Exchange (PFX) format, which contains both the private and public keys.

> **NOTE**
> When using a TLS certificate from Key Vault for a listener, you must ensure your Application Gateway always has access to that linked key vault resource and the certificate object within it. This enables seamless operations of TLS termination feature and maintains the overall health of your gateway resource. If an application gateway resource detects a misconfigured key vault, it automatically puts the associated HTTPS listener(s) in a disabled state. Learn more.

Supported certificates

See Overview of TLS termination and end to end TLS with Application Gateway

Additional protocol support

HTTP2 support

HTTP/2 protocol support is available to clients that connect to application gateway listeners only. The communication to back-end server pools is over HTTP/1.1. By default, HTTP/2 support is disabled. The following Azure PowerShell code snippet shows how to enable this:

```
$gw = Get-AzApplicationGateway -Name test -ResourceGroupName hm

$gw.EnableHttp2 = $true

Set-AzApplicationGateway -ApplicationGateway $gw
```

WebSocket support

WebSocket support is enabled by default. There's no user-configurable setting to enable or disable it. You can use WebSockets with both HTTP and HTTPS listeners.

Custom error pages

You can define custom error at the global level or the listener level. But creating global-level custom error pages from the Azure portal is currently not supported. You can configure a custom error page for a 403 web application firewall error or a 502 maintenance page at the listener level. You must also specify a publicly accessible blob URL for the given error status code. For more information, see Create Application Gateway custom error pages.

To configure a global custom error page, see Azure PowerShell configuration.

TLS policy

You can centralize TLS/SSL certificate management and reduce encryption-decryption overhead for a back-end server farm. Centralized TLS handling also lets you specify a central TLS policy that's suited to your security requirements. You can choose *default*, *predefined*, or *custom* TLS policy.

You configure TLS policy to control TLS protocol versions. You can configure an application gateway to use a minimum protocol version for TLS handshakes from TLS1.0, TLS1.1, and TLS1.2. By default, SSL 2.0 and 3.0 are disabled and aren't configurable. For more information, see Application Gateway TLS policy overview.

After you create a listener, you associate it with a request-routing rule. That rule determines how requests that are received on the listener are routed to the back end.

Next steps

- Learn about request routing rules.

Application Gateway request routing rules

8/22/2022 • 4 minutes to read • Edit Online

When you create an application gateway using the Azure portal, you create a default rule (*rule1*). This rule binds the default listener (*appGatewayHttpListener*) with the default back-end pool (*appGatewayBackendPool*) and the default back-end HTTP settings (*appGatewayBackendHttpSettings*). After you create the gateway, you can edit the settings of the default rule or create new rules.

Rule type

When you create a rule, you choose between *basic* and *path-based*.

- Choose basic if you want to forward all requests on the associated listener (for example, *blog.contoso.com/*)* to a single back-end pool.
- Choose path-based if you want to route requests from specific URL paths to specific back-end pools. The path pattern is applied only to the path of the URL, not to its query parameters.

Order of processing rules

For the v1 and v2 SKU, pattern matching of incoming requests is processed in the order that the paths are listed in the URL path map of the path-based rule. If a request matches the pattern in two or more paths in the path map, the path that's listed first is matched. And the request is forwarded to the back end that's associated with that path.

Associated listener

Associate a listener to the rule so that the *request-routing rule* that's associated with the listener is evaluated to determine the back-end pool to route the request to.

Associated back-end pool

Associate to the rule the back-end pool that contains the back-end targets that serve requests that the listener receives.

- For a basic rule, only one back-end pool is allowed. All requests on the associated listener are forwarded to that back-end pool.
- For a path-based rule, add multiple back-end pools that correspond to each URL path. The requests that match the URL path that's entered are forwarded to the corresponding back-end pool. Also, add a default back-end pool. Requests that don't match any URL path in the rule are forwarded to that pool.

Associated back-end HTTP setting

Add a back-end HTTP setting for each rule. Requests are routed from the application gateway to the back-end targets by using the port number, protocol, and other information that's specified in this setting.

For a basic rule, only one back-end HTTP setting is allowed. All requests on the associated listener are forwarded to the corresponding back-end targets by using this HTTP setting.

For a path-based rule, add multiple back-end HTTP settings that correspond to each URL path. Requests that match the URL path in this setting are forwarded to the corresponding back-end targets by using the HTTP settings that correspond to each URL path. Also, add a default HTTP setting. Requests that don't match any URL path in this rule are forwarded to the default back-end pool by using the default HTTP setting.

Redirection setting

If redirection is configured for a basic rule, all requests on the associated listener are redirected to the target. This is *global* redirection. If redirection is configured for a path-based rule, only requests in a specific site area are redirected. An example is a shopping cart area that's denoted by */cart/**. This is *path-based* redirection.

For more information about redirects, see Application Gateway redirect overview.

Redirection type

Choose the type of redirection required: *Permanent(301)*, *Temporary(307)*, *Found(302)*, or *See other(303)*.

Redirection target

Choose another listener or an external site as the redirection target.

Listener

Choose listener as the redirection target to redirect traffic from one listener to another on the gateway. This setting is required when you want to enable HTTP-to-HTTPS redirection. It redirects traffic from the source listener that checks for incoming HTTP requests to the destination listener that checks for incoming HTTPS requests. You can also choose to include the query string and path from the original request in the request that's forwarded to the redirection target.

For more information about HTTP-to-HTTPS redirection, see:

- HTTP-to-HTTPS redirection by using the Azure portal
- HTTP-to-HTTPS redirection by using PowerShell
- HTTP-to-HTTPS redirection by using the Azure CLI

External site

Choose external site when you want to redirect the traffic on the listener that's associated with this rule to an external site. You can choose to include the query string from the original request in the request that's forwarded to the redirection target. You can't forward the path to the external site that was in the original request.

For more information about redirection, see:

- Redirect traffic to an external site by using PowerShell
- Redirect traffic to an external site by using the CLI

Rewrite HTTP headers and URL

By using rewrite rules, you can add, remove, or update HTTP(S) request and response headers as well as URL path and query string parameters as the request and response packets move between the client and backend pools via the application gateway.

The headers and URL parameters can be set to static values or to other headers and server variables. This helps with important use cases, such as extracting client IP addresses, removing sensitive information about the backend, adding more security, and so on. For more information, see:

- Rewrite HTTP headers and URL overview
- Configure HTTP header rewrite
- Configure URL rewrite

Next steps

- Learn about HTTP settings

Application Gateway HTTP settings configuration

8/22/2022 • 7 minutes to read • Edit Online

The application gateway routes traffic to the back-end servers by using the configuration that you specify here. After you create an HTTP setting, you must associate it with one or more request-routing rules.

Cookie-based affinity

Azure Application Gateway uses gateway-managed cookies for maintaining user sessions. When a user sends the first request to Application Gateway, it sets an affinity cookie in the response with a hash value which contains the session details, so that the subsequent requests carrying the affinity cookie will be routed to the same backend server for maintaining stickiness.

This feature is useful when you want to keep a user session on the same server and when session state is saved locally on the server for a user session. If the application can't handle cookie-based affinity, you can't use this feature. To use it, make sure that the clients support cookies.

> **NOTE**
>
> Some vulnerability scans may flag the Application Gateway affinity cookie because the Secure or HttpOnly flags are not set. These scans do not take into account that the data in the cookie is generated using a one-way hash. The cookie does not contain any user information and is used purely for routing.

The Chromium browser v80 update brought a mandate where HTTP cookies without SameSite attribute have to be treated as SameSite=Lax. In the case of CORS (Cross-Origin Resource Sharing) requests, if the cookie has to be sent in a third-party context, it has to use *SameSite=None; Secure* attributes and it should be sent over HTTPS only. Otherwise, in an HTTP only scenario, the browser doesn't send the cookies in the third-party context. The goal of this update from Chrome is to enhance security and to avoid Cross-Site Request Forgery (CSRF) attacks.

To support this change, starting February 17 2020, Application Gateway (all the SKU types) will inject another cookie called *ApplicationGatewayAffinityCORS* in addition to the existing *ApplicationGatewayAffinity* cookie. The *ApplicationGatewayAffinityCORS* cookie has two more attributes added to it ("*SameSite=None; Secure*") so that sticky sessions are maintained even for cross-origin requests.

Note that the default affinity cookie name is *ApplicationGatewayAffinity* and you can change it. In case you're using a custom affinity cookie name, an additional cookie is added with CORS as suffix. For example, *CustomCookieNameCORS*.

> **NOTE**
>
> If the attribute *SameSite=None* is set, it is mandatory that the cookie also contains the *Secure* flag, and must be sent over HTTPS. If session affinity is required over CORS, you must migrate your workload to HTTPS. Please refer to TLS offload and End-to-End TLS documentation for Application Gateway here – Overview, Configure an application gateway with TLS termination using the Azure portal, Configure end-to-end TLS by using Application Gateway with the portal.

Connection draining

Connection draining helps you gracefully remove back-end pool members during planned service updates. You can apply this setting to all members of a back-end pool by enabling connection draining on the HTTP setting. It

ensures that all deregistering instances of a back-end pool continue to maintain existing connections and serve on-going requests for a configurable timeout and don't receive any new requests or connections. The only exception to this are requests bound for deregistering instances because of gateway-managed session affinity and will continue to be forwarded to the deregistering instances. Connection draining applies to back-end instances that are explicitly removed from the back-end pool.

Protocol

Application Gateway supports both HTTP and HTTPS for routing requests to the back-end servers. If you choose HTTP, traffic to the back-end servers is unencrypted. If unencrypted communication isn't acceptable, choose HTTPS.

This setting combined with HTTPS in the listener supports end-to-end TLS. This allows you to securely transmit sensitive data encrypted to the back end. Each back-end server in the back-end pool that has end-to-end TLS enabled must be configured with a certificate to allow secure communication.

Port

This setting specifies the port where the back-end servers listen to traffic from the application gateway. You can configure ports ranging from 1 to 65535.

Trusted root certificate

If you select HTTPS as the back-end protocol, the Application Gateway requires a trusted root certificate to trust the back-end pool for end-to-end SSL. By default, the **Use well known CA certificate** option is set to **No**. If you plan to use a self-signed certificate, or a certificate signed by an internal Certificate Authority, then you must provide the Application Gateway the matching public certificate that the back-end pool will be using. This certificate must be uploaded directly to the Application Gateway in .CER format.

If you plan to use a certificate on the back-end pool that is signed by a trusted public Certificate Authority, then you can set the **Use well known CA certificate** option to **Yes** and skip uploading a public certificate.

Request timeout

This setting is the number of seconds that the application gateway waits to receive a response from the back-end server.

Override back-end path

This setting lets you configure an optional custom forwarding path to use when the request is forwarded to the back end. Any part of the incoming path that matches the custom path in the **override backend path** field is copied to the forwarded path. The following table shows how this feature works:

- When the HTTP setting is attached to a basic request-routing rule:

ORIGINAL REQUEST	OVERRIDE BACK-END PATH	REQUEST FORWARDED TO BACK END
/home/	/override/	/override/home/
/home/secondhome/	/override/	/override/home/secondhome/

- When the HTTP setting is attached to a path-based request-routing rule:

ORIGINAL REQUEST	PATH RULE	OVERRIDE BACK-END PATH	REQUEST FORWARDED TO BACK END
/pathrule/home/	/pathrule*	/override/	/override/home/
/pathrule/home/secondhome/	/pathrule*	/override/	/override/home/secondhome/
/home/	/pathrule*	/override/	/override/home/
/home/secondhome/	/pathrule*	/override/	/override/home/secondhome/
/pathrule/home/	/pathrule/home*	/override/	/override/
/pathrule/home/secondhome/	/pathrule/home*	/override/	/override/secondhome/
/pathrule/	/pathrule/	/override/	/override/

Use custom probe

This setting associates a custom probe with an HTTP setting. You can associate only one custom probe with an HTTP setting. If you don't explicitly associate a custom probe, the default probe is used to monitor the health of the back end. We recommend that you create a custom probe for greater control over the health monitoring of your back ends.

> **NOTE**
> The custom probe doesn't monitor the health of the back-end pool unless the corresponding HTTP setting is explicitly associated with a listener.

Configuring the host name

Application Gateway allows for the connection established to the backend to use a *different* hostname than the one used by the client to connect to Application Gateway. While this configuration can be useful in some cases, overriding the hostname to be different between the client and application gateway and application gateway to backend target, should be done with care.

In production, it is recommended to keep the hostname used by the client towards the application gateway as the same hostname used by the application gateway to the backend target. This avoids potential issues with absolute URLs, redirect URLs, and host-bound cookies.

Before setting up Application Gateway that deviates from this, please review the implications of such configuration as discussed in more detail in Architecture Center: Preserve the original HTTP host name between a reverse proxy and its backend web application

There are two aspects of an HTTP setting that influence the `Host` HTTP header that is used by Application Gateway to connect to the backend:

- "Pick host name from backend-address"
- "Host name override"

Pick host name from back-end address

This capability dynamically sets the *host* header in the request to the host name of the back-end pool. It uses an IP address or FQDN.

This feature helps when the domain name of the back end is different from the DNS name of the application gateway, and the back end relies on a specific host header to resolve to the correct endpoint.

An example case is multi-tenant services as the back end. An app service is a multi-tenant service that uses a shared space with a single IP address. So, an app service can only be accessed through the hostnames that are configured in the custom domain settings.

By default, the custom domain name is *example.azurewebsites.net*. To access your app service by using an application gateway through a hostname that's not explicitly registered in the app service or through the application gateway's FQDN, you can override the hostname in the original request to the app service's hostname. To do this, enable the **pick host name from backend address** setting.

For a custom domain whose existing custom DNS name is mapped to the app service, the recommended configuration is not to enable the **pick host name from backend address**.

> **NOTE**
>
> This setting is not required for App Service Environment, which is a dedicated deployment.

Host name override

This capability replaces the *host* header in the incoming request on the application gateway with the host name that you specify.

For example, if *www.contoso.com* is specified in the **Host name** setting, the original request * `https://appgw.eastus.cloudapp.azure.com/path1` is changed to * `https://www.contoso.com/path1` when the request is forwarded to the back-end server.

Next steps

- Learn about the back-end pool

Application Gateway high traffic support

8/22/2022 • 10 minutes to read • Edit Online

> **NOTE**
>
> This article describes a few suggested guidelines to help you set up your Application Gateway to handle extra traffic for any high traffic volume that may occur. The alert thresholds are purely suggestions and generic in nature. Users can determine alert thresholds based on their workload and utilization expectations.

You can use Application Gateway with Web Application Firewall (WAF) for a scalable and secure way to manage traffic to your web applications.

It is important that you scale your Application Gateway according to your traffic and with a bit of a buffer so that you are prepared for any traffic surges or spikes and minimizing the impact that it may have in your QoS. The following suggestions help you set up Application Gateway with WAF to handle extra traffic.

Please check the metrics documentation for the complete list of metrics offered by Application Gateway. See visualize metrics in the Azure portal and the Azure monitor documentation on how to set alerts for metrics.

Scaling for Application Gateway v1 SKU (Standard/WAF SKU)

Set your instance count based on your peak CPU usage

If you are using a v1 SKU gateway, you'll have the ability to set your Application Gateway up to 32 instances for scaling. Check your Application Gateway's CPU utilization in the past one month for any spikes above 80%, it is available as a metric for you to monitor. It is recommended that you set your instance count according to your peak usage and with a 10% to 20% additional buffer to account for any traffic spikes.

Use the v2 SKU over v1 for its autoscaling capabilities and performance benefits

The v2 SKU offers autoscaling to ensure that your Application Gateway can scale up as traffic increases. It also offers other significant performance benefits, such as 5x better TLS offload performance, quicker deployment and update times, zone redundancy, and more when compared to v1. For more information, see our v2

documentation and see our v1 to v2 migration documentation to learn how to migrate your existing v1 SKU gateways to v2 SKU.

Autoscaling for Application Gateway v2 SKU (Standard_v2/WAF_v2 SKU)

Set maximum instance count to the maximum possible (125)

For Application Gateway v2 SKU, setting the maximum instance count to the maximum possible value of 125 allows the Application Gateway to scale out as needed. This allows it to handle the possible increase in traffic to your applications. You will only be charged for the Capacity Units (CUs) you use.

Make sure to check your subnet size and available IP address count in your subnet and set your maximum instance count based on that. If your subnet doesn't have enough space to accommodate, you will have to re-create your gateway in the same or different subnet which has enough capacity.

Set your minimum instance count based on your average Compute Unit usage

For Application Gateway v2 SKU, autoscaling takes six to seven minutes to scale out and provision additional set of instances ready to take traffic. Until then, if there are short spikes in traffic, your existing gateway instances might get under stress and this may cause unexpected latency or loss of traffic.

It is recommended that you set your minimum instance count to an optimal level. For example, if you require 50 instances to handle the traffic at peak load, then setting the minimum 25 to 30 is a good idea rather than at <10 so that even when there are short bursts of traffic, Application Gateway would be able to handle it and give enough time for autoscaling to respond and take effect.

Check your Compute Unit metric for the past one month. Compute unit metric is a representation of your gateway's CPU utilization and based on your peak usage divided by 10, you can set the minimum number of instances required. Note that 1 application gateway instance can handle a minimum of 10 compute units

Manual scaling for Application Gateway v2 SKU (Standard_v2/WAF_v2)

Set your instance count based on your peak Compute Unit usage

Unlike autoscaling, in manual scaling, you must manually set the number of instances of your application gateway based on the traffic requirements. It is recommended that you set your instance count according to your peak usage and with a 10% to 20% additional buffer to account for any traffic spikes. For example, if your traffic requires 50 instances at peak, provision 55 to 60 instances to handle unexpected traffic spikes that may occur.

Check your Compute Unit metric for the past one month. Compute unit metric is a representation of your gateway's CPU utilization and based on your peak usage divided by 10, you can set the number of instances required, since 1 application gateway instance can handle a minimum of 10 compute units

Monitoring and alerting

To get notified of any traffic or utilization anomalies, you can set up alerts on certain metrics. See metrics documentation for the complete list of metrics offered by Application Gateway. See visualize metrics in the Azure portal and the Azure monitor documentation on how to set alerts for metrics.

To configure alerts using ARM templates, see Configure Azure Monitor alerts for Application Gateway.

Alerts for Application Gateway v1 SKU (Standard/WAF)

Alert if average CPU utilization crosses 80%

Under normal conditions, CPU usage should not regularly exceed 90%, as this may cause latency in the websites hosted behind the Application Gateway and disrupt the client experience. You can indirectly control or improve CPU utilization by modifying the configuration of the Application Gateway by increasing the instance count or by moving to a larger SKU size or doing both. Set an alert if the CPU utilization metric goes above 80% average.

Alert if Unhealthy host count crosses threshold

This metric indicates number of backend servers that application gateway is unable to probe successfully. This will catch issues where Application gateway instances are unable to connect to the backend. Alert if this number goes above 20% of backend capacity. E.g. if currently you have 30 backend servers in their backend pool, set an alert if the unhealthy host count goes above 6.

Alert if Response status (4xx, 5xx) crosses threshold

Create alert when Application Gateway response status is 4xx or 5xx. There could be occasional 4xx or 5xx response seen due to transient issues. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

Alert if Failed requests crosses threshold

Create alert when Failed requests metric crosses threshold. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

Example: Setting up an alert for more than 100 failed requests in the last 5 minutes

This example shows you how to use the Azure portal to set up an alert when the failed request count in the last 5 minutes is more than 100.

1. Navigate to your **Application Gateway**.
2. On the left panel, select **Metrics** under the **Monitoring** tab.
3. Add a metric for **Failed requests**.
4. Click on **New alert rule** and define your condition and actions
5. Click on **Create alert rule** to create and enable the alert

Alerts for Application Gateway v2 SKU (Standard_v2/WAF_v2)

Alert if Compute Unit utilization crosses 75% of average usage

Compute unit is the measure of compute utilization of your Application Gateway. Check your average compute unit usage in the last one month and set alert if it crosses 75% of it. For example, if your average usage is 10 compute units, set an alert on 7.5 CUs. This alerts you if usage is increasing and gives you time to respond. You can raise the minimum if you think this traffic will be sustained to alert you that traffic may be increasing. Follow the scaling suggestions above to scale out as necessary.

Example: Setting up an alert on 75% of average CU usage

This example shows you how to use the Azure portal to set up an alert when 75% of average CU usage is reached.

1. Navigate to your **Application Gateway**.
2. On the left panel, select **Metrics** under the **Monitoring** tab.
3. Add a metric for **Average Current Compute Units**.

4. If you've set your minimum instance count to be your average CU usage, go ahead and set an alert when 75% of your minimum instances are in use. For example, if your average usage is 10 CUs, set an alert on 7.5 CUs. This alerts you if usage is increasing and gives you time to respond. You can raise the minimum if you think this traffic will be sustained to alert you that traffic may be increasing.

> **NOTE**
>
> You can set the alert to occur at a lower or higher CU utilization percentage depending on how sensitive you want to be to potential traffic spikes.

Alert if Capacity Unit utilization crosses 75% of peak usage

Capacity units represent overall gateway utilization in terms of throughput, compute, and connection count. Check your maximum capacity unit usage in the last one month and set alert if it crosses 75% of it. For example, if your maximum usage is 100 capacity units, set an alert on 75 CUs. Follow the above two suggestions to scale out, as necessary.

Alert if Unhealthy host count crosses threshold

This metric indicates number of backend servers that application gateway is unable to probe successfully. This will catch issues where Application gateway instances are unable to connect to the backend. Alert if this number goes above 20% of backend capacity. E.g. if currently you have 30 backend servers in their backend pool, set an alert if the unhealthy host count goes above 6.

Alert if Response status (4xx, 5xx) crosses threshold

Create alert when Application Gateway response status is 4xx or 5xx. There could be occasional 4xx or 5xx response seen due to transient issues. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

Alert if Failed requests crosses threshold

Create alert when Failed requests metric crosses threshold. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

Alert if Backend last byte response time crosses threshold

This metric indicates the time interval between start of establishing a connection to backend server and receiving the last byte of the response body. Create an alert if the backend response latency is more that certain threshold from usual. For example, set this to be alerted when backend response latency increases by more than 30% from the usual value.

Alert if Application Gateway total time crosses threshold

This is the interval from the time when Application Gateway receives the first byte of the HTTP request to the time when the last response byte has been sent to the client. Should create an alert if the backend response latency is more that certain threshold from usual. For example, they can set this to be alerted when total time latency increases by more than 30% from the usual value.

Set up WAF with geo filtering and bot protection to stop attacks

If you want an extra layer of security in front of your application, use the Application Gateway WAF_v2 SKU for WAF capabilities. You can configure the v2 SKU to only allow access to your applications from a given country/region or countries/regions. You set up a WAF custom rule to explicitly allow or block traffic based on the geo location. For more information, see geo filtering custom rules and how to configure custom rules on Application Gateway WAF_v2 SKU through PowerShell.

Enable bot protection to block known bad bots. This should reduce the amount of traffic getting to your application. For more information, see bot protection with set up instructions.

Turn on diagnostics on Application Gateway and WAF

Diagnostic logs allow you to view firewall logs, performance logs, and access logs. You can use these logs in Azure to manage and troubleshoot Application Gateways. For more information, see our diagnostics documentation.

Set up an TLS policy for extra security

Ensure you're using the latest TLS policy version (AppGwSslPolicy20220101) or higher. These support minimum TLS version 1.2 with stronger ciphers. For more information, see configuring TLS policy versions and cipher suites via PowerShell.

Scaling Application Gateway v2 and WAF v2

8/22/2022 • 2 minutes to read • Edit Online

Application Gateway and WAF can be configured to scale in two modes:

- **Autoscaling** - With autoscaling enabled, the Application Gateway and WAF v2 SKUs scale out or in based on application traffic requirements. This mode offers better elasticity to your application and eliminates the need to guess the application gateway size or instance count. This mode also allows you to save cost by not requiring the gateway to run at peak-provisioned capacity for expected maximum traffic load. You must specify a minimum and optionally maximum instance count. Minimum capacity ensures that Application Gateway and WAF v2 don't fall below the minimum instance count specified, even without traffic. Each instance is roughly equivalent to 10 more reserved Capacity Units. Zero signifies no reserved capacity and is purely autoscaling in nature. You can also optionally specify a maximum instance count, which ensures that the Application Gateway doesn't scale beyond the specified number of instances. You'll only be billed for the amount of traffic served by the Gateway. The instance counts can range from 0 to 125. The default value for maximum instance count is 20 if not specified.
- **Manual** - You can also choose Manual mode where the gateway won't autoscale. In this mode, if there's more traffic than what Application Gateway or WAF can handle, it could result in traffic loss. With manual mode, specifying instance count is mandatory. Instance count can vary from 1 to 125 instances.

Autoscaling and High Availability

Azure Application Gateways are always deployed in a highly available fashion. The service is made out of multiple instances that are created as configured (if autoscaling is disabled) or required by the application load (if autoscaling is enabled). Note that from the user's perspective you don't necessarily have visibility into the individual instances, but just into the Application Gateway service as a whole. If a certain instance has a problem and stops being functional, Azure Application Gateway will transparently create a new instance.

Even if you configure autoscaling with zero minimum instances the service will still be highly available, which is always included with the fixed price.

However, creating a new instance can take some time (around six or seven minutes). If you don't want to have this downtime, you can configure a minimum instance count of two, ideally with Availability Zone support. This way you'll have at least two instances in your Azure Application Gateway under normal circumstances. So if one of them had a problem the other will try to handle the traffic while a new instance is being created. An Azure Application Gateway instance can support around 10 Capacity Units, so depending on how much traffic you typically have you might want to configure your minimum instance autoscaling setting to a value higher than two.

For scale-in events, Application Gateway will drain existing connections for 5 minutes on the instance that is subject for removal. After 5 minutes, existing connections will be closed and the instance removed. Any new connections during or after the 5 minute scale-in time will be established to other existing instances on the same gateway.

Next steps

- Learn more about Application Gateway v2
- Create an autoscaling, zone redundant application gateway with a reserved virtual IP address using Azure PowerShell

Application Gateway multiple site hosting

8/22/2022 • 7 minutes to read • Edit Online

Multiple site hosting enables you to configure more than one web application on the same port of application gateways using public-facing listeners. It allows you to configure a more efficient topology for your deployments by adding up to 100+ websites to one application gateway. Each website can be directed to its own backend pool. For example, three domains, contoso.com, fabrikam.com, and adatum.com, point to the IP address of the application gateway. You'd create three multi-site listeners and configure each listener for the respective port and protocol setting.

You can also define wildcard host names in a multi-site listener and up to 5 host names per listener. To learn more, see wildcard host names in listener.

> **IMPORTANT**
>
> Rules are processed in the order they are listed in the portal for the v1 SKU. For v2 SKU use rule priority to specify the processing order. It is highly recommended to configure multi-site listeners first prior to configuring a basic listener. This will ensure that traffic gets routed to the right back end. If a basic listener is listed first and matches an incoming request, it gets processed by that listener.

Requests for `http://contoso.com` are routed to ContosoServerPool, and `http://fabrikam.com` are routed to FabrikamServerPool.

Similarly, you can host multiple subdomains of the same parent domain on the same application gateway deployment. For example, you can host `http://blog.contoso.com` and `http://app.contoso.com` on a single application gateway deployment.

Request Routing rules evaluation order

While using multi-site listeners, to ensure that the client traffic is routed to the accurate backend, it's important to have the request routing rules be present in the correct order. For example, if you have 2 listeners with associated Host name as `*.contoso.com` and `shop.contoso.com` respectively, the listener with the `shop.contoso.com` Host name would have to be processed before the listener with `*.contoso.com`. If the listener with `*.contoso.com` is processed first, then no client traffic would be received by the more specific `shop.contoso.com` listener.

This ordering can be established by providing a 'Priority' field value to the request routing rules associated with

the listeners. You can specify an integer value from 1 to 20000 with 1 being the highest priority and 20000 being the lowest priority. In case the incoming client traffic matches with multiple listeners, the request routing rule with highest priority will be used for serving the request. Each request routing rule needs to have a unique priority value.

The priority field only impacts the order of evaluation of a request routing rule, this wont change the order of evaluation of path based rules within a `PathBasedRouting` request routing rule.

> **NOTE**
> If you wish to use rule priority, you will have to specify rule priority field values for all the existing request routing rules. Once the rule priority field is in use, any new routing rule that is created would also need to have a rule priority field value as part of its config. Starting with API version 2021-08-01 rule priority field would be a mandatory field as part of the request routing rules. From this API version, rule priority field values would be auto-populated for existing request routing rules based on current ordering of evaluation as part of the first PUT call. Any future updates to request routing rules would need to have the rule priority field provided as part of the configuration.

> **IMPORTANT**
> Rule priority field values for existing request routing rules based on current order would be automatically populated if any configuration updates are applied using API version 2021-08-01 and above, portal, Azure PowerShell and Azure CLI. Any future updates to request routing rules would need to have the rule priority field provided as part of the configuration.

Wildcard host names in listener

Application Gateway allows host-based routing using multi-site HTTP(S) listener. Now, you can use wildcard characters like asterisk (*) and question mark (?) in the host name, and up to 5 host names per multi-site HTTP(S) listener. For example, `*.contoso.com`.

Using a wildcard character in the host name, you can match multiple host names in a single listener. For example, `*.contoso.com` can match with `ecom.contoso.com`, `b2b.contoso.com` and `customer1.b2b.contoso.com` and so on. Using an array of host names, you can configure more than one host name for a listener, to route requests to a backend pool. For example, a listener can contain `contoso.com, fabrikam.com` which will accept requests for both the host names.

> **NOTE**
> This feature is available only for Standard_v2 and WAF_v2 SKU of Application Gateway.

In Azure PowerShell, you must use `-HostNames` instead of `-HostName`. With HostNames, you can mention up to 5 host names as comma-separated values and use wildcard characters. For example, `-HostNames "*.contoso.com","*.fabrikam.com"`

In Azure CLI, you must use `--host-names` instead of `--host-name`. With host-names, you can mention up to 5

host names as comma-separated values and use wildcard characters. For example,

```
--host-names "*.contoso.com,*.fabrikam.com"
```

In the Azure portal, under the multi-site listener, you must chose the **Multiple/Wildcard** host type to mention up to five host names with allowed wildcard characters.

Allowed characters in the host names field

- `(A-Z,a-z,0-9)` - alphanumeric characters
- `-` - hyphen or minus
- `.` - period as a delimiter
- `*` - can match with multiple characters in the allowed range
- `?` - can match with a single character in the allowed range

Conditions for using wildcard characters and multiple host names in a listener

- You can only mention up to 5 host names in a single listener
- Asterisk `*` can be mentioned only once in a component of a domain style name or host name. For example, component1*.component2*.component3. `(*.contoso-*.com)` is valid.
- There can only be up to two asterisks `*` in a host name. For example, `*.contoso.*` is valid and `*.contoso.*.*.com` is invalid.
- There can only be a maximum of 4 wildcard characters in a host name. For example, `????.contoso.com`, `w??.contoso*.edu.*` are valid, but `????.contoso.*` is invalid.
- Using asterisk `*` and question mark `?` together in a component of a host name (`*?` or `?*` or `**`) is invalid. For example, `*?.contoso.com` and `**.contoso.com` are invalid.

Considerations and limitations of using wildcard or multiple host names in a listener

- SSL termination and End-to-End SSL requires you to configure the protocol as HTTPS and upload a certificate to be used in the listener configuration. If it's a multi-site listener, you can input the host name as well, usually this is the CN of the SSL certificate. When you are specifying multiple host names in the listener or use wildcard characters, you must consider the following:
  - If it's a wildcard hostname like *.contoso.com, you must upload a wildcard certificate with CN like *.contoso.com
  - If multiple host names are mentioned in the same listener, you must upload a SAN certificate (Subject Alternative Names) with the CNs matching the host names mentioned.
- You cannot use a regular expression to mention the host name. You can only use wildcard characters like asterisk (*) and question mark (?) to form the host name pattern.
- For backend health check, you cannot associate multiple custom probes per HTTP settings. Instead, you can probe one of the websites at the backend or use "127.0.0.1" to probe the localhost of the backend server. However, when you are using wildcard or multiple host names in a listener, the requests for all the specified domain patterns will be routed to the backend pool depending on the rule type (basic or path-based).
- The properties "hostname" takes one string as input, where you can mention only one non-wildcard domain name and "hostnames" takes an array of strings as input, where you can mention up to 5 wildcard domain names. But both the properties cannot be used at once.

See create multi-site using Azure PowerShell or using Azure CLI for the step-by-step guide on how to configure wildcard host names in a multi-site listener.

Host headers and Server Name Indication (SNI)

There are three common mechanisms for enabling multiple site hosting on the same infrastructure.

1. Host multiple web applications each on a unique IP address.
2. Use host name to host multiple web applications on the same IP address.
3. Use different ports to host multiple web applications on the same IP address.

Currently Application Gateway supports a single public IP address where it listens for traffic. So multiple applications, each with its own IP address is currently not supported.

Application Gateway supports multiple applications each listening on different ports, but this scenario requires the applications to accept traffic on non-standard ports.

Application Gateway relies on HTTP 1.1 host headers to host more than one website on the same public IP address and port. The sites hosted on application gateway can also support TLS offload with Server Name Indication (SNI) TLS extension. This scenario means that the client browser and backend web farm must support HTTP/1.1 and TLS extension as defined in RFC 6066.

Next steps

Learn how to configure multiple site hosting in Application Gateway

- Using Azure portal
- Using Azure PowerShell
- Using Azure CLI

You can visit Resource Manager template using multiple site hosting for an end to end template-based deployment.

URL Path Based Routing overview

8/22/2022 • 2 minutes to read • Edit Online

URL Path Based Routing allows you to route traffic to back-end server pools based on URL Paths of the request.

One of the scenarios is to route requests for different content types to different backend server pools.

In the following example, Application Gateway is serving traffic for contoso.com from three back-end server pools for example: VideoServerPool, ImageServerPool, and DefaultServerPool.

Requests for http://contoso.com/video/* are routed to VideoServerPool, and http://contoso.com/images/* are routed to ImageServerPool. DefaultServerPool is selected if none of the path patterns match.

> **IMPORTANT**
>
> For both the v1 and v2 SKUs, rules are processed in the order they are listed in the portal. The best practice when you create path rules is to have the least specific path (the ones with wildcards) at the end. If wildcards are on the top, then they take priority even if there's a more specific match in subsequent path rules.
>
> If a basic listener is listed first and matches an incoming request, it gets processed by that listener. However, it's highly recommended to configure multi-site listeners first prior to configuring a basic listener. This ensures that traffic gets routed to the right back end.

UrlPathMap configuration element

The urlPathMap element is used to specify Path patterns to back-end server pool mappings. The following code example is the snippet of urlPathMap element from template file.

```
"urlPathMaps": [{
    "name": "{urlpathMapName}",
    "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/urlPathMaps/{urlpath
MapName}",
    "properties": {
        "defaultBackendAddressPool": {
            "id": "/subscriptions/
{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/backendAddressPools/{poolName1}"
        },
        "defaultBackendHttpSettings": {
            "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/backendHttpSettingsC
ollection/{settingname1}"
        },
        "pathRules": [{
            "name": "{pathRuleName}",
            "properties": {
                "paths": [
                    "{pathPattern}"
                ],
                "backendAddressPool": {
                    "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/backendAddressPools/
{poolName2}"
                },
                "backendHttpsettings": {
                    "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/backendHttpSettingsC
ollection/{settingName2}"
                }
            }
        }]
    }
}]
```

PathPattern

PathPattern is a list of path patterns to match. Each path must start with / and may use * as a wildcard character. The string fed to the path matcher doesn't include any text after the first `?` or `#`, and those chars aren't allowed here. Otherwise, any characters allowed in a URL are allowed in PathPattern.

Path rules are case insensitive.

PATH PATTERN	IS SUPPORTED?
`/images/*`	yes
`/images*`	yes
`/images/*.jpg`	no
`/*.jpg`	no
`/Repos/*/Comments/*`	no
`/CurrentUser/Comments/*`	yes

Examples

Path-based rule processing when wildcard (*) is used:

Example 1:

`/master-dev* to contoso.com`

`/master-dev/api-core/ to fabrikam.com`

`/master-dev/* to microsoft.com`

Because the wildcard path `/master-dev*` is present above more granular paths, all client requests containing `/master-dev` are routed to contoso.com, including the specific `/master-dev/api-core/`. To ensure that the client requests are routed to the appropriate paths, it's critical to have the granular paths above wildcard paths.

Example 2:

`/ (default) to contoso.com`

`/master-dev/api-core/ to fabrikam.com`

`/master-dev/api to bing.com`

`/master-dev/* to microsoft.com`

All client requests with the path pattern `/master-dev/*` are processed in the order as listed. If there's no match within the path rules, the request is routed to the default target.

For more information, see Resource Manager template using URL-based routing.

PathBasedRouting rule

RequestRoutingRule of type PathBasedRouting is used to bind a listener to a urlPathMap. All requests that are received for this listener are routed based on policy specified in urlPathMap. Snippet of PathBasedRouting rule:

```
"requestRoutingRules": [
    {

"name": "{ruleName}",
"id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/requestRoutingRules/
{ruleName}",
"properties": {
    "ruleType": "PathBasedRouting",
    "httpListener": {
        "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/httpListeners/<liste
nerName>"
    },
    "urlPathMap": {
        "id":
"/subscriptions/{subscriptionId}/../microsoft.network/applicationGateways/{gatewayName}/urlPathMaps/{urlpath
MapName}"
    }

}
    }
]
```

Next steps

After learning about URL-based content routing, go to create an application gateway using URL-based routing to create an application gateway with URL routing rules.

Application Gateway redirect overview

8/22/2022 • 2 minutes to read • Edit Online

You can use application gateway to redirect traffic. It has a generic redirection mechanism which allows for redirecting traffic received at one listener to another listener or to an external site. This simplifies application configuration, optimizes the resource usage, and supports new redirection scenarios including global and path-based redirection.

A common redirection scenario for many web applications is to support automatic HTTP to HTTPS redirection to ensure all communication between application and its users occurs over an encrypted path. In the past, customers have used techniques such as creating a dedicated backend pool whose sole purpose is to redirect requests it receives on HTTP to HTTPS. With redirection support in Application Gateway, you can accomplish this simply by adding a new redirect configuration to a routing rule, and specifying another listener with HTTPS protocol as the target listener.

Redirection types

A redirect type sets the response status code for the clients to understand the purpose of the redirect. The following types of redirection are supported:

- 301 (Moved permanently): Indicates that the target resource has been assigned a new permanent URI. Any future references to this resource will use one of the enclosed URIs. Use 301 status code for HTTP to HTTPS redirection.
- 302 (Found): Indicates that the target resource is temporarily under a different URI. Since the redirection can change on occasion, the client should continue to use the effective request URI for future requests.
- 303 (See Other): Indicates that the target resource is redirecting the user agent to a different resource, as indicated by a URI in the Location header field.
- 307 (Temporary redirect): Indicates that the target resource is temporarily under a different URI. The user agent MUST NOT change the request method if it does an automatic redirection to that URI. Since the redirection can change over time, the client ought to continue using the original effective request URI for future requests.

Redirection capabilities

- **Listener redirection**

  Redirects from one listener to another listener. Listener redirection is commonly used to enable HTTP to HTTPS redirection.

  When configuring redirects with a multi-site target listener, it is required that all the host names (with or without wildcard characters) are defined as part of the source listener are also part of the destination listener. This ensures that no traffic is dropped due to missing host names on the destination listener while configuring HTTP to HTTPS redirection.

- **Path-based redirection**

  This type of redirection enables redirection only on a specific site area, for example, redirecting HTTP to HTTPS requests for a shopping cart area denoted by /cart/*.

- **Redirect to external site**

With this change, customers need to create a new redirect configuration object, which specifies the target listener or external site to which redirection is desired. The configuration element also supports options to enable appending the URI path and query string to the redirected URL. You can also choose the type of redirection. Once created, this redirect configuration is attached to the source listener via a new rule. When using a basic rule, the redirect configuration is associated with a source listener and is a global redirect. When a path-based rule is used, the redirect configuration is defined on the URL path map. So it only applies to the specific path area of a site.

Next steps

Configure URL redirection on an application gateway

Rewrite HTTP headers and URL with Application Gateway

8/22/2022 • 17 minutes to read • Edit Online

Application Gateway allows you to rewrite selected content of requests and responses. With this feature, you can translate URLs, query string parameters as well as modify request and response headers. It also allows you to add conditions to ensure that the URL or the specified headers are rewritten only when certain conditions are met. These conditions are based on the request and response information.

> **NOTE**
>
> HTTP header and URL rewrite features are only available for the Application Gateway v2 SKU

Rewrite types supported

Request and response headers

HTTP headers allow a client and server to pass additional information with a request or response. By rewriting these headers, you can accomplish important tasks, such as adding security-related header fields like HSTS/ X-XSS-Protection, removing response header fields that might reveal sensitive information, and removing port information from X-Forwarded-For headers.

Application Gateway allows you to add, remove, or update HTTP request and response headers while the request and response packets move between the client and back-end pools.

To learn how to rewrite request and response headers with Application Gateway using Azure portal, see here.

Supported headers

You can rewrite all headers in requests and responses, except for the Connection, and Upgrade headers. You can also use the application gateway to create custom headers and add them to the requests and responses being routed through it.

URL path and query string

With URL rewrite capability in Application Gateway, you can:

- Rewrite the host name, path and query string of the request URL
- Choose to rewrite the URL of all requests on a listener or only those requests which match one or more of the conditions you set. These conditions are based on the request properties (request header and server variables).
- Choose to route the request (select the backend pool) based on either the original URL or the rewritten URL

To learn how to rewrite URL with Application Gateway using Azure portal, see here.

Rewrite actions

You use rewrite actions to specify the URL, request headers or response headers that you want to rewrite and the new value to which you intend to rewrite them to. The value of a URL or a new or existing header can be set to these types of values:

- Text
- Request header. To specify a request header, you need to use the syntax {http_req_*headerName*}
- Response header. To specify a response header, you need to use the syntax {http_resp_*headerName*}
- Server variable. To specify a server variable, you need to use the syntax {var_*serverVariable*}. See the list of supported server variables
- A combination of text, a request header, a response header, and a server variable.

Rewrite Conditions

You can use rewrite conditions, an optional configuration, to evaluate the content of HTTP(S) requests and responses and perform a rewrite only when one or more conditions are met. The application gateway uses these types of variables to evaluate the content of requests and responses:

- HTTP headers in the request
- HTTP headers in the response
- Application Gateway server variables

You can use a condition to evaluate whether a specified variable is present, whether a specified variable matches a specific value, or whether a specified variable matches a specific pattern.

Pattern Matching

Application Gateway uses regular expressions for pattern matching in the condition. You can use the Perl Compatible Regular Expressions (PCRE) library to set up regular expression pattern matching in the conditions. To learn about regular expression syntax, see the Perl regular expressions main page.

Capturing

To capture a substring for later use, put parentheses around the subpattern that matches it in the condition regex definition. The first pair of parentheses stores its substring in 1, the second pair in 2, and so on. You may use as many parentheses as you like; Perl just keeps defining more numbered variables for you to represent these captured strings. Some examples from ref:

- (\d)(\d) # Match two digits, capturing them into groups 1 and 2
- (\d+) # Match one or more digits, capturing them all into group 1
- (\d)+ # Match a digit one or more times, capturing the last into group 1

> **NOTE**
> Use of / to prefix and suffix the pattern should not be specified in the pattern to match value. For example, (\d)(\d) will match two digits. /(\d)(\d)/ will not match two digits.

Once captured, you can reference them in the action set using the following format:

- For a request header capture, you must use {http_req_headerName_groupNumber}. For example, {http_req_User-Agent_1} or {http_req_User-Agent_2}
- For a response header capture, you must use {http_resp_headerName_groupNumber}. For example,

{http_resp_Location_1} or {http_resp_Location_2}

- For a server variable, you must use {var_serverVariableName_groupNumber}. For example, {var_uri_path_1} or {var_uri_path_2}

> **NOTE**
>
> The case of the condition variable needs to match case of the capture variable. For example, if my condition variable is User-Agent, my capture variable must be in the case of User-Agent (i.e. {http_req_User-Agent_2}). If my condition variable is defined as user-agent, my capture variable must be in the case of user-agent (i.e. {http_req_user-agent_2}).

If you want to use the whole value, you should not mention the number. Simply use the format {http_req_headerName}, etc. without the groupNumber.

Server variables

Application Gateway uses server variables to store useful information about the server, the connection with the client, and the current request on the connection. Examples of information stored include the client's IP address and the web browser type. Server variables change dynamically, for example, when a new page loads or when a form is posted. You can use these variables to evaluate rewrite conditions and rewrite headers. In order to use the value of server variables to rewrite headers, you will need to specify these variables in the syntax {var_*serverVariableName*}

Application gateway supports the following server variables:

VARIABLE NAME	DESCRIPTION
add_x_forwarded_for_proxy	The X-Forwarded-For client request header field with the `client_ip` variable (see explanation later in this table) appended to it in the format IP1, IP2, IP3, and so on. If the X-Forwarded-For field isn't in the client request header, the `add_x_forwarded_for_proxy` variable is equal to the `$client_ip` variable. This variable is particularly useful when you want to rewrite the X-Forwarded-For header set by Application Gateway so that the header contains only the IP address without the port information.
ciphers_supported	A list of the ciphers supported by the client.
ciphers_used	The string of ciphers used for an established TLS connection.
client_ip	The IP address of the client from which the application gateway received the request. If there's a reverse proxy before the application gateway and the originating client, `client_ip` will return the IP address of the reverse proxy.
client_port	The client port.
client_tcp_rtt	Information about the client TCP connection. Available on systems that support the TCP_INFO socket option.
client_user	When HTTP authentication is used, the user name supplied for authentication.
host	In this order of precedence: the host name from the request line, the host name from the Host request header field, or the server name matching a request. Example: In the request `http://contoso.com:8080/article.aspx?id=123&title=fabrikam`, host value will be is `contoso.com`

VARIABLE NAME	DESCRIPTION
cookie_*name*	The *name* cookie.
http_method	The method used to make the URL request. For example, GET or POST.
http_status	The session status. For example, 200, 400, or 403.
http_version	The request protocol. Usually HTTP/1.0, HTTP/1.1, or HTTP/2.0.
query_string	The list of variable/value pairs that follows the "?" in the requested URL. Example: In the request `http://contoso.com:8080/article.aspx?id=123&title=fabrikam`, query_string value will be `id=123&title=fabrikam`
received_bytes	The length of the request (including the request line, header, and request body).
request_query	The arguments in the request line.
request_scheme	The request scheme: http or https.
request_uri	The full original request URI (with arguments). Example: in the request `http://contoso.com:8080/article.aspx?id=123&title=fabrikam*`, request_uri value will be `/article.aspx?id=123&title=fabrikam`
sent_bytes	The number of bytes sent to a client.
server_port	The port of the server that accepted a request.
ssl_connection_protocol	The protocol of an established TLS connection.
ssl_enabled	"On" if the connection operates in TLS mode. Otherwise, an empty string.
uri_path	Identifies the specific resource in the host that the web client wants to access. This is the part of the request URI without the arguments. Example: In the request `http://contoso.com:8080/article.aspx?id=123&title=fabrikam`, uri_path value will be `/article.aspx`

Mutual authentication server variables

Application Gateway supports the following server variables for mutual authentication scenarios. Use these server variables the same way as above with the other server variables.

VARIABLE NAME	DESCRIPTION
client_certificate	The client certificate in PEM format for an established SSL connection.
client_certificate_end_date	The end date of the client certificate.

VARIABLE NAME	DESCRIPTION
client_certificate_fingerprint	The SHA1 fingerprint of the client certificate for an established SSL connection.
client_certificate_issuer	The "issuer DN" string of the client certificate for an established SSL connection.
client_certificate_serial	The serial number of the client certificate for an established SSL connection.
client_certificate_start_date	The start date of the client certificate.
client_certificate_subject	The "subject DN" string of the client certificate for an established SSL connection.
client_certificate_verification	The result of the client certificate verification: *SUCCESS*, *FAILED:<reason>*, or *NONE* if a certificate was not present.

Rewrite configuration

To configure a rewrite rule, you need to create a rewrite rule set and add the rewrite rule configuration in it.

A rewrite rule set contains:

- **Request routing rule association:** The rewrite configuration is associated to the source listener via the routing rule. When you use a basic routing rule, the rewrite configuration is associated with a source listener and is a global header rewrite. When you use a path-based routing rule, the rewrite configuration is defined on the URL path map. In that case, it applies only to the specific path area of a site. You can create multiple rewrite sets and apply each rewrite set to multiple listeners. But you can apply only one rewrite set to a specific listener.
- **Rewrite Condition**: It is an optional configuration. Rewrite conditions evaluate the content of the HTTP(S) requests and responses. The rewrite action will occur if the HTTP(S) request or response matches the rewrite condition. If you associate more than one condition with an action, the action occurs only when all the conditions are met. In other words, the operation is a logical AND operation.
- **Rewrite type**: There are 3 types of rewrites available:
  - Rewriting request headers
  - Rewriting response headers
  - Rewriting URL components
    - **URL path**: The value to which the path is to be rewritten to.
    - **URL Query String**: The value to which the query string is to be rewritten to.
    - **Re-evaluate path map**: Used to determine whether the URL path map is to be re-evaluated or not. If kept unchecked, the original URL path will be used to match the path-pattern in the URL path map. If set to true, the URL path map will be re-evaluated to check the match with the rewritten path. Enabling this switch helps in routing the request to a different backend pool post rewrite.

Rewrite configuration common pitfalls

- Enabling 'Re-evaluate path map' is not allowed for basic request routing rules. This is to prevent infinite evaluation loop for a basic routing rule.
- There needs to be at least 1 conditional rewrite rule or 1 rewrite rule which does not have 'Re-evaluate path map' enabled for path-based routing rules to prevent infinite evaluation loop for a path-based routing rule.

- Incoming requests would be terminated with a 500 error code in case a loop is created dynamically based on client inputs. The Application Gateway will continue to serve other requests without any degradation in such a scenario.

Using URL rewrite or Host header rewrite with Web Application Firewall (WAF_v2 SKU)

When you configure URL rewrite or host header rewrite, the WAF evaluation will happen after the modification to the request header or URL parameters (post-rewrite). And when you remove the URL rewrite or host header rewrite configuration on your Application Gateway, the WAF evaluation will be done before the header rewrite (pre-rewrite). This order ensures that WAF rules are applied to the final request that would be received by your backend pool.

For example, say you have the following header rewrite rule for the header `"Accept" : "text/html"` - if the value of header `"Accept"` is equal to `"text/html"`, then rewrite the value to `"image/png"`.

Here, with only header rewrite configured, the WAF evaluation will be done on `"Accept" : "text/html"`. But when you configure URL rewrite or host header rewrite, then the WAF evaluation will be done on `"Accept" : "image/png"`.

Common scenarios for header rewrite

Remove port information from the X-Forwarded-For header

Application Gateway inserts an X-Forwarded-For header into all requests before it forwards the requests to the backend. This header is a comma-separated list of IP ports. There might be scenarios in which the back-end servers only need the headers to contain IP addresses. You can use header rewrite to remove the port information from the X-Forwarded-For header. One way to do this is to set the header to the add_x_forwarded_for_proxy server variable. Alternatively, you can also use the variable client_ip:

Modify a redirection URL

Modification of a redirect URL can be useful under certain circumstances. For example: clients were originally redirected to a path like "/blog" but now should be sent to "/updates" due to a change in content structure.

> **WARNING**
>
> The need to modify a redirection URL sometimes comes up in the context of a configuration whereby Application Gateway is configured to override the hostname towards the backend. The hostname as seen by the backend is in that case different from the hostname as seen by the browser. In this situation, the redirect would not use the correct hostname. This configuration is not recommended.
>
> The limitations and implications of such a configuration are described in Preserve the original HTTP host name between a reverse proxy and its back-end web application. The recommended setup for App Service is to follow the instructions for **"Custom Domain (recommended)"** in Configure App Service with Application Gateway. Rewriting the location header on the response as described in the below example should be considered a workaround and does not address the root cause.

When the app service sends a redirection response, it uses the same hostname in the location header of its response as the one in the request it receives from the application gateway. So the client will make the request directly to `contoso.azurewebsites.net/path2` instead of going through the application gateway (`contoso.com/path2`). Bypassing the application gateway isn't desirable.

You can resolve this issue by setting the hostname in the location header to the application gateway's domain name.

Here are the steps for replacing the hostname:

1. Create a rewrite rule with a condition that evaluates if the location header in the response contains azurewebsites.net. Enter the pattern `(https?):\/\/.*azurewebsites\.net(.*)$`.
2. Perform an action to rewrite the location header so that it has the application gateway's hostname. Do this by entering `{http_resp_Location_1}://contoso.com{http_resp_Location_2}` as the header value. Alternatively, you can also use the server variable `host` to set the hostname to match the original request.

Implement security HTTP headers to prevent vulnerabilities

You can fix several security vulnerabilities by implementing necessary headers in the application response. These security headers include X-XSS-Protection, Strict-Transport-Security, and Content-Security-Policy. You can use Application Gateway to set these headers for all responses.

Delete unwanted headers

You might want to remove headers that reveal sensitive information from an HTTP response. For example, you might want to remove information like the back-end server name, operating system, or library details. You can use the application gateway to remove these headers:

Check for the presence of a header

You can evaluate an HTTP request or response header for the presence of a header or server variable. This evaluation is useful when you want to perform a header rewrite only when a certain header is present.

Common scenarios for URL rewrite

Parameter based path selection

To accomplish scenarios where you want to choose the backend pool based on the value of a header, part of the URL, or query string in the request, you can use the combination of URL Rewrite capability and path-based routing. For example, if you have a shopping website and the product category is passed as query string in the URL, and you want to route the request to backend based on the query string, then:

Step1: Create a path-map as shown in the image below

Step 2 (a): Create a rewrite set which has 3 rewrite rules:

- The first rule has a condition that checks the *query_string* variable for *category=shoes* and has an action that rewrites the URL path to */listing1* and has **Re-evaluate path map** enabled
- The second rule has a condition that checks the *query_string* variable for *category=bags* and has an action that rewrites the URL path to */listing2* and has **Re-evaluate path map** enabled
- The third rule has a condition that checks the *query_string* variable for *category=accessories* and has an action that rewrites the URL path to */listing3* and has **Re-evaluate path map** enabled

Step 2 (b): Associate this rewrite set with the default path of the above path-based rule

Now, if the user requests *contoso.com/listing?category=any*, then it will be matched with the default path since none of the path patterns in the path map (/listing1, /listing2, /listing3) will match. Since you associated the above rewrite set with this path, this rewrite set will be evaluated. As the query string will not match the condition in any of the 3 rewrite rules in this rewrite set, no rewrite action will take place and therefore, the request will be routed unchanged to the backend associated with the default path (which is *GenericList*).

If the user requests *contoso.com/listing?category=shoes*, then again the default path will be matched. However, in this case the condition in the first rule will match and therefore, the action associated with the condition will be executed which will rewrite the URL path to */listing1* and re-evaluate the path-map. When the path-map is re-evaluated, the request will now match the path associated with pattern */listing1* and the request will be routed to the backend associated with this pattern, which is ShoesListBackendPool.

> **NOTE**
> This scenario can be extended to any header or cookie value, URL path, query string or server variables based on the conditions defined and essentially enables you to route requests based on those conditions.

Rewrite query string parameters based on the URL

Consider a scenario of a shopping website where the user visible link should be simple and legible, but the backend server needs the query string parameters to show the right content.

In that case, Application Gateway can capture parameters from the URL and add query string key-value pairs from those from the URL. For example, let's say the user wants to rewrite, `https://www.contoso.com/fashion/shirts` to `https://www.contoso.com/buy.aspx?category=fashion&product=shirts`, it can be achieved through the following URL rewrite configuration.

Condition - If server variable `uri_path` equals to the pattern `/(.+)/(.+)`

Action - Set URL path to `buy.aspx` and query string to `category={var_uri_path_1}&product={var_uri_path_2}`

For a step-by-step guide to achieve the scenario described above, see Rewrite URL with Application Gateway using Azure portal

URL rewrite vs URL redirect

In the case of a URL rewrite, Application Gateway rewrites the URL before the request is sent to the backend. This will not change what users see in the browser because the changes are hidden from the user.

In the case of a URL redirect, Application Gateway sends a redirect response to the client with the new URL. That, in turn, requires the client to resend its request to the new URL provided in the redirect. The URL that the user sees in the browser will update to the new URL.

Limitations

- If a response has more than one header with the same name, then rewriting the value of one of those headers will result in dropping the other headers in the response. This can usually happen with Set-Cookie header since you can have more than one Set-Cookie header in a response. One such scenario is when you are using an app service with an application gateway and have configured cookie-based session affinity on the application gateway. In this case the response will contain two Set-Cookie headers: one used by the app service, for example:

  ```
  Set-Cookie:
  ARRAffinity=ba127f1caf6ac822b2347cc18bba0364d699ca1ad44d20e0ec01ea80cda2a735;Path=/;HttpOnly;Domain=sitename.azurewebsites
  ```

  and another for application gateway affinity, for example,

  `Set-Cookie: ApplicationGatewayAffinity=c1a2bd51lfd396387f96bl9cc3d2c516; Path=/`. Rewriting one of the Set-Cookie headers in this scenario can result in removing the other Set-Cookie header from the response.
- Rewrites are not supported when the application gateway is configured to redirect the requests or to show a custom error page.
- Request header names can contain alphanumeric characters and hyphens. Headers names containing other characters will be discarded when a request is sent to the backend target.
- Response header names can contain any alphanumeric characters and specific symbols as defined in RFC 7230, with the exception of underscores (_).
- Connection and upgrade headers cannot be rewritten
- Rewrites are not supported for 4xx and 5xx responses generated directly from Application Gateway

Next steps

- Learn how to rewrite HTTP headers with Application Gateway using Azure portal
- Learn how to rewrite URL with Application Gateway using Azure portal

Application Gateway Private Link (preview)

8/22/2022 • 2 minutes to read • Edit Online

Today, you can deploy your critical workloads securely behind Application Gateway, gaining the flexibility of Layer 7 load balancing features. Access to the backend workloads is possible in two ways:

- Public IP address - your workloads are accessible over the Internet.
- Private IP address- your workloads are accessible via a private IP address, but within the same VNet as the Application Gateway.

Private Link for Application Gateway allows you to connect workloads over a private connection spanning across VNets and subscriptions. When configured, a private endpoint will be placed into a defined virtual network's subnet, providing a private IP address for clients looking to communicate to the gateway. For a list of other PaaS services that support Private Link functionality, see What is Azure Private Link?.

> **IMPORTANT**
>
> Azure Application Gateway Private Link is currently in public preview.

Features and capabilities

Private Link allows you to extend private connectivity to Application Gateway via a Private Endpoint in the following scenarios:

- VNet in the same or different region from Application Gateway
- VNet in the same or different subscription from Application Gateway
- VNet in the same or different subscription and the same or different Azure AD tenant from Application Gateway

You may also choose to block inbound public (Internet) access to Application Gateway and allow access only via private endpoints. Inbound management traffic still needs to be allowed to application gateway. For more information, see Application Gateway infrastructure configuration

All features supported by Application Gateway are supported when accessed through a private endpoint, including support for AGIC.

Private Link components

Four components are required to implement Private Link with Application Gateway:

- Application Gateway Private Link Configuration

  A Private link configuration can be associated with an Application Gateway Frontend IP address, which can then be used to establish a connection using a Private Endpoint. If there's no association to an Application Gateway frontend IP address, then the Private Link feature won't be enabled.

- Application Gateway Frontend IP address

  The public or private IP address where the Application Gateway Private Link Configuration needs to be associated to enable the Private Link Capabilities.

- Private Endpoint

  An Azure network resource that allocates a private IP address in your VNet address space. It's used to connect to the Application Gateway via the private IP address similar to many other Azure Services like Storage, KeyVault, etc., that provide private link access.

- Private Endpoint Connection

  A connection on Application Gateway originated by Private Endpoints. You can auto-approve, manually approve, or reject connections to grant or deny access.

Limitations

- API version 2020-03-01 or later should be used to configure Private Link configurations.
- Static IP allocation method in the Private Link Configuration object isn't supported.
- The subnet used for PrivateLinkConfiguration cannot be same as the Application Gateway subnet.
- Private link configuration for Application Gateway does not expose the "Alias" property and must be referenced via resource URI.
- Private Endpoint creation does not create a *.privatelink DNS record/zone. All DNS records should be entered in existing zones used for your Application Gateway.
- Azure Front Door and Application Gateway do not support chaining via Private Link.
- Source IP address and x-forwarded-for headers will contain the Private link IP addresses

Next steps

- Configure Azure Application Gateway Private Link
- What is Azure Private Link?

Overview of TLS termination and end to end TLS with Application Gateway

8/22/2022 • 13 minutes to read • Edit Online

Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL), is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and encrypted. Application gateway supports both TLS termination at the gateway as well as end to end TLS encryption.

TLS termination

Application Gateway supports TLS termination at the gateway, after which traffic typically flows unencrypted to the backend servers. There are a number of advantages of doing TLS termination at the application gateway:

- **Improved performance** – The biggest performance hit when doing TLS decryption is the initial handshake. To improve performance, the server doing the decryption caches TLS session IDs and manages TLS session tickets. If this is done at the application gateway, all requests from the same client can use the cached values. If it's done on the backend servers, then each time the client's requests go to a different server the client must reauthenticate. The use of TLS tickets can help mitigate this issue, but they are not supported by all clients and can be difficult to configure and manage.
- **Better utilization of the backend servers** – SSL/TLS processing is very CPU intensive, and is becoming more intensive as key sizes increase. Removing this work from the backend servers allows them to focus on what they are most efficient at, delivering content.
- **Intelligent routing** – By decrypting the traffic, the application gateway has access to the request content, such as headers, URI, and so on, and can use this data to route requests.
- **Certificate management** – Certificates only need to be purchased and installed on the application gateway and not all backend servers. This saves both time and money.

To configure TLS termination, a TLS/SSL certificate must be added to the listener. This allows the Application Gateway to decrypt incoming traffic and encrypt response traffic to the client. The certificate provided to the Application Gateway must be in Personal Information Exchange (PFX) format, which contains both the private and public keys.

> **IMPORTANT**
>
> The certificate on the listener requires the entire certificate chain to be uploaded (the root certificate from the CA, the intermediates and the leaf certificate) to establish the chain of trust.

> **NOTE**
>
> Application gateway does not provide any capability to create a new certificate or send a certificate request to a certification authority.

For the TLS connection to work, you need to ensure that the TLS/SSL certificate meets the following conditions:

- That the current date and time is within the "Valid from" and "Valid to" date range on the certificate.
- That the certificate's "Common Name" (CN) matches the host header in the request. For example, if the client is making a request to `https://www.contoso.com/`, then the CN must be `www.contoso.com`.

Certificates supported for TLS termination

Application gateway supports the following types of certificates:

- CA (Certificate Authority) certificate: A CA certificate is a digital certificate issued by a certificate authority (CA)
- EV (Extended Validation) certificate: An EV certificate is a certificate that conforms to industry standard certificate guidelines. This will turn the browser locator bar green and publish the company name as well.
- Wildcard Certificate: This certificate supports any number of subdomains based on *.site.com, where your subdomain would replace the *. It doesn't, however, support site.com, so in case the users are accessing your website without typing the leading "www", the wildcard certificate will not cover that.
- Self-Signed certificates: Client browsers do not trust these certificates and will warn the user that the virtual service's certificate is not part of a trust chain. Self-signed certificates are good for testing or environments where administrators control the clients and can safely bypass the browser's security alerts. Production workloads should never use self-signed certificates.

For more information, see configure TLS termination with application gateway.

Size of the certificate

Check the Application Gateway limits section to know the maximum TLS/SSL certificate size supported.

End-to-end TLS encryption

You may not want unencrypted communication to the backend servers. You may have security requirements, compliance requirements, or the application may only accept a secure connection. Azure Application Gateway has end-to-end TLS encryption to support these requirements.

End-to-end TLS allows you to encrypt and securely transmit sensitive data to the backend while you use Application Gateway's Layer-7 load-balancing features. These features include cookie-based session affinity, URL-based routing, support for routing based on sites, the ability to rewrite or inject X-Forwarded-* headers, and so on.

When configured with end-to-end TLS communication mode, Application Gateway terminates the TLS sessions at the gateway and decrypts user traffic. It then applies the configured rules to select an appropriate backend pool instance to route traffic to. Application Gateway then initiates a new TLS connection to the backend server and re-encrypts data using the backend server's public key certificate before transmitting the request to the backend. Any response from the web server goes through the same process back to the end user. End-to-end TLS is enabled by setting protocol setting in Backend HTTP Setting to HTTPS, which is then applied to a backend pool.

For the Application Gateway and WAF v1 SKU, the TLS policy applies to both frontend and backend traffic. On the front end, Application Gateway acts as the server and enforces the policy. On the backend, Application Gateway acts as the client and sends the protocol/cipher information as the preference during the TLS handshake.

For the Application Gateway and WAF v2 SKU, the TLS policy applies only to the frontend traffic and all ciphers are offered to the backend server, which has control to select specific ciphers and TLS version during the handshake.

Application Gateway only communicates with those backend servers that have either allow listed their certificate with the Application Gateway or whose certificates are signed by well-known CA authorities and the certificate's CN matches the host name in the HTTP backend settings. These include the trusted Azure services such as Azure App Service/Web Apps and Azure API Management.

If the certificates of the members in the backend pool are not signed by well-known CA authorities, then each instance in the backend pool with end to end TLS enabled must be configured with a certificate to allow secure

communication. Adding the certificate ensures that the application gateway only communicates with known back-end instances. This further secures the end-to-end communication.

> **NOTE**
>
> The certificate added to **Backend HTTP Setting** to authenticate the backend servers can be the same as the certificate added to the **listener** for TLS termination at application gateway or different for enhanced security.

In this example, requests using TLS1.2 are routed to backend servers in Pool1 using end to end TLS.

End to end TLS and allow listing of certificates

Application Gateway only communicates with those backend servers that have either allow listed their certificate with the Application Gateway or whose certificates are signed by well-known CA authorities and the certificate's CN matches the host name in the HTTP backend settings. There are some differences in the end-to-end TLS setup process with respect to the version of Application Gateway used. The following section explains them individually.

End-to-end TLS with the v1 SKU

To enable end-to-end TLS with the backend servers and for Application Gateway to route requests to them, the health probes must succeed and return healthy response.

For HTTPS health probes, the Application Gateway v1 SKU uses an exact match of the authentication certificate (public key of the backend server certificate and not the root certificate) to be uploaded to the HTTP settings.

Only connections to known and allowed backends are then allowed. The remaining backends are considered unhealthy by the health probes. Self-signed certificates are for test purposes only and not recommended for production workloads. Such certificates must be allow listed with the application gateway as described in the preceding steps before they can be used.

> **NOTE**
> Authentication and trusted root certificate setup are not required for trusted Azure services such as Azure App Service. They are considered trusted by default.

End-to-end TLS with the v2 SKU

Authentication Certificates have been deprecated and replaced by Trusted Root Certificates in the Application Gateway v2 SKU. They function similarly to Authentication Certificates with a few key differences:

- Certificates signed by well known CA authorities whose CN matches the host name in the HTTP backend settings do not require any additional step for end to end TLS to work.

  For example, if the backend certificates are issued by a well known CA and has a CN of contoso.com, and the backend http setting's host field is also set to contoso.com, then no additional steps are required. You can set the backend http setting protocol to HTTPS and both the health probe and data path would be TLS enabled. If you're using Azure App Service or other Azure web services as your backend, then these are implicitly trusted as well and no further steps are required for end to end TLS.

> **NOTE**
> In order for a TLS/SSL certificate to be trusted, that certificate of the backend server must have been issued by a CA that is well-known. If the certificate was not issued by a trusted CA, the application gateway will then check to see if the certificate of the issuing CA was issued by a trusted CA, and so on until either a trusted CA is found (at which point a trusted, secure connection will be established) or no trusted CA can be found (at which point the application gateway will mark the backend unhealthy). Therefore, it is recommended the backend server certificate contain both the root and intermediate CAs.

- If the backend server certificate is self-signed, or signed by unknown CA/intermediaries, then to enable end to end TLS in Application Gateway v2 a trusted root certificate must be uploaded. Application Gateway will only communicate with backends whose server certificate's root certificate matches one of the list of trusted root certificates in the backend http setting associated with the pool.
- In addition to the root certificate match, Application Gateway v2 also validates if the Host setting specified in the backend http setting matches that of the common name (CN) presented by the backend server's TLS/SSL certificate. When trying to establish a TLS connection to the backend, Application Gateway v2 sets the Server Name Indication (SNI) extension to the Host specified in the backend http setting.
- If **pick hostname from backend target** is chosen instead of the Host field in the backend http setting, then the SNI header is always set to the backend pool FQDN and the CN on the backend server TLS/SSL certificate must match its FQDN. Backend pool members with IPs aren't supported in this scenario.
- The root certificate is a base64 encoded root certificate from the backend server certificates.

SNI differences in the v1 and v2 SKU

As mentioned previously, Application Gateway terminates TLS traffic from the client at the Application Gateway Listener (let's call it the frontend connection), decrypts the traffic, applies the necessary rules to determine the backend server to which the request has to be forwarded, and establishes a new TLS session with the backend server (let's call it the backend connection).

The following tables outline the differences in SNI between the v1 and v2 SKU in terms of frontend and backend connections.

Frontend TLS connection (client to application gateway)

SCENARIO	V1	V2
If the client specifies SNI header and all the multi-site listeners are enabled with "Require SNI" flag	Returns the appropriate certificate and if the site doesn't exist (according to the server_name), then the connection is reset.	Returns appropriate certificate if available, otherwise, returns the certificate of the first HTTPS listener according to the order specified by the request routing rules associated with the HTTPS listeners
If the client doesn't specify a SNI header and if all the multi-site headers are enabled with "Require SNI"	Resets the connection	Returns the certificate of the first HTTPS listener according to the order specified by the request routing rules associated with the HTTPS listeners
If the client doesn't specify SNI header and if there's a basic listener configured with a certificate	Returns the certificate configured in the basic listener to the client (default or fallback certificate)	Returns the certificate configured in the basic listener

Backend TLS connection (application gateway to the backend server)

For probe traffic

SCENARIO	V1	V2
SNI (server_name) header during the TLS handshake as FQDN	Set as FQDN from the backend pool. As per RFC 6066, literal IPv4 and IPv6 addresses are not permitted in SNI hostname. **Note:** FQDN in the backend pool should DNS resolve to backend server's IP address (public or private)	SNI header (server_name) is set as the hostname from the custom probe attached to the HTTP settings (if configured), otherwise from the hostname mentioned in the HTTP settings, otherwise from the FQDN mentioned in the backend pool. The order of precedence is custom probe > HTTP settings > backend pool. **Note:** If the hostnames configured in HTTP settings and custom probe are different, then according to the precedence, SNI will be set as the hostname from the custom probe.
If the backend pool address is an IP address (v1) or if custom probe hostname is configured as IP address (v2)	SNI (server_name) won't be set. **Note:** In this case, the backend server should be able to return a default/fallback certificate and this should be allow listed in HTTP settings under authentication certificate. If there's no default/fallback certificate configured in the backend server and SNI is expected, the server might reset the connection and will lead to probe failures	In the order of precedence mentioned previously, if they have IP address as hostname, then SNI won't be set as per RFC 6066. **Note:** SNI also won't be set in v2 probes if no custom probe is configured and no hostname is set on HTTP settings or backend pool

> **NOTE**
>
> If a custom probe is not configured, then Application Gateway sends a default probe in this format - <protocol>://127.0.0.1:<port>/. For example, for a default HTTPS probe, it'll be sent as https://127.0.0.1:443/. Note that, the 127.0.0.1 mentioned here is only used as HTTP host header and as per RFC 6066, will not be used as SNI header. For more information on health probe errors, check the backend health troubleshooting guide.

For live traffic

SCENARIO	V1	V2
SNI (server_name) header during the TLS handshake as FQDN	Set as FQDN from the backend pool. As per RFC 6066, literal IPv4 and IPv6 addresses are not permitted in SNI hostname. **Note:** FQDN in the backend pool should DNS resolve to backend server's IP address (public or private)	SNI header (server_name) is set as the hostname from the HTTP settings, otherwise, if *PickHostnameFromBackendAddress* option is chosen or if no hostname is mentioned, then it'll be set as the FQDN in the backend pool configuration
If the backend pool address is an IP address or hostname is not set in HTTP settings	SNI won't be set as per RFC 6066 if the backend pool entry is not an FQDN	SNI will be set as the hostname from the input FQDN from the client and the backend certificate's CN has to match with this hostname.
Hostname is not provided in HTTP Settings, but a FQDN is specified as the Target for a backend pool member	SNI will be set as the hostname from the input FQDN from the client and the backend certificate's CN has to match with this hostname.	SNI will be set as the hostname from the input FQDN from the client and the backend certificate's CN has to match with this hostname.

Next steps

After learning about end to end TLS, go to Configure end to end TLS by using Application Gateway with PowerShell to create an application gateway using end to end TLS.

Application Gateway TLS policy overview

8/22/2022 • 5 minutes to read • Edit Online

You can use Azure Application Gateway to centralize TLS/SSL certificate management and reduce encryption and decryption overhead from a back-end server farm. This centralized TLS handling also lets you specify a central TLS policy that's suited to your organizational security requirements. This helps you meet compliance requirements as well as security guidelines and recommended practices.

The TLS policy includes control of the TLS protocol version as well as the cipher suites and the order in which ciphers are used during a TLS handshake. Application Gateway offers two mechanisms for controlling TLS policy. You can use either a predefined policy or a custom policy.

Usage and version details

- SSL 2.0 and 3.0 are disabled for all application gateways and are not configurable.
- A custom TLS policy allows you to select any TLS protocol as the minimum protocol version for your gateway: TLSv1_0, TLSv1_1, TLSv1_2, or TLSv1_3.
- If no TLS policy is defined, the minimum protocol version is set to TLSv1_0, and protocol versions v1.0, v1.1, and v1.2 are supported.
- The new **Predefined and Customv2 policies** that support **TLS v1.3** are currently in **Preview** and only available with Application Gateway V2 SKUs (Standard_v2 or WAF_v2).
- Using a new Predefined or Customv2 policy enhances SSL security and performance posture of the entire gateway (for SSL Policy and SSL Profile). Hence, both old and new policies cannot co-exist on a gateway. You must use any of the older predefined or custom policies across the gateway if clients require older TLS versions or ciphers (for example, TLS v1.0).
- TLS cipher suites used for the connection are also based on the type of the certificate being used. The cipher suites used in "client to application gateway connections" are based on the type of listener certificates on the application gateway. Whereas the cipher suites used in establishing "application gateway to backend pool connections" are based on the type of server certificates presented by the backend servers.

Predefined TLS policy

Application Gateway offers several predefined security policies. You can configure your gateway with any of these policies to get the appropriate level of security. The policy names are annotated by the year and month in which they were configured (AppGwSslPolicy<YYYYMMDD>). Each policy offers different TLS protocol versions and/or cipher suites. These predefined policies are configured keeping in mind the best practices and recommendations from the Microsoft Security team. We recommend that you use the newest TLS policies to ensure the best TLS security.

The following table shows the list of cipher suites and minimum protocol version support for each predefined policy. The ordering of the cipher suites determines the priority order during TLS negotiation. To know the exact ordering of the cipher suites for these predefined policies, you can refer to the PowerShell, CLI, REST API or the Listeners blade in portal.

PREDEFINED POLICY NAMES (APPGWSSLPOLICY<YYYYMMDD>)	20150501	20170401	20170401S	20220101 (PREVIEW)	20220101S (PREVIEW)
Minimum Protocol Version	1.0	1.1	1.2	1.2	1.2
Enabled protocol versions	1.0 1.1 1.2	1.1 1.2	1.2	1.2 1.3	1.2 1.3
Default	True	False	False	False	False
TLS_AES_128_GCM_SHA256	✗	✗	✗	✓	✓
TLS_AES_256_GCM_SHA384	✗	✗	✗	✓	✓
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✓	✓
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✓	✓
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	✓	✗	✗	✓	✗
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	✓	✗	✗	✓	✗
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	✓	✓	✓	✗	✗
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	✓	✓	✓	✗	✗
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	✓	✗	✗	✗	✗
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	✓	✗	✗	✗	✗
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	✓	✗	✗	✗	✗

PREDEFINED POLICY NAMES (APPGWSSLPOLICY<YYYYMMDD>)	20150501	20170401	20170401S	20220101 (PREVIEW)	20220101S (PREVIEW)
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	✓	✗	✗	✗	✗
TLS_RSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✗	✗
TLS_RSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✗	✗
TLS_RSA_WITH_AES_256_CBC_SHA256	✓	✓	✓	✗	✗
TLS_RSA_WITH_AES_128_CBC_SHA256	✓	✓	✓	✗	✗
TLS_RSA_WITH_AES_256_CBC_SHA	✓	✓	✓	✗	✗
TLS_RSA_WITH_AES_128_CBC_SHA	✓	✓	✓	✗	✗
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	✓	✓	✓	✓	✗
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	✓	✓	✓	✓	✗
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	✓	✓	✓	✗	✗
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	✓	✓	✓	✗	✗

PREDEFINED POLICY NAMES (APPGWSSLPOLICY<YYYYMMDD>)	20150501	20170401	20170401S	20220101 (PREVIEW)	20220101S (PREVIEW)
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	✓	✗	✗	✗	✗
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	✓	✗	✗	✗	✗
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	✓	✗	✗	✗	✗
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	✓	✗	✗	✗	✗
TLS_RSA_WITH_3DES_EDE_CBC_SHA	✓	✗	✗	✗	✗
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	✓	✗	✗	✗	✗

Custom TLS policy

If a TLS policy needs to be configured for your requirements, you can use a Custom TLS policy. With a custom TLS policy, you have complete control over the minimum TLS protocol version to support, as well as the supported cipher suites and their priority order.

> **NOTE**
> The newer, stronger ciphers and TLSv1.3 support are only available with the **CustomV2 policy (Preview)**. It provides enhanced security and performance benefits.

> **IMPORTANT**
> - If you are using a custom SSL policy in Application Gateway v1 SKU (Standard or WAF), make sure that you add the mandatory cipher "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256" to the list. This cipher is required to enable metrics and logging in the Application Gateway v1 SKU. This is not mandatory for Application Gateway v2 SKU (Standard_v2 or WAF_v2).
> - The cipher suites “TLS_AES_128_GCM_SHA256” and “TLS_AES_256_GCM_SHA384” are mandatory for TLSv1.3. You need NOT mention these explicitly when setting a CustomV2 policy with minimum protocol version 1.2 or 1.3 through PowerShell or CLI. Accordingly, these ciphers suites will not appear in the Get Details output, with an exception of Portal.

Cipher suites

Application Gateway supports the following cipher suites from which you can choose your custom policy. The ordering of the cipher suites determines the priority order during TLS negotiation.

- TLS_AES_128_GCM_SHA256 (available only with Customv2)
- TLS_AES_256_GCM_SHA384 (available only with Customv2)
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

Limitations

- The connections to backend servers are always with minimum protocol TLS v1.0 and up to TLS v1.2. Therefore, only TLS versions 1.0, 1.1 and 1.2 are supported to establish a secured connection with backend servers.
- As of now, the TLS 1.3 implementation is not enabled with "Zero Round Trip Time (0-RTT)" feature.
- Application Gateway v2 does not support the following DHE ciphers. These won't be used for the TLS connections with clients even though they are mentioned in the predefined policies. Instead of DHE ciphers, secure and faster ECDHE ciphers are recommended.
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  - TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
  - TLS_DHE_DSS_WITH_AES_128_CBC_SHA
  - TLS_DHE_DSS_WITH_AES_256_CBC_SHA256

- TLS_DHE_DSS_WITH_AES_256_CBC_SHA

Next steps

If you want to learn to configure a TLS policy, see Configure TLS policy versions and cipher suites on Application Gateway.

Overview of mutual authentication with Application Gateway

8/22/2022 • 4 minutes to read • Edit Online

Mutual authentication, or client authentication, allows for the Application Gateway to authenticate the client sending requests. Usually, only the client is authenticating the Application Gateway; mutual authentication allows for both the client and the Application Gateway to authenticate each other.

> **NOTE**
>
> We recommend using TLS 1.2 with mutual authentication as TLS 1.2 will be mandated in the future.

Mutual authentication

Application Gateway supports certificate-based mutual authentication where you can upload a trusted client CA certificate(s) to the Application Gateway, and the gateway will use that certificate to authenticate the client sending a request to the gateway. With the rise in IoT use cases and increased security requirements across industries, mutual authentication provides a way for you to manage and control which clients can talk to your Application Gateway.

To configure mutual authentication, a trusted client CA certificate is required to be uploaded as part of the client authentication portion of an SSL profile. The SSL profile will then need to be associated to a listener in order to complete configuration of mutual authentication. There must always be a root CA certificate in the client certificate that you upload. You can upload a certificate chain as well, but the chain must include a root CA certificate in addition to as many intermediate CA certificates as you'd like.

For example, if your client certificate contains a root CA certificate, multiple intermediate CA certificates, and a leaf certificate, make sure that the root CA certificate and all the intermediate CA certificates are uploaded onto Application Gateway in one file. For more information on how to extract a trusted client CA certificate, see how to extract trusted client CA certificates.

If you're uploading a certificate chain with root CA and intermediate CA certificates, the certificate chain must be uploaded as a PEM or CER file to the gateway.

> **NOTE**
>
> Mutual authentication is only available on Standard_v2 and WAF_v2 SKUs.

Certificates supported for mutual authentication

Application Gateway supports the following types of certificates:

- CA (Certificate Authority) certificate: A CA certificate is a digital certificate issued by a certificate authority (CA).
- Self-signed CA certificates: Client browsers do not trust these certificates and will warn the user that the virtual service's certificate is not part of a trust chain. Self-signed CA certificates are good for testing or in environments where administrators control the clients and can safely bypass the browser's security alerts.

> **IMPORTANT**
> Production workloads should never use self-signed CA certificates.

For more information on how to set up mutual authentication, see configure mutual authentication with Application Gateway.

> **IMPORTANT**
> Make sure you upload the entire trusted client CA certificate chain to the Application Gateway when using mutual authentication.

Each SSL profile can support up to five trusted client CA certificate chains.

Additional client authentication validation

Verify client certificate DN

You have the option to verify the client certificate's immediate issuer and only allow the Application Gateway to trust that issuer. This options is off by default but you can enable this through Portal, PowerShell, or Azure CLI.

If you choose to enable the Application Gateway to verify the client certificate's immediate issuer, here's how to determine what client certificate issuer DN will be extracted from the certificates uploaded.

- **Scenario 1:** Certificate chain includes: root certificate - intermediate certificate - leaf certificate
  - *Intermediate certificate's* subject name is what Application Gateway will extract as the client certificate issuer DN and will be verified against.
- **Scenario 2:** Certificate chain includes: root certificate - intermediate1 certificate - intermediate2 certificate - leaf certificate
  - *Intermediate2 certificate's* subject name will be what's extracted as the client certificate issuer DN and will be verified against.
- **Scenario 3:** Certificate chain includes: root certificate - leaf certificate
  - *Root certificate's* subject name will be extracted and used as client certificate issuer DN.
- **Scenario 4:** Multiple certificate chains of the same length in the same file. Chain 1 includes: root certificate - intermediate1 certificate - leaf certificate. Chain 2 includes: root certificate - intermediate2 certificate - leaf certificate.
  - *Intermediate1 certificate's* subject name will be extracted as client certificate issuer DN.
- **Scenario 5:** Multiple certificate chains of different lengths in the same file. Chain 1 includes: root certificate - intermediate1 certificate - leaf certificate. Chain 2 includes root certificate - intermediate2 certificate - intermediate3 certificate - leaf certificate.
  - *Intermediate3 certificate's* subject name will be extracted as client certificate issuer DN.

> **IMPORTANT**
> We recommend only uploading one certificate chain per file. This is especially important if you enable verify client certificate DN. By uploading multiple certificate chains in one file, you will end up in scenario four or five and may run into issues later down the line when the client certificate presented doesn't match the client certificate issuer DN Application Gateway extracted from the chains.

For more information on how to extract trusted client CA certificate chains, see how to extract trusted client CA certificate chains.

Server variables

With mutual authentication, there are additional server variables that you can use to pass information about the client certificate to the backend servers behind the Application Gateway. For more information about which server variables are available and how to use them, check out server variables.

Certificate Revocation

Client certificate revocation with OCSP (Online Certificate Status Protocol) will be supported shortly.

Next steps

After learning about mutual authentication, go to Configure Application Gateway with mutual authentication in PowerShell to create an Application Gateway using mutual authentication.

TLS termination with Key Vault certificates

8/22/2022 • 8 minutes to read • Edit Online

Azure Key Vault is a platform-managed secret store that you can use to safeguard secrets, keys, and TLS/SSL certificates. Azure Application Gateway supports integration with Key Vault for server certificates that are attached to HTTPS-enabled listeners. This support is limited to the v2 SKU of Application Gateway.

Application Gateway offers two models for TLS termination:

- Provide TLS/SSL certificates attached to the listener. This model is the traditional way to pass TLS/SSL certificates to Application Gateway for TLS termination.
- Provide a reference to an existing Key Vault certificate or secret when you create a HTTPS-enabled listener.

Application Gateway integration with Key Vault offers many benefits, including:

- Stronger security, because TLS/SSL certificates aren't directly handled by the application development team. Integration allows a separate security team to:
  - Set up application gateways.
  - Control application gateway lifecycles.
  - Grant permissions to selected application gateways to access certificates that are stored in your Key Vault.
- Support for importing existing certificates into your Key Vault. Or use Key Vault APIs to create and manage new certificates with any of the trusted Key Vault partners.
- Support for automatic renewal of certificates that are stored in your Key Vault.

Supported certificates

Application Gateway currently supports software-validated certificates only. Hardware security module (HSM)-validated certificates aren't supported.

After Application Gateway is configured to use Key Vault certificates, its instances retrieve the certificate from Key Vault and install them locally for TLS termination. The instances poll Key Vault at four-hour intervals to retrieve a renewed version of the certificate, if it exists. If an updated certificate is found, the TLS/SSL certificate that's currently associated with the HTTPS listener is automatically rotated.

> **TIP**
> Any change to Application Gateway will force a check against Key Vault to see if any new versions of certificates are available. This includes, but not limited to, changes to Frontend IP Configurations, Listeners, Rules, Backend Pools, Resource Tags, and more. If an updated certificate is found, the new certificate will immediately be presented.

Application Gateway uses a secret identifier in Key Vault to reference the certificates. For Azure PowerShell, the Azure CLI, or Azure Resource Manager, we strongly recommend that you use a secret identifier that doesn't specify a version. This way, Application Gateway will automatically rotate the certificate if a newer version is available in your Key Vault. An example of a secret URI without a version is `https://myvault.vault.azure.net/secrets/mysecret/`.

The Azure portal supports only Key Vault certificates, not secrets. Application Gateway still supports referencing secrets from Key Vault, but only through non-portal resources like PowerShell, the Azure CLI, APIs, and Azure Resource Manager templates (ARM templates).

References to Key Vaults in other Azure subscriptions is supported, but must be configured via ARM Template, Azure PowerShell, CLI, Bicep, etc. Cross-subscription key vault configuration is not supported by Application Gateway via Azure Portal today.

Certificate settings in Key Vault

For TLS termination, Application Gateway only supports certificates in Personal Information Exchange (PFX) format. You can either import an existing certificate or create a new one in your Key Vault. To avoid any failures, ensure that the certificate's status is set to **Enabled** in Key Vault.

How integration works

Application Gateway integration with Key Vault is a three-step configuration process:

> **NOTE**
>
> Azure Application Gateway integration with Key Vault supports both Vault access policy and Azure role-based access control permission models.

Obtain a user-assigned managed identity

Application Gateway uses a managed identity to retrieve certificates from Key Vault on your behalf.

You can either create a new user-assigned managed identity or reuse an existing with the integration. To create a new user-assigned managed identity, see Create a user-assigned managed identity using the Azure portal.

Delegate user-assigned managed identity to Key Vault

Define access policies to use the user-assigned managed identity with your Key Vault:

1. In the Azure portal, go to **Key Vault**.
2. Select the Key Vault that contains your certificate.
3. If you're using the permission model **Vault access policy**: Select **Access Policies**, select **+ Add Access Policy**, select **Get** for **Secret permissions**, and choose your user-assigned managed identity for **Select principal**. Then select **Save**.

   If you're using **Azure role-based access control** follow the article Assign a managed identity access to a resource and assign the user-assigned managed identity the **Key Vault Secrets User** role to the Azure Key Vault.

Verify Firewall Permissions to Key Vault

As of March 15, 2021, Key Vault recognizes Application Gateway as a trusted service by leveraging User Managed Identities for authentication to Azure Key Vault. With the use of service endpoints and enabling the trusted services option for Key Vault's firewall, you can build a secure network boundary in Azure. You can deny access to traffic from all networks (including internet traffic) to Key Vault but still make Key Vault accessible for

an Application Gateway resource under your subscription.

When you're using a restricted Key Vault, use the following steps to configure Application Gateway to use firewalls and virtual networks:

> **TIP**
>
> Steps 1-3 are not required if your Key Vault has a Private Endpoint enabled. The application gateway can access the Key Vault using the private IP address.

1. In the Azure portal, in your Key Vault, select **Networking**.
2. On the **Firewalls and virtual networks** tab, select **Selected networks**.
3. For **Virtual networks**, select **+ Add existing virtual networks**, and then add the virtual network and subnet for your Application Gateway instance. During the process, also configure the `Microsoft.KeyVault` service endpoint by selecting its checkbox.
4. Select **Yes** to allow trusted services to bypass the Key Vault's firewall.

> **NOTE**
>
> If you deploy the Application Gateway instance via an ARM template by using either the Azure CLI or PowerShell, or via an Azure application deployed from the Azure portal, the SSL certificate is stored in the Key Vault as a Base64-encoded PFX file. You must complete the steps in Use Azure Key Vault to pass secure parameter value during deployment.
>
> It's particularly important to set `enabledForTemplateDeployment` to `true`. The certificate might or might not have a password. In the case of a certificate with a password, the following example shows a possible configuration for the `sslCertificates` entry in `properties` for the ARM template configuration for Application Gateway.
>
> ```
> "sslCertificates": [
>  {
>  "name": "appGwSslCertificate",
>  "properties": {
>  "data": "[parameters('appGatewaySSLCertificateData')]",
>  "password": "[parameters('appGatewaySSLCertificatePassword')]"
>  }
>  }
> ]
> ```
>
> The values of `appGatewaySSLCertificateData` and `appGatewaySSLCertificatePassword` are looked up from the Key Vault, as described in Reference secrets with dynamic ID. Follow the references backward from `parameters('secretName')` to see how the lookup happens. If the certificate is passwordless, omit the `password` entry.

Configure Application Gateway Listener

Key Vault permission Vault access policy model

Navigate to your Application Gateway in the Azure portal and select the **Listeners** tab. Select **Add Listener** (or select an existing listener) and specify **HTTPS** for the protocol.

Under **Choose a certificate**, select **Create new** and then select **Choose a certificate from Key Vault** under **Https settings**.

For Cert name, type a friendly name for the certificate to be referenced in Key Vault. Choose your Managed identity, Key Vault, and Certificate.

Once selected, select **Add** (if creating) or **Save** (if editing) to apply the referenced Key Vault certificate to the listener.

Key Vault Azure role-based access control permission model

Application Gateway supports certificates referenced in Key Vault via the Role-based access control permission model. The first few steps to reference the Key Vault must be completed via ARM template, Bicep, CLI, or PowerShell.

> **NOTE**
>
> Specifying Azure Key Vault certificates that are subject to the role-based access control permission model is not supported via the portal.

In this example, we'll use PowerShell to reference a new Key Vault secret.

```
# Get the Application Gateway we want to modify
$appgw = Get-AzApplicationGateway -Name MyApplicationGateway -ResourceGroupName MyResourceGroup
# Specify the resource id to the user assigned managed identity - This can be found by going to the
properties of the managed identity
Set-AzApplicationGatewayIdentity -ApplicationGateway $appgw -UserAssignedIdentityId
"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/MyMan
agedIdentity"
# Get the secret ID from Key Vault
$secret = Get-AzKeyVaultSecret -VaultName "MyKeyVault" -Name "CertificateName"
$secretId = $secret.Id.Replace($secret.Version, "") # Remove the secret version so AppGW will use the latest
version in future syncs
# Specify the secret ID from Key Vault
Add-AzApplicationGatewaySslCertificate -KeyVaultSecretId $secretId -ApplicationGateway $appgw -Name
$secret.Name
# Commit the changes to the Application Gateway
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Once the commands have been executed, you can navigate to your Application Gateway in the Azure portal and select the Listeners tab. Click Add Listener (or select an existing) and specify the Protocol to HTTPS.

Under **Choose a certificate** select the certificate named in the previous steps. Once selected, select *Add* (if creating) or *Save* (if editing) to apply the referenced Key Vault certificate to the listener.

Investigating and resolving Key Vault errors

> **NOTE**
>
> It is important to consider any impact on your Application Gateway resource when making changes or revoking access to your Key Vault resource. In case your application gateway is unable to access the associated key vault or locate the certificate object in it, it will automatically put that listener in a disabled state.
>
> You can identify this user-driven event by viewing the Resource Health for your Application Gateway. Learn more.

Azure Application Gateway doesn't just poll for the renewed certificate version on Key Vault at every four-hour interval. It also logs any error and is integrated with Azure Advisor to surface any misconfiguration with a recommendation for its fix.

1. Sign-in to your Azure portal
2. Select Advisor
3. Select Operational Excellence category from the left menu.
4. You will find a recommendation titled **Resolve Azure Key Vault issue for your Application Gateway**, if your gateway is experiencing this issue. Ensure the correct Subscription is selected from the drop-down options above.
5. Select it to view the error details, the associated key vault resource and the troubleshooting guide to fix your exact issue.

By identifying such an event through Azure Advisor or Resource Health, you can quickly resolve any configuration problems with your Key Vault. We strongly recommend you take advantage of Azure Advisor and Resource Health alerts to stay informed when a problem is detected.

For Advisor alert, use "Resolve Azure Key Vault issue for your Application Gateway" in the recommendation type as shown below.

Home > Advisor > Resolve Azure Key Vault issue for your Application Gateway >

Create Advisor Alerts

Create an alert to trigger an action any time you have a new recommendation that meets the criteria you specify below.

SCOPE

Tell us the subscription and resource group you want to receive new recommendation alerts for.

Subscription * My Subscription 1

Resource Group

CONDITION

Configure your alerts to only show for the recommendation type or category and impact level that you care about.

Signal New recommendation is available (recommendations)

Configured by Category and impact level Recommendation Type

Recommendation Type Resolve Azure Key Vault issue for your Application Gateway

ACTION GROUPS

Notify your team via email and text message or automate actions using webhooks, runbooks, functions, logic apps or integrating with external ITSM solutions each time you have a new recommendation. Learn more

ACTION GROUP NAME	ACTION GROUP TYPE
* Select an existing action group or create a new one	

Select existing Create new

ALERT DETAILS

Provide details on your alert so that you can identify and manage it later.

Alert rule name * Specify alert rule name

Description Describe your alert

Enable rule upon creation Yes No

Save alert to resource group *

You can configure the Resource health alert as illustrated below.

Next steps

Configure TLS termination with Key Vault certificates by using Azure PowerShell

What is Application Gateway Ingress Controller?

8/22/2022 • 3 minutes to read • Edit Online

The Application Gateway Ingress Controller (AGIC) is a Kubernetes application, which makes it possible for Azure Kubernetes Service (AKS) customers to leverage Azure's native Application Gateway L7 load-balancer to expose cloud software to the Internet. AGIC monitors the Kubernetes cluster it is hosted on and continuously updates an Application Gateway, so that selected services are exposed to the Internet.

The Ingress Controller runs in its own pod on the customer's AKS. AGIC monitors a subset of Kubernetes Resources for changes. The state of the AKS cluster is translated to Application Gateway specific configuration and applied to the Azure Resource Manager (ARM).

Benefits of Application Gateway Ingress Controller

AGIC helps eliminate the need to have another load balancer/public IP in front of the AKS cluster and avoids multiple hops in your datapath before requests reach the AKS cluster. Application Gateway talks to pods using their private IP directly and does not require NodePort or KubeProxy services. This also brings better performance to your deployments.

Ingress Controller is supported exclusively by Standard_v2 and WAF_v2 SKUs, which also brings you autoscaling benefits. Application Gateway can react in response to an increase or decrease in traffic load and scale accordingly, without consuming any resources from your AKS cluster.

Using Application Gateway in addition to AGIC also helps protect your AKS cluster by providing TLS policy and Web Application Firewall (WAF) functionality.

AGIC is configured via the Kubernetes Ingress resource, along with Service and Deployments/Pods. It provides a number of features, leveraging Azure's native Application Gateway L7 load balancer. To name a few:

- URL routing
- Cookie-based affinity

- TLS termination
- End-to-end TLS
- Support for public, private, and hybrid web sites
- Integrated web application firewall

Difference between Helm deployment and AKS Add-On

There are two ways to deploy AGIC for your AKS cluster. The first way is through Helm; the second is through AKS as an add-on. The primary benefit of deploying AGIC as an AKS add-on is that it's much simpler than deploying through Helm. For a new setup, you can deploy a new Application Gateway and a new AKS cluster with AGIC enabled as an add-on in one line in Azure CLI. The add-on is also a fully managed service, which provides added benefits such as automatic updates and increased support. Both ways of deploying AGIC (Helm and AKS add-on) are fully supported by Microsoft. Additionally, the add-on allows for better integration with AKS as a first class add-on.

The AGIC add-on is still deployed as a pod in the customer's AKS cluster, however, there are a few differences between the Helm deployment version and the add-on version of AGIC. Below is a list of differences between the two versions:

- Helm deployment values cannot be modified on the AKS add-on:
  - `verbosityLevel` will be set to 5 by default
  - `usePrivateIp` will be set to be false by default; this can be overwritten by the use-private-ip annotation
  - `shared` is not supported on add-on
  - `reconcilePeriodSeconds` is not supported on add-on
  - `armAuth.type` is not supported on add-on
- AGIC deployed via Helm supports ProhibitedTargets, which means AGIC can configure the Application Gateway specifically for AKS clusters without affecting other existing backends. AGIC add-on doesn't currently support this.
- Since AGIC add-on is a managed service, customers will automatically be updated to the latest version of AGIC add-on, unlike AGIC deployed through Helm where the customer must manually update AGIC.

> **NOTE**
> Customers can only deploy one AGIC add-on per AKS cluster, and each AGIC add-on currently can only target one Application Gateway. For deployments that require more than one AGIC per cluster or multiple AGICs targeting one Application Gateway, please continue to use AGIC deployed through Helm.

Next steps

- **AKS Add-On Greenfield Deployment**: Instructions on installing AGIC add-on, AKS, and Application Gateway on blank-slate infrastructure.
- **AKS Add-On Brownfield Deployment**: Install AGIC add-on on an AKS cluster with an existing Application Gateway.
- **Helm Greenfield Deployment**: Install AGIC through Helm, new AKS cluster, and new Application Gateway on blank-slate infrastructure.
- **Helm Brownfield Deployment**: Deploy AGIC through Helm on an existing AKS cluster and Application Gateway.

Monitoring Azure Application Gateway

8/22/2022 • 7 minutes to read • Edit Online

When you have critical applications and business processes relying on Azure resources, you want to monitor those resources for their availability, performance, and operation.

This article describes the monitoring data generated by Azure Application Gateway. Azure Application Gateway uses Azure Monitor. If you are unfamiliar with the features of Azure Monitor common to all Azure services that use it, read Monitoring Azure resources with Azure Monitor.

Monitoring overview page in Azure portal

The **Overview** page in the Azure portal for each Application Gateway includes the following metrics:

- Sum Total Requests
- Sum Failed Requests
- Sum Response Status by HttpStatus
- Sum Throughput
- Sum CurrentConnections
- Avg Healthy Host Count By BackendPool HttpSettings
- Avg Unhealthy Host Count By BackendPool HttpSettings

This is just a subset of the metrics available for Application Gateway. For more information, see Monitoring Azure Application Gateway data reference.

Azure Monitor Network Insights

Some services in Azure have a special focused pre-built monitoring dashboard in the Azure portal that provides a starting point for monitoring your service. These special dashboards are called "insights".

Azure Monitor Network Insights provides a comprehensive view of health and metrics for all deployed network resources (including Application Gateway), without requiring any configuration. For more information, see Azure Monitor Network Insights.

Monitoring data

Azure Application Gateway collects the same kinds of monitoring data as other Azure resources that are described in Monitoring data from Azure resources.

See Monitoring Azure Application Gateway data reference for detailed information on the metrics and logs metrics created by Azure Application Gateway.

Collection and routing

Platform metrics and the Activity log are collected and stored automatically, but can be routed to other locations by using a diagnostic setting.

Resource Logs are not collected and stored until you create a diagnostic setting and route them to one or more locations.

See Create diagnostic setting to collect platform logs and metrics in Azure for the detailed process for creating a

diagnostic setting using the Azure portal, CLI, or PowerShell. When you create a diagnostic setting, you specify which categories of logs to collect. The categories for Azure Application Gateway are listed in Azure Application Gateway monitoring data reference.

The metrics and logs you can collect are discussed in the following sections.

Analyzing metrics

You can analyze metrics for Azure Application Gateway with metrics from other Azure services using metrics explorer by opening **Metrics** from the **Azure Monitor** menu. See Getting started with Azure Metrics Explorer for details on using this tool.

For a list of the platform metrics collected for Azure Application Gateway, see Monitoring Application Gateway data reference metrics.

For reference, you can see a list of all resource metrics supported in Azure Monitor.

Analyzing logs

Data in Azure Monitor Logs is stored in tables where each table has its own set of unique properties.

All resource logs in Azure Monitor have the same fields followed by service-specific fields. The common schema is outlined in Common and service-specific schema for Azure Resource Logs.

The Activity log is a platform login Azure that provides insight into subscription-level events. You can view it independently or route it to Azure Monitor Logs, where you can do much more complex queries using Log Analytics.

For a list of the types of resource logs collected for Azure Application Gateway, see Monitoring Azure Application Gateway data reference.

For a list of the tables used by Azure Monitor Logs and queryable by Log Analytics, see Monitoring Azure Application Gateway data reference.

Sample Kusto queries

> **IMPORTANT**
> When you select **Logs** from the Application Gateway menu, Log Analytics is opened with the query scope set to the current Application Gateway. This means that log queries will only include data from that resource. If you want to run a query that includes data from other Application Gateways or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See Log query scope and time range in Azure Monitor Log Analytics for details.

You can use the following queries to help you monitor your Application Gateway resource.

```
// Requests per hour
// Count of the incoming requests on the Application Gateway.
// To create an alert for this query, click '+ New alert rule'
AzureDiagnostics
| where ResourceType == "APPLICATIONGATEWAYS" and OperationName == "ApplicationGatewayAccess"
| summarize AggregatedValue = count() by bin(TimeGenerated, 1h), _ResourceId
| render timechart
```

```
// Failed requests per hour
// Count of requests to which Application Gateway responded with an error.
// To create an alert for this query, click '+ New alert rule'
AzureDiagnostics
| where ResourceType == "APPLICATIONGATEWAYS" and OperationName == "ApplicationGatewayAccess" and
httpStatus_d > 399
| summarize AggregatedValue = count() by bin(TimeGenerated, 1h), _ResourceId
| render timechart
```

```
// Top 10 Client IPs
// Count of requests per client IP.
AzureDiagnostics
| where ResourceType == "APPLICATIONGATEWAYS" and OperationName == "ApplicationGatewayAccess"
| summarize AggregatedValue = count() by clientIP_s
| top 10 by AggregatedValue
```

```
// Errors by user agent
// Number of errors by user agent.
// To create an alert for this query, click '+ New alert rule'
AzureDiagnostics
| where ResourceType == "APPLICATIONGATEWAYS" and OperationName == "ApplicationGatewayAccess" and
httpStatus_d > 399
| summarize AggregatedValue = count() by userAgent_s, _ResourceId
| sort by AggregatedValue desc
```

Alerts

Azure Monitor alerts proactively notify you when important conditions are found in your monitoring data. They allow you to identify and address issues in your system before your customers notice them. You can set alerts on metrics, logs, and the activity log. Different types of alerts have benefits and drawbacks

If you are creating or running an application which use Application Gateway Azure Monitor Application Insights may offer additional types of alerts.

The following tables list common and recommended alert rules for Application Gateway.

Application Gateway v1

ALERT TYPE	CONDITION	DESCRIPTION
Metric	CPU utilization crosses 80%	Under normal conditions, CPU usage should not regularly exceed 90%, as this may cause latency in the websites hosted behind the Application Gateway and disrupt the client experience.
Metric	Unhealthy host count crosses threshold	Indicates the number of backend servers that Application Gateway is unable to probe successfully. This catches issues where the Application Gateway instances are unable to connect to the backend. Alert if this number goes above 20% of backend capacity.

ALERT TYPE	CONDITION	DESCRIPTION
Metric	Response status (4xx, 5xx) crosses threshold	When Application Gateway response status is 4xx or 5xx. There could be occasional 4xx or 5xx response seen due to transient issues. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.
Metric	Failed requests crosses threshold	When failed requests metric crosses a threshold. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

Application Gateway v2

ALERT TYPE	CONDITION	DESCRIPTION
Metric	Compute Unit utilization crosses 75% of average usage	Compute unit is the measure of compute utilization of your Application Gateway. Check your average compute unit usage in the last one month and set alert if it crosses 75% of it.
Metric	Capacity Unit utilization crosses 75% of peak usage	Capacity units represent overall gateway utilization in terms of throughput, compute, and connection count. Check your maximum capacity unit usage in the last one month and set alert if it crosses 75% of it.
Metric	Unhealthy host count crosses threshold	Indicates number of backend servers that application gateway is unable to probe successfully. This will catch issues where Application gateway instances are unable to connect to the backend. Alert if this number goes above 20% of backend capacity.
Metric	Response status (4xx, 5xx) crosses threshold	When Application Gateway response status is 4xx or 5xx. There could be occasional 4xx or 5xx response seen due to transient issues. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.
Metric	Failed requests crosses threshold	When Failed requests metric crosses threshold. You should observe the gateway in production to determine static threshold or use dynamic threshold for the alert.

ALERT TYPE	CONDITION	DESCRIPTION
Metric	Backend last byte response time crosses threshold	Indicates the time interval between start of establishing a connection to backend server and receiving the last byte of the response body. Create an alert if the backend response latency is more that certain threshold from usual.
Metric	Application Gateway total time crosses threshold	This is the interval from the time when Application Gateway receives the first byte of the HTTP request to the time when the last response byte has been sent to the client. Should create an alert if the backend response latency is more that certain threshold from usual.

Next steps

- See Monitoring Application Gateway data reference for a reference of the metrics, logs, and other important values created by Application Gateway.
- See Monitoring Azure resources with Azure Monitor for details on monitoring Azure resources.

Application Gateway health monitoring overview

8/22/2022 • 6 minutes to read • Edit Online

Azure Application Gateway by default monitors the health of all resources in its back-end pool and automatically removes any resource considered unhealthy from the pool. Application Gateway continues to monitor the unhealthy instances and adds them back to the healthy back-end pool once they become available and respond to health probes. By default, Application gateway sends the health probes with the same port that is defined in the back-end HTTP settings. A custom probe port can be configured using a custom health probe.

The source IP address Application Gateway uses for health probes depends on the backend pool:

- If the server address in the backend pool is a public endpoint, then the source address is the application gateway's frontend public IP address.
- If the server address in the backend pool is a private endpoint, then the source IP address is from the application gateway subnet's private IP address space.

In addition to using default health probe monitoring, you can also customize the health probe to suit your application's requirements. In this article, both default and custom health probes are covered.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Default health probe

An application gateway automatically configures a default health probe when you don't set up any custom probe configuration. The monitoring behavior works by making an HTTP GET request to the IP addresses or FQDN configured in the back-end pool. For default probes if the backend http settings are configured for HTTPS, the probe uses HTTPS to test health of the backend servers.

For example: You configure your application gateway to use back-end servers A, B, and C to receive HTTP network traffic on port 80. The default health monitoring tests the three servers every 30 seconds for a healthy HTTP response with a 30 second timeout for each request. A healthy HTTP response has a status code between 200 and 399. In this case, the HTTP GET request for the health probe will look like `http://127.0.0.1/`.

If the default probe check fails for server A, the application gateway stops forwarding requests to this server. The default probe still continues to check for server A every 30 seconds. When server A responds successfully to one request from a default health probe, application gateway starts forwarding the requests to the server again.

Default health probe settings

PROBE PROPERTY	VALUE	DESCRIPTION
Probe URL	<protocol>://127.0.0.1:<port>/	The protocol and port are inherited from the backend HTTP settings to which the probe is associated
Interval	30	The amount of time in seconds to wait before the next health probe is sent.
Time-out	30	The amount of time in seconds the application gateway waits for a probe response before marking the probe as unhealthy. If a probe returns as healthy, the corresponding backend is immediately marked as healthy.
Unhealthy threshold	3	Governs how many probes to send in case there's a failure of the regular health probe. In v1 SKU, these additional health probes are sent in quick succession to determine the health of the backend quickly and don't wait for the probe interval. In the case of v2 SKU, the health probes wait the interval. The back-end server is marked down after the consecutive probe failure count reaches the unhealthy threshold.

The default probe looks only at <protocol>://127.0.0.1:<port> to determine health status. If you need to configure the health probe to go to a custom URL or modify any other settings, you must use custom probes. For more information about HTTPS probes, see Overview of TLS termination and end to end TLS with Application Gateway.

Probe intervals

All instances of Application Gateway probe the backend independent of each other. The same probe configuration applies to each Application Gateway instance. For example, if the probe configuration is to send health probes every 30 seconds and the application gateway has two instances, then both instances send the health probe every 30 seconds.

Also if there are multiple listeners, then each listener probes the backend independent of each other. For example, if there are two listeners pointing to the same backend pool on two different ports (configured by two backend http settings) then each listener probes the same backend independently. In this case, there are two probes from each application gateway instance for the two listeners. If there are two instances of the application gateway in this scenario, the backend virtual machine would see four probes per the configured probe interval.

Custom health probe

Custom probes allow you to have more granular control over the health monitoring. When using custom probes, you can configure a custom hostname, URL path, probe interval, and how many failed responses to accept before marking the back-end pool instance as unhealthy, etc.

Custom health probe settings

The following table provides definitions for the properties of a custom health probe.

PROBE PROPERTY	DESCRIPTION
Name	Name of the probe. This name is used to identify and refer to the probe in back-end HTTP settings.
Protocol	Protocol used to send the probe. This has to match with the protocol defined in the back-end HTTP settings it is associated to
Host	Host name to send the probe with. In v1 SKU, this value will be used only for the host header of the probe request. In v2 SKU, it will be used both as host header as well as SNI
Path	Relative path of the probe. A valid path starts with '/'
Port	If defined, this is used as the destination port. Otherwise, it uses the same port as the HTTP settings that it is associated to. This property is only available in the v2 SKU
Interval	Probe interval in seconds. This value is the time interval between two consecutive probes
Time-out	Probe time-out in seconds. If a valid response isn't received within this time-out period, the probe is marked as failed
Unhealthy threshold	Probe retry count. The back-end server is marked down after the consecutive probe failure count reaches the unhealthy threshold

Probe matching

By default, an HTTP(S) response with status code between 200 and 399 is considered healthy. Custom health probes additionally support two matching criteria. Matching criteria can be used to optionally modify the default interpretation of what makes a healthy response.

The following are matching criteria:

- **HTTP response status code match** - Probe matching criterion for accepting user specified http response code or response code ranges. Individual comma-separated response status codes or a range of status code is supported.
- **HTTP response body match** - Probe matching criterion that looks at HTTP response body and matches with a user specified string. The match only looks for presence of user specified string in response body and isn't a full regular expression match.

Match criteria can be specified using the `New-AzApplicationGatewayProbeHealthResponseMatch` cmdlet.

For example:

```
$match = New-AzApplicationGatewayProbeHealthResponseMatch -StatusCode 200-399
$match = New-AzApplicationGatewayProbeHealthResponseMatch -Body "Healthy"
```

Once the match criteria is specified, it can be attached to probe configuration using a `-Match` parameter in PowerShell.

NSG considerations

You must allow incoming Internet traffic on TCP ports 65503-65534 for the Application Gateway v1 SKU, and TCP ports 65200-65535 for the v2 SKU with the destination subnet as **Any** and source as **GatewayManager** service tag. This port range is required for Azure infrastructure communication.

Additionally, outbound Internet connectivity can't be blocked, and inbound traffic coming from the **AzureLoadBalancer** tag must be allowed.

For more information, see Application Gateway configuration overview.

Next steps

After learning about Application Gateway health monitoring, you can configure a custom health probe in the Azure portal or a custom health probe using PowerShell and the Azure Resource Manager deployment model.

Back-end health and diagnostic logs for Application Gateway

8/22/2022 • 13 minutes to read • Edit Online

You can monitor Azure Application Gateway resources in the following ways:

- Back-end health: Application Gateway provides the capability to monitor the health of the servers in the back-end pools through the Azure portal and through PowerShell. You can also find the health of the back-end pools through the performance diagnostic logs.
- Logs: Logs allow for performance, access, and other data to be saved or consumed from a resource for monitoring purposes.
- Metrics: Application Gateway has several metrics which help you verify that your system is performing as expected.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Back-end health

Application Gateway provides the capability to monitor the health of individual members of the back-end pools through the portal, PowerShell, and the command-line interface (CLI). You can also find an aggregated health summary of back-end pools through the performance diagnostic logs.

The back-end health report reflects the output of the Application Gateway health probe to the back-end instances. When probing is successful and the back end can receive traffic, it's considered healthy. Otherwise, it's considered unhealthy.

> **IMPORTANT**
> If there is a network security group (NSG) on an Application Gateway subnet, open port ranges 65503-65534 for v1 SKUs, and 65200-65535 for v2 SKUs on the Application Gateway subnet for inbound traffic. This port range is required for Azure infrastructure communication. They are protected (locked down) by Azure certificates. Without proper certificates, external entities, including the customers of those gateways, will not be able to initiate any changes on those endpoints.

View back-end health through the portal

In the portal, back-end health is provided automatically. In an existing application gateway, select **Monitoring** > **Backend health**.

Each member in the back-end pool is listed on this page (whether it's a NIC, IP, or FQDN). Back-end pool name, port, back-end HTTP settings name, and health status are shown. Valid values for health status are **Healthy**, **Unhealthy**, and **Unknown**.

> **NOTE**
>
> If you see a back-end health status of **Unknown**, ensure that access to the back end is not blocked by an NSG rule, a user-defined route (UDR), or a custom DNS in the virtual network.

View back-end health through PowerShell

The following PowerShell code shows how to view back-end health by using the `Get-AzApplicationGatewayBackendHealth` cmdlet:

```
Get-AzApplicationGatewayBackendHealth -Name ApplicationGateway1 -ResourceGroupName Contoso
```

View back-end health through Azure CLI

```
az network application-gateway show-backend-health --resource-group AdatumAppGatewayRG --name
AdatumAppGateway
```

Results

The following snippet shows an example of the response:


```
{
"BackendAddressPool": {
    "Id": "/subscriptions/00000000-0000-0000-
000000000000/resourceGroups/ContosoRG/providers/Microsoft.Network/applicationGateways/applicationGateway1/ba
ckendAddressPools/appGatewayBackendPool"
},
"BackendHttpSettingsCollection": [
    {
    "BackendHttpSettings": {
        "Id": "/00000000-0000-0000-
000000000000/resourceGroups/ContosoRG/providers/Microsoft.Network/applicationGateways/applicationGateway1/ba
ckendHttpSettingsCollection/appGatewayBackendHttpSettings"
    },
    "Servers": [
        {
        "Address": "hostname.westus.cloudapp.azure.com",
        "Health": "Healthy"
        },
        {
        "Address": "hostname.westus.cloudapp.azure.com",
        "Health": "Healthy"
        }
    ]
    }
]
}
```

Diagnostic logs

You can use different types of logs in Azure to manage and troubleshoot application gateways. You can access some of these logs through the portal. All logs can be extracted from Azure Blob storage and viewed in different tools, such as Azure Monitor logs, Excel, and Power BI. You can learn more about the different types of logs from the following list:

- **Activity log**: You can use Azure activity logs (formerly known as operational logs and audit logs) to view all operations that are submitted to your Azure subscription, and their status. Activity log entries are collected by default, and you can view them in the Azure portal.
- **Access log**: You can use this log to view Application Gateway access patterns and analyze important information. This includes the caller's IP, requested URL, response latency, return code, and bytes in and out. An access log is collected every 60 seconds. This log contains one record per instance of Application Gateway. The Application Gateway instance is identified by the instanceId property.
- **Performance log**: You can use this log to view how Application Gateway instances are performing. This log captures performance information for each instance, including total requests served, throughput in bytes, total requests served, failed request count, and healthy and unhealthy back-end instance count. A performance log is collected every 60 seconds. The Performance log is available only for the v1 SKU. For the v2 SKU, use Metrics for performance data.
- **Firewall log**: You can use this log to view the requests that are logged through either detection or prevention mode of an application gateway that is configured with the web application firewall. Firewall logs are collected every 60 seconds.

> **NOTE**
>
> Logs are available only for resources deployed in the Azure Resource Manager deployment model. You cannot use logs for resources in the classic deployment model. For a better understanding of the two models, see the Understanding Resource Manager deployment and classic deployment article.

You have three options for storing your logs:

- **Storage account**: Storage accounts are best used for logs when logs are stored for a longer duration and reviewed when needed.
- **Event hubs**: Event hubs are a great option for integrating with other security information and event management (SIEM) tools to get alerts on your resources.
- **Azure Monitor logs**: Azure Monitor logs is best used for general real-time monitoring of your application or looking at trends.

Enable logging through PowerShell

Activity logging is automatically enabled for every Resource Manager resource. You must enable access and performance logging to start collecting the data available through those logs. To enable logging, use the following steps:

1. Note your storage account's resource ID, where the log data is stored. This value is of the form: /subscriptions/<subscriptionId>/resourceGroups/<resource group name>/providers/Microsoft.Storage/storageAccounts/<storage account name>. You can use any storage account in your subscription. You can use the Azure portal to find this information.

2. Note your application gateway's resource ID for which logging is enabled. This value is of the form: /subscriptions/<subscriptionId>/resourceGroups/<resource group name>/providers/Microsoft.Network/applicationGateways/<application gateway name>. You can use the portal to find this information.

3. Enable diagnostic logging by using the following PowerShell cmdlet:

```
Set-AzDiagnosticSetting  -ResourceId /subscriptions/<subscriptionId>/resourceGroups/<resource group name>/providers/Microsoft.Network/applicationGateways/<application gateway name> -StorageAccountId /subscriptions/<subscriptionId>/resourceGroups/<resource group name>/providers/Microsoft.Storage/storageAccounts/<storage account name> -Enabled $true
```

> **TIP**
>
> Activity logs do not require a separate storage account. The use of storage for access and performance logging incurs service charges.

Enable logging through the Azure portal

1. In the Azure portal, find your resource and select **Diagnostic settings**.

   For Application Gateway, three logs are available:

   - Access log
   - Performance log
   - Firewall log

2. To start collecting data, select **Turn on diagnostics**.

3. The **Diagnostics settings** page provides the settings for the diagnostic logs. In this example, Log Analytics stores the logs. You can also use event hubs and a storage account to save the diagnostic logs.

4. Type a name for the settings, confirm the settings, and select **Save**.

Activity log

Azure generates the activity log by default. The logs are preserved for 90 days in the Azure event logs store. Learn more about these logs by reading the View events and activity log article.

Access log

The access log is generated only if you've enabled it on each Application Gateway instance, as detailed in the preceding steps. The data is stored in the storage account that you specified when you enabled the logging. Each access of Application Gateway is logged in JSON format as shown below.

For Application Gateway Standard and WAF SKU (v1)

VALUE	DESCRIPTION
instanceId	Application Gateway instance that served the request.
clientIP	Originating IP for the request.
clientPort	Originating port for the request.
httpMethod	HTTP method used by the request.
requestUri	URI of the received request.
RequestQuery	**Server-Routed**: Back-end pool instance that was sent the request. **X-AzureApplicationGateway-LOG-ID**: Correlation ID used for the request. It can be used to troubleshoot traffic issues on the back-end servers. **SERVER-STATUS**: HTTP response code that Application Gateway received from the back end.
UserAgent	User agent from the HTTP request header.
httpStatus	HTTP status code returned to the client from Application Gateway.
httpVersion	HTTP version of the request.
receivedBytes	Size of packet received, in bytes.
sentBytes	Size of packet sent, in bytes.
timeTaken	Length of time (in milliseconds) that it takes for a request to be processed and its response to be sent. This is calculated as the interval from the time when Application Gateway receives the first byte of an HTTP request to the time when the response send operation finishes. It's important to note that the Time-Taken field usually includes the time that the request and response packets are traveling over the network.
sslEnabled	Whether communication to the back-end pools used TLS/SSL. Valid values are on and off.
host	The hostname with which the request has been sent to the backend server. If backend hostname is being overridden, this name will reflect that.
originalHost	The hostname with which the request was received by the Application Gateway from the client.

```
{
    "resourceId":
"/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/PEERINGTEST/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONGATEWAYS/
{applicationGatewayName}",
    "operationName": "ApplicationGatewayAccess",
    "time": "2017-04-26T19:27:38Z",
    "category": "ApplicationGatewayAccessLog",
    "properties": {
        "instanceId": "ApplicationGatewayRole_IN_0",
        "clientIP": "191.96.249.97",
        "clientPort": 46886,
        "httpMethod": "GET",
        "requestUri": "/phpmyadmin/scripts/setup.php",
        "requestQuery": "X-AzureApplicationGateway-CACHE-HIT=0&SERVER-ROUTED=10.4.0.4&X-
AzureApplicationGateway-LOG-ID=874f1f0f-6807-41c9-b7bc-f3cfa74aa0b1&SERVER-STATUS=404",
        "userAgent": "-",
        "httpStatus": 404,
        "httpVersion": "HTTP/1.0",
        "receivedBytes": 65,
        "sentBytes": 553,
        "timeTaken": 205,
        "sslEnabled": "off",
        "host": "www.contoso.com",
        "originalHost": "www.contoso.com"
    }
}
```

For Application Gateway and WAF v2 SKU

VALUE	DESCRIPTION
instanceId	Application Gateway instance that served the request.
clientIP	Originating IP for the request.
httpMethod	HTTP method used by the request.
requestUri	URI of the received request.
UserAgent	User agent from the HTTP request header.
httpStatus	HTTP status code returned to the client from Application Gateway.
httpVersion	HTTP version of the request.
receivedBytes	Size of packet received, in bytes.
sentBytes	Size of packet sent, in bytes.
clientResponseTime	Length of time (in **seconds**) that it takes for the first byte of a client request to be processed and the first byte sent in the response to the client.
timeTaken	Length of time (in **seconds**) that it takes for the first byte of a client request to be processed and its last-byte sent in the response to the client. It's important to note that the Time-Taken field usually includes the time that the request and response packets are traveling over the network.

VALUE	DESCRIPTION
WAFEvaluationTime	Length of time (in **seconds**) that it takes for the request to be processed by the WAF.
WAFMode	Value can be either Detection or Prevention
transactionId	Unique identifier to correlate the request received from the client
sslEnabled	Whether communication to the back-end pools used TLS. Valid values are on and off.
sslCipher	Cipher suite being used for TLS communication (if TLS is enabled).
sslProtocol	SSL/TLS protocol being used (if TLS is enabled).
serverRouted	The backend server that application gateway routes the request to.
serverStatus	HTTP status code of the backend server.
serverResponseLatency	Latency of the response (in **seconds**) from the backend server.
host	Address listed in the host header of the request. If rewritten using header rewrite, this field contains the updated host name
originalRequestUriWithArgs	This field contains the original request URL
requestUri	This field contains the URL after the rewrite operation on Application Gateway
originalHost	This field contains the original request host name

```
{
    "timeStamp": "2021-10-14T22:17:11+00:00",
    "resourceId":
"/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/{resourceGroupName}/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONG
ATEWAYS/{applicationGatewayName}",
    "listenerName": "HTTP-Listener",
    "ruleName": "Storage-Static-Rule",
    "backendPoolName": "StaticStorageAccount",
    "backendSettingName": "StorageStatic-HTTPS-Setting",
    "operationName": "ApplicationGatewayAccess",
    "category": "ApplicationGatewayAccessLog",
    "properties": {
        "instanceId": "appgw_2",
        "clientIP": "185.42.129.24",
        "clientPort": 45057,
        "httpMethod": "GET",
        "originalRequestUriWithArgs": "\/",
        "requestUri": "\/",
        "requestQuery": "",
        "userAgent": "Mozilla\/5.0 (Windows NT 6.1; WOW64) AppleWebKit\/537.36 (KHTML, like Gecko)
Chrome\/52.0.2743.116 Safari\/537.36",
        "httpStatus": 200,
        "httpVersion": "HTTP\/1.1",
        "receivedBytes": 184,
        "sentBytes": 466,
        "clientResponseTime": 0,
        "timeTaken": 0.034,
        "WAFEvaluationTime": "0.000",
        "WAFMode": "Detection",
        "transactionId": "592d1649f75a8d480a3c4dc6a975309d",
        "sslEnabled": "on",
        "sslCipher": "ECDHE-RSA-AES256-GCM-SHA384",
        "sslProtocol": "TLSv1.2",
        "sslClientVerify": "NONE",
        "sslClientCertificateFingerprint": "",
        "sslClientCertificateIssuerName": "",
        "serverRouted": "52.239.221.65:443",
        "serverStatus": "200",
        "serverResponseLatency": "0.028",
        "originalHost": "20.110.30.194",
        "host": "20.110.30.194"
    }
}
```


Performance log

The performance log is generated only if you have enabled it on each Application Gateway instance, as detailed in the preceding steps. The data is stored in the storage account that you specified when you enabled the logging. The performance log data is generated in 1-minute intervals. It is available only for the v1 SKU. For the v2 SKU, use Metrics for performance data. The following data is logged:

VALUE	DESCRIPTION
instanceId	Application Gateway instance for which performance data is being generated. For a multiple-instance application gateway, there is one row per instance.
healthyHostCount	Number of healthy hosts in the back-end pool.
unHealthyHostCount	Number of unhealthy hosts in the back-end pool.
requestCount	Number of requests served.

VALUE	DESCRIPTION
latency	Average latency (in milliseconds) of requests from the instance to the back end that serves the requests.
failedRequestCount	Number of failed requests.
throughput	Average throughput since the last log, measured in bytes per second.

```
{
    "resourceId":
"/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/{resourceGroupName}/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONG
ATEWAYS/{applicationGatewayName}",
    "operationName": "ApplicationGatewayPerformance",
    "time": "2016-04-09T00:00:00Z",
    "category": "ApplicationGatewayPerformanceLog",
    "properties":
    {
        "instanceId":"ApplicationGatewayRole_IN_1",
        "healthyHostCount":"4",
        "unHealthyHostCount":"0",
        "requestCount":"185",
        "latency":"0",
        "failedRequestCount":"0",
        "throughput":"119427"
    }
}
```

> **NOTE**
> Latency is calculated from the time when the first byte of the HTTP request is received to the time when the last byte of the HTTP response is sent. It's the sum of the Application Gateway processing time plus the network cost to the back end, plus the time that the back end takes to process the request.

Firewall log

The firewall log is generated only if you have enabled it for each application gateway, as detailed in the preceding steps. This log also requires that the web application firewall is configured on an application gateway. The data is stored in the storage account that you specified when you enabled the logging. The following data is logged:

VALUE	DESCRIPTION
instanceId	Application Gateway instance for which firewall data is being generated. For a multiple-instance application gateway, there is one row per instance.
clientIp	Originating IP for the request.
clientPort	Originating port for the request.
requestUri	URL of the received request.
ruleSetType	Rule set type. The available value is OWASP.

VALUE	DESCRIPTION
ruleSetVersion	Rule set version used. Available values are 2.2.9 and 3.0.
ruleId	Rule ID of the triggering event.
message	User-friendly message for the triggering event. More details are provided in the details section.
action	Action taken on the request. Available values are Blocked and Allowed (for custom rules), Matched (when a rule matches a part of the request), and Detected and Blocked (these are both for mandatory rules, depending on if the WAF is in detection or prevention mode).
site	Site for which the log was generated. Currently, only Global is listed because rules are global.
details	Details of the triggering event.
details.message	Description of the rule.
details.data	Specific data found in request that matched the rule.
details.file	Configuration file that contained the rule.
details.line	Line number in the configuration file that triggered the event.
hostname	Hostname or IP address of the Application Gateway.
transactionId	Unique ID for a given transaction which helps group multiple rule violations that occurred within the same request.

```
{
    "timeStamp": "2021-10-14T22:17:11+00:00",
    "resourceId":
"/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/{resourceGroupName}/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONG
ATEWAYS/{applicationGatewayName}",
    "operationName": "ApplicationGatewayFirewall",
    "category": "ApplicationGatewayFirewallLog",
    "properties": {
        "instanceId": "appgw_2",
        "clientIp": "185.42.129.24",
        "clientPort": "",
        "requestUri": "\/",
        "ruleSetType": "OWASP_CRS",
        "ruleSetVersion": "3.0.0",
        "ruleId": "920350",
        "message": "Host header is a numeric IP address",
        "action": "Matched",
        "site": "Global",
        "details": {
            "message": "Warning. Pattern match \\\"^[\\\\d.:]+$\\\" at REQUEST_HEADERS:Host .... ",
            "data": "20.110.30.194:80",
            "file": "rules\/REQUEST-920-PROTOCOL-ENFORCEMENT.conf",
            "line": "791"
        },
        "hostname": "20.110.30.194:80",
        "transactionId": "592d1649f75a8d480a3c4dc6a975309d",
        "policyId": "default",
        "policyScope": "Global",
        "policyScopeName": "Global"
    }
}
```


View and analyze the activity log

You can view and analyze activity log data by using any of the following methods:

- **Azure tools**: Retrieve information from the activity log through Azure PowerShell, the Azure CLI, the Azure REST API, or the Azure portal. Step-by-step instructions for each method are detailed in the Activity operations with Resource Manager article.
- **Power BI**: If you don't already have a Power BI account, you can try it for free. By using the Power BI template apps, you can analyze your data.

View and analyze the access, performance, and firewall logs

Azure Monitor logs can collect the counter and event log files from your Blob storage account. It includes visualizations and powerful search capabilities to analyze your logs.

You can also connect to your storage account and retrieve the JSON log entries for access and performance logs. After you download the JSON files, you can convert them to CSV and view them in Excel, Power BI, or any other data-visualization tool.

> **TIP**
>
> If you are familiar with Visual Studio and basic concepts of changing values for constants and variables in C#, you can use the log converter tools available from GitHub.

Analyzing Access logs through GoAccess

We have published a Resource Manager template that installs and runs the popular GoAccess log analyzer for Application Gateway Access Logs. GoAccess provides valuable HTTP traffic statistics such as Unique Visitors, Requested Files, Hosts, Operating Systems, Browsers, HTTP Status codes and more. For more details, please see the Readme file in the Resource Manager template folder in GitHub.

Next steps

- Visualize counter and event logs by using Azure Monitor logs.
- Visualize your Azure activity log with Power BI blog post.
- View and analyze Azure activity logs in Power BI and more blog post.

Metrics for Application Gateway

8/22/2022 • 9 minutes to read • Edit Online

Application Gateway publishes data points, called metrics, to Azure Monitor for the performance of your Application Gateway and backend instances. These metrics are numerical values in an ordered set of time-series data that describe some aspect of your application gateway at a particular time. If there are requests flowing through the Application Gateway, it measures and sends its metrics in 60-second intervals. If there are no requests flowing through the Application Gateway or no data for a metric, the metric is not reported. For more information, see Azure Monitor metrics.

Metrics supported by Application Gateway V2 SKU

Timing metrics

Application Gateway provides several built-in timing metrics related to the request and response, which are all measured in milliseconds.

> **NOTE**
>
> If there are more than one listener in the Application Gateway, then always filter by *Listener* dimension while comparing different latency metrics in order to get meaningful inference.

- **Backend connect time**

  Time spent establishing a connection with the backend application.

This includes the network latency as well as the time taken by the backend server's TCP stack to establish new connections. In case of TLS, it also includes the time spent on handshake.

- **Backend first byte response time**

  Time interval between start of establishing a connection to backend server and receiving the first byte of the response header.

  This approximates the sum of *Backend connect time*, time taken by the request to reach the backend from Application Gateway, time taken by backend application to respond (the time the server took to generate content, potentially fetch database queries), and the time taken by first byte of the response to reach the Application Gateway from the backend.

- **Backend last byte response time**

  Time interval between start of establishing a connection to backend server and receiving the last byte of the response body.

  This approximates the sum of *Backend first byte response time* and data transfer time (this number may vary greatly depending on the size of objects requested and the latency of the server network).

- **Application gateway total time**

  Average time that it takes for a request to be received, processed and its response to be sent.

  This is the interval from the time when Application Gateway receives the first byte of the HTTP request to the time when the last response byte has been sent to the client. This includes the processing time taken by Application Gateway, the *Backend last byte response time*, and the time taken by Application Gateway to send all the response.

- **Client RTT**

  Average round trip time between clients and Application Gateway.

These metrics can be used to determine whether the observed slowdown is due to the client network, Application Gateway performance, the backend network and backend server TCP stack saturation, backend application performance, or large file size.

For example, If there's a spike in *Backend first byte response time* trend but the *Backend connect time* trend is stable, then it can be inferred that the Application gateway to backend latency and the time taken to establish the connection is stable, and the spike is caused due to an increase in the response time of backend application. On the other hand, if the spike in *Backend first byte response time* is associated with a corresponding spike in *Backend connect time*, then it can be deduced that either the network between Application Gateway and backend server or the backend server TCP stack has saturated.

If you notice a spike in *Backend last byte response time* but the *Backend first byte response time* is stable, then it can be deduced that the spike is because of a larger file being requested.

Similarly, if the *Application gateway total time* has a spike but the *Backend last byte response time* is stable, then it can either be a sign of performance bottleneck at the Application Gateway or a bottleneck in the network between client and Application Gateway. Additionally, if the *client RTT* also has a corresponding spike, then it indicates that that the degradation is because of the network between client and Application Gateway.

Application Gateway metrics

For Application Gateway, the following metrics are available:

- **Bytes received**

  Count of bytes received by the Application Gateway from the clients

- **Bytes sent**

  Count of bytes sent by the Application Gateway to the clients

- **Client TLS protocol**

  Count of TLS and non-TLS requests initiated by the client that established connection with the Application Gateway. To view TLS protocol distribution, filter by the dimension TLS Protocol.

- **Current capacity units**

  Count of capacity units consumed to load balance the traffic. There are three determinants to capacity unit - compute unit, persistent connections and throughput. Each capacity unit is composed of at most: 1 compute unit, or 2500 persistent connections, or 2.22-Mbps throughput.

- **Current compute units**

  Count of processor capacity consumed. Factors affecting compute unit are TLS connections/sec, URL Rewrite computations, and WAF rule processing.

- **Current connections**

  The total number of concurrent connections active from clients to the Application Gateway

- **Estimated Billed Capacity units**

  With the v2 SKU, the pricing model is driven by consumption. Capacity units measure consumption-based cost that is charged in addition to the fixed cost. *Estimated Billed Capacity units* indicates the number of capacity units using which the billing is estimated. This is calculated as the greater value between *Current capacity units* (capacity units required to load balance the traffic) and *Fixed billable capacity units* (minimum capacity units kept provisioned).

- **Failed Requests**

  Number of requests that Application Gateway has served with 5xx server error codes. This includes the 5xx codes that are generated from the Application Gateway as well as the 5xx codes that are generated from the backend. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

- **Fixed Billable Capacity Units**

  The minimum number of capacity units kept provisioned as per the *Minimum scale units* setting (one instance translates to 10 capacity units) in the Application Gateway configuration.

- **New connections per second**

  The average number of new TCP connections per second established from clients to the Application Gateway and from the Application Gateway to the backend members.

- **Response Status**

  HTTP response status returned by Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.

- **Throughput**

  Number of bytes per second the Application Gateway has served

- **Total Requests**

  Count of successful requests that Application Gateway has served. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

Backend metrics

For Application Gateway, the following metrics are available:

- **Backend response status**

  Count of HTTP response status codes returned by the backends. This does not include any response codes generated by the Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.

- **Healthy host count**

  The number of backends that are determined healthy by the health probe. You can filter on a per backend pool basis to show the number of healthy hosts in a specific backend pool.

- **Unhealthy host count**

  The number of backends that are determined unhealthy by the health probe. You can filter on a per backend pool basis to show the number of unhealthy hosts in a specific backend pool.

- **Requests per minute per Healthy Host**

  The average number of requests received by each healthy member in a backend pool in a minute. You must specify the backend pool using the *BackendPool HttpSettings* dimension.

Web Application Firewall (WAF) metrics

For information on WAF Monitoring, see WAF v2 Metrics

Metrics supported by Application Gateway V1 SKU

Application Gateway metrics

For Application Gateway, the following metrics are available:

- **CPU Utilization**

  Displays the utilization of the CPUs allocated to the Application Gateway. Under normal conditions, CPU usage should not regularly exceed 90%, as this may cause latency in the websites hosted behind the Application Gateway and disrupt the client experience. You can indirectly control or improve CPU utilization by modifying the configuration of the Application Gateway by increasing the instance count or by moving to a larger SKU size, or doing both.

- **Current connections**

  Count of current connections established with Application Gateway

- **Failed Requests**

  Number of requests that failed due to connection issues. This count includes requests that failed due to exceeding the "Request time-out" HTTP setting and requests that failed due to connection issues between Application gateway and backend. This count does not include failures due to no healthy backend being available. 4xx and 5xx responses from the backend are also not considered as part of this metric.

- **Response Status**

  HTTP response status returned by Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.

- **Throughput**

  Number of bytes per second the Application Gateway has served

- **Total Requests**

  Count of successful requests that Application Gateway has served. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

Backend metrics

For Application Gateway, the following metrics are available:

- **Healthy host count**

  The number of backends that are determined healthy by the health probe. You can filter on a per backend pool basis to show the number of healthy hosts in a specific backend pool.

- **Unhealthy host count**

  The number of backends that are determined unhealthy by the health probe. You can filter on a per backend pool basis to show the number of unhealthy hosts in a specific backend pool.

Web Application Firewall (WAF) metrics

For information on WAF Monitoring, see WAF v1 Metrics

Metrics visualization

Browse to an application gateway, under **Monitoring** select **Metrics**. To view the available values, select the **METRIC** drop-down list.

In the following image, you see an example with three metrics displayed for the last 30 minutes:

To see a current list of metrics, see Supported metrics with Azure Monitor.

Alert rules on metrics

You can start alert rules based on metrics for a resource. For example, an alert can call a webhook or email an administrator if the throughput of the application gateway is above, below, or at a threshold for a specified period.

The following example walks you through creating an alert rule that sends an email to an administrator after throughput breaches a threshold:

1. select **Add metric alert** to open the **Add rule** page. You can also reach this page from the metrics page.

2. On the **Add rule** page, fill out the name, condition, and notify sections, and select **OK**.

   - In the **Condition** selector, select one of the four values: **Greater than**, **Greater than or equal**, **Less than**, or **Less than or equal to**.
   - In the **Period** selector, select a period from five minutes to six hours.
   - If you select **Email owners, contributors, and readers**, the email can be dynamic based on the users who have access to that resource. Otherwise, you can provide a comma-separated list of users in the **Additional administrator email(s)** box.

If the threshold is breached, an email that's similar to the one in the following image arrives:

A list of alerts appears after you create a metric alert. It provides an overview of all the alert rules.

To learn more about alert notifications, see Receive alert notifications.

To understand more about webhooks and how you can use them with alerts, visit Configure a webhook on an Azure metric alert.

Next steps

- Visualize counter and event logs by using Azure Monitor logs.
- Visualize your Azure activity log with Power BI blog post.
- View and analyze Azure activity logs in Power BI and more blog post.

Overview of WebSocket support in Application Gateway

8/22/2022 • 3 minutes to read • Edit Online

Application Gateway provides native support for WebSocket across all gateway sizes. There is no user-configurable setting to selectively enable or disable WebSocket support.

WebSocket protocol standardized in RFC6455 enables a full duplex communication between a server and a client over a long running TCP connection. This feature allows for a more interactive communication between the web server and the client, which can be bidirectional without the need for polling as required in HTTP-based implementations. WebSocket has low overhead unlike HTTP and can reuse the same TCP connection for multiple request/responses resulting in a more efficient utilization of resources. WebSocket protocols are designed to work over traditional HTTP ports of 80 and 443.

You can continue using a standard HTTP listener on port 80 or 443 to receive WebSocket traffic. WebSocket traffic is then directed to the WebSocket enabled backend server using the appropriate backend pool as specified in application gateway rules. The backend server must respond to the application gateway probes, which are described in the health probe overview section. Application gateway health probes are HTTP/HTTPS only. Each backend server must respond to HTTP probes for application gateway to route WebSocket traffic to the server.

It's used in apps that benefit from fast, real-time communication, such as chat, dashboard, and game apps.

How does WebSocket work

To establish a WebSocket connection, a specific HTTP-based handshake is exchanged between the client and the server. If successful, the application-layer protocol is "upgraded" from HTTP to WebSockets, using the previously established TCP connection. Once this occurs, HTTP is completely out of the picture; data can be sent or received using the WebSocket protocol by both endpoints, until the WebSocket connection is closed.

NOTE

As described, the HTTP protocol is used only to perform a handshake when establishing a WebSocket connection. Once the handshake is completed, a WebSocket connection gets opened for transmitting the data, and the Web Application Firewall (WAF) cannot parse any contents. Therefore, WAF does not perform any inspections on such data.

Listener configuration element

An existing HTTP listener can be used to support WebSocket traffic. The following is a snippet of an httpListeners element from a sample template file. You would need both HTTP and HTTPS listeners to support WebSocket and secure WebSocket traffic. Similarly you can use the portal or Azure PowerShell to create an application gateway with listeners on port 80/443 to support WebSocket traffic.

```
"httpListeners": [
        {
            "name": "appGatewayHttpsListener",
            "properties": {
                "FrontendIPConfiguration": {
                    "Id":
"/subscriptions/{subscriptionId/resourceGroups/{resourceGroupName/providers/Microsoft.Network/applicationGat
eways/{applicationGatewayName/frontendIPConfigurations/DefaultFrontendPublicIP"
                },
                "FrontendPort": {
                    "Id":
"/subscriptions/{subscriptionId/resourceGroups/{resourceGroupName/providers/Microsoft.Network/applicationGat
eways/{applicationGatewayName/frontendPorts/appGatewayFrontendPort443'"
                },
                "Protocol": "Https",
                "SslCertificate": {
                    "Id":
"/subscriptions/{subscriptionId/resourceGroups/{resourceGroupName/providers/Microsoft.Network/applicationGat
eways/{applicationGatewayName/sslCertificates/appGatewaySslCert1'"
                },
            }
        },
        {
            "name": "appGatewayHttpListener",
            "properties": {
                "FrontendIPConfiguration": {
                    "Id":
"/subscriptions/{subscriptionId/resourceGroups/{resourceGroupName/providers/Microsoft.Network/applicationGat
eways/{applicationGatewayName/frontendIPConfigurations/appGatewayFrontendIP'"
                },
                "FrontendPort": {
                    "Id":
"/subscriptions/{subscriptionId/resourceGroups/{resourceGroupName/providers/Microsoft.Network/applicationGat
eways/{applicationGatewayName/frontendPorts/appGatewayFrontendPort80'"
                },
                "Protocol": "Http",
            }
        }
    ],
```

BackendAddressPool, BackendHttpSetting, and Routing rule configuration

A BackendAddressPool is used to define a backend pool with WebSocket enabled servers. The backendHttpSetting is defined with a backend port 80 and 443. The request timeout value in HTTP Settings also applies to the WebSocket session. There is no change required in the routing rule, which is used to tie the appropriate listener to the corresponding backend address pool.

```
"requestRoutingRules": [{
    "name": "<ruleName1>",
    "properties": {
        "RuleType": "Basic",
        "httpListener": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/httpListeners/appGatewayHttpsListener')]"
        },
        "backendAddressPool": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/backendAddressPools/ContosoServerPool')]"
        },
        "backendHttpSettings": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/backendHttpSettingsCollection/appGatewayBackendHttpSettings')]"
        }
    }

}, {
    "name": "<ruleName2>",
    "properties": {
        "RuleType": "Basic",
        "httpListener": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/httpListeners/appGatewayHttpListener')]"
        },
        "backendAddressPool": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/backendAddressPools/ContosoServerPool')]"
        },
        "backendHttpSettings": {
            "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationG
ateways/{applicationGatewayName}/backendHttpSettingsCollection/appGatewayBackendHttpSettings')]"
        }

    }
}]
```

WebSocket enabled backend

Your backend must have a HTTP/HTTPS web server running on the configured port (usually 80/443) for WebSocket to work. This requirement is because WebSocket protocol requires the initial handshake to be HTTP with upgrade to WebSocket protocol as a header field. The following is an example of a header:

```
    GET /chat HTTP/1.1
    Host: server.example.com
    Upgrade: websocket
    Connection: Upgrade
    Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
    Origin: https://example.com
    Sec-WebSocket-Protocol: chat, superchat
    Sec-WebSocket-Version: 13
```

Another reason for this is that application gateway backend health probe supports HTTP and HTTPS protocols only. If the backend server does not respond to HTTP or HTTPS probes, it is taken out of backend pool.

Next steps

After learning about WebSocket support, go to create an application gateway to get started with a WebSocket enabled web application.

Understanding Pricing for Azure Application Gateway and Web Application Firewall

8/22/2022 • 14 minutes to read • Edit Online

> **NOTE**
> Prices shown in this article are examples and are for illustration purposes only. For pricing information according to your region, see the Pricing page.

Azure Application Gateway is a layer 7 load-balancing solution, which enables scalable, highly available, and secure web application delivery on Azure.

There are no upfront costs or termination costs associated with Application Gateway. You will be billed only for the resources pre-provisioned and utilized based on actual hourly consumption. Costs associated with Application Gateway are classified into two components: fixed costs and variable costs. Actual costs within each component will vary according to the SKU being utilized.

This article describes the costs associated with each SKU and it is recommended that users utilize this document for planning and managing costs associated with the Azure Application Gateway.

V2 SKUs

Application Gateway V2 and WAF V2 SKUs support autoscaling and guarantee high availability by default.

Key Terms

Capacity Unit

Capacity Unit is the measure of capacity utilization for an Application Gateway across multiple parameters.

A single Capacity Unit consists of the following parameters:

- 2500 Persistent connections
- 2.22-Mbps throughput
- 1 Compute Unit

If any one of these parameters are exceeded, then another n capacity unit(s) are necessary, even if the other two parameters don't exceed this single capacity unit's limits. The parameter with the highest utilization among the three above will be internally used for calculating capacity units, which is in turn billed.

Compute Unit

Compute Unit is the measure of compute capacity consumed. Factors affecting compute unit consumption are TLS connections/sec, URL Rewrite computations, and WAF rule processing. The number of requests a compute unit can handle depends on various criteria like TLS certificate key size, key exchange algorithm, header rewrites, and in case of WAF - incoming request size.

Compute unit guidance:

- Standard_v2 - Each compute unit is capable of approximately 50 connections per second with RSA 2048-bit key TLS certificate.
- WAF_v2 - Each compute unit can support approximately 10 concurrent requests per second for 70-30% mix of traffic with 70% requests less than 2 KB GET/POST and remaining higher. WAF performance is not affected by response size currently.

Instance Count

Pre-provisioning of resources for Application Gateway V2 SKUs is defined in terms of instance count. Each instance guarantees a minimum of 10 capacity units in terms of processing capability. The same instance could potentially support more than 10 capacity units for different traffic patterns depending upon the Capacity Unit parameters.

Manually defined scale and limits set for autoscaling (minimum or maximum) are set in terms of Instance Count. The manually set scale for instance count and the minimum instance count in autoscale config will reserve 10 capacity units/instance. These reserved CUs will be billed as long as the Application Gateway is active regardless of the actual resource consumption. If actual consumption crosses the 10 capacity units/instance threshold, additional capacity units will be billed under the variable component.

V2 SKUs are billed based on the consumption and constitute of two parts:

- Fixed Costs

  Cost based on the time the Application Gateway V2 /WAF V2 is provisioned and available for processing requests. This ensures high availability - even if 0 instances are reserved by specifying 0 in minimum instance count as part of autoscaling.

  The fixed cost also includes the cost associated with the public IP attached to the Application Gateway.

  The number of instances running at any point of time is not considered as a factor for fixed costs for V2 SKUs. The fixed costs of running a Standard_V2 (or WAF_V2) would be same per hour regardless of the number of instances running within the same Azure region.

- Capacity Unit Costs

  Cost based on the number of capacity units either reserved or utilized additionally as required for processing the incoming requests. Consumption based costs are computed hourly.

Total Costs = Fixed Costs + Capacity Unit Costs

> **NOTE**
>
> A partial hour is billed as a full hour.

The following table shows example prices based on a snapshot of East US pricing and are for illustration purposes only.

Fixed Costs (East US region pricing)

V2 SKU	COSTS ($/HR)
Standard_V2	$0.246
WAF_V2	$0.443

Monthly price estimates are based on 730 hours of usage per month.

Variable Costs (East US region pricing)

CAPACITY UNIT	STANDARD_V2 ($/HR)	WAF_V2 ($/HR)
1 CU	$0.008	$0.0144

For more pricing information according to your region, see the pricing page.

> **NOTE**
> Outbound data transfers - data going out of Azure data centers from application gateways will be charged at standard data transfer rates.

Example 1 (a) – Manual Scaling

Let's assume you've provisioned a Standard_V2 Application Gateway with manual scaling set to 8 instances for the entire month. During this time, it receives an average of 88.8-Mbps data transfer.

Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

1 CU can handle 2.22-Mbps throughput.

CUs required to handle 88.8 Mbps = 88.8 / 2.22 = 40 CUs

Pre-provisioned CUs = 8 (Instance count) * 10 = 80

Since 80 (reserved capacity) > 40 (required capacity), no additional CUs are required.

Fixed Price = $0.246 * 730 (Hours) = $179.58

Variable Costs = $0.008 * 8 (Instance Units) * 10(capacity units) * 730 (Hours) = $467.2

Total Costs = $179.58 + $467.2 = $646.78

Example 1 (b) – Manual Scaling With traffic going beyond provisioned capacity

Let's assume you've provisioned a Standard_V2 Application Gateway with manual scaling set to 3 instances for the entire month. During this time, it receives an average of 88.8-Mbps data transfer.

Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

1 CU can handle 2.22 Mbps throughput.

CUs required to handle 88.8 Mbps = 88.8 / 2.22 = 40

Pre-provisioned CUs = 3 (Instance count) * 10 = 30

Since 40 (required capacity) > 30 (reserved capacity), additional CUs are required. The number of additional CUs utilized would depend on the free capacity available with each instance.

If processing capacity equivalent to 10 additional CUs was available for use within the 3 reserved instances.

Fixed Price = $0.246 * 730 (Hours) = $179.58

Variable Costs = $0.008 * (3(Instance Units) * 10(capacity units) + 10 (additional capacity units)) * 730 (Hours) = $233.6

Total Costs = $179.58 + $233.6 = $413.18

However, if processing capacity equivalent to only say 7 additional CUs was available for use within the 3 reserved instances. In this scenario the Application Gateway resource is under scaled and could potentially lead to increase in latency or requests getting dropped.

Fixed Price = $0.246 * 730 (Hours) = $179.58

Variable Costs = $0.008 * (3(Instance Units) * 10(capacity units) + 7 (additional capacity units)) * 730 (Hours) = $216.08

Total Costs = $179.58 + $216.08 = $395.66

> **NOTE**
> In case of Manual Scaling, any additional requests exceeding the maximum processing capacity of the reserved instances may cause impact to the availability of your application. In situations of high load, reserved instances may be able to provide more than 10 Capacity units of processing capacity depending upon the configuration and type of incoming requests. But it is recommended to provision the number of instances as per your traffic requirements.

Example 2 – WAF_V2 instance with Autoscaling

Let's assume you've provisioned a WAF_V2 with autoscaling enabled and set the minimum instance count to 6 for the entire month. The request load has caused the WAF instance to scale out and utilize 65 Capacity units(scale out of 5 capacity units, while 60 units were reserved) for the entire month. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Monthly price estimates are based on 730 hours of usage per month.

Fixed Price = $0.443 * 730 (Hours) = $323.39

Variable Costs = $0.0144 * 65(capacity units) * 730 (Hours) = $683.28

Total Costs = $323.39 + $683.28 = $1006.67

> **NOTE**
> Actual Traffic observed for your Application Gateway is unlikely to have such a constant pattern of traffic and the observed load on your Application Gateway would fluctuate according to actual usage.

Example 3 (a) – WAF_V2 instance with Autoscaling and 0 Min scale config

Let's assume you've provisioned a WAF_V2 with autoscaling enabled and set the minimum instance count as 0 for the entire month. The request load on the WAF is minimum but consistently present per hour for the entire month. The load is below the capacity of a single capacity unit. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Monthly price estimates are based on 730 hours of usage per month.

Fixed Price = $0.443 * 730 (Hours) = $323.39

Variable Costs = $0.0144 * 1(capacity units) * 730 (Hours) = $10.512

Total Costs = $323.39 + $10.512 = $333.902

Example 3 (b) – WAF_V2 instance with Autoscaling with 0 Min instance count

Let's assume you've provisioned a WAF_V2 with autoscaling enabled and set the minimum instance count to 0 for the entire month. However, there is 0 traffic directed to the WAF instance for the entire month. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Fixed Price = $0.443 * 730 (Hours) = $323.39

Variable Costs = $0.0144 * 0(capacity units) * 730 (Hours) = $0

Total Costs = $323.39 + $0 = $323.39

Example 3 (C) – WAF_V2 instance with manual scaling set to 1 instance

Let's assume you've provisioned a WAF_V2 and set it to manual scaling with the minimum acceptable value of 1 instance for the entire month. However, there is 0 traffic directed to the WAF for the entire month. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Monthly price estimates are based on 730 hours of usage per month.

Fixed Price = $0.443 * 730 (Hours) = $323.39

Variable Costs = $0.0144 * 1(Instance count) * 10(capacity units) * 730 (Hours) = $105.12

Total Costs = $323.39 + $105.12 = $428.51

Example 4 – WAF_V2 with Autoscaling, capacity unit calculations

Let's assume you've provisioned a WAF_V2 with autoscaling enabled and set the minimum instance count to 0 for the entire month. During this time, it receives 25 new TLS connections/sec with an average of 8.88-Mbps data transfer. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Monthly price estimates are based on 730 hours of usage per month.

Fixed Price = $0.443 * 730 (Hours) = $323.39

Variable Costs = $0.0144 * 730 (Hours) * {Max (25/50, 8.88/2.22)} = $42.048 (4 Capacity units required to handle 8.88 Mbps)

Total Costs = $323.39 + $42.048 = $365.438

Example 5 (a) – Standard_V2 with Autoscaling, time-based calculations

Let's assume you've provisioned a standard_V2 with autoscaling enabled and set the minimum instance count to 0 and this application gateway is active for 2 hours. During the first hour, it receives traffic that can be handled by 10 Capacity Units and during the second hour it receives traffic that required 20 Capacity Units to handle the load. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Fixed Price = $0.246 * 2 (Hours) = $0.492

Variable Costs = $0.008 * 10(capacity units) * 1 (Hours) + $0.008 * 20(capacity units) * 1 (Hours) = $0.24

Total Costs = $0.492 + $0.24 = $0.732

V1 SKUs

Standard Application Gateway and WAF V1 SKUs are billed as a combination of:

- Fixed Cost

  Cost based on the time a particular type of Application Gateway/WAF is provisioned and running for processing requests. The fixed cost component takes in consideration the following factors:

  - Tier - Standard Application Gateway or WAF
  - Size - Small, Medium & Large
  - Instance Count - Number of instances to be deployed

  For N instances, the costs associated will be N * cost of one Instance of a particular Tier(Standard & WAF) & Size(Small, Medium & Large) combination.

- Variable Cost

  Cost based on the amount of data processed by the Application Gateway/WAF. Both the request and response bytes processed by the Application Gateway would be considered for billing.

Total Cost = Fixed Cost + Variable Cost

Standard Application Gateway

Fixed Cost & Variable Cost will be calculated according to the Application Gateway type. The following table shows example prices based on a snapshot of East US pricing and are meant for illustration purposes only.

Fixed Cost (East US region pricing)

APPLICATION GATEWAY TYPE	COSTS ($/HR)
Small	$0.025
Medium	$0.07

APPLICATION GATEWAY TYPE	COSTS ($/HR)
Large	$0.32

Monthly price estimates are based on 730 hours of usage per month.

Variable Cost (East US region pricing)

DATA PROCESSED	SMALL ($/GB)	MEDIUM ($/GB)	LARGE ($/GB)
First 10 TB/month	$0.008	Free	Free
Next 30 TB (10–40 TB)/month	$0.008	$0.007	Free
Over 40 TB/month	$0.008	$0.007	$0.0035

For more pricing information according to your region, see the pricing page.

WAF V1

Fixed Cost & Variable Costs will be calculated according to the provisioned Application Gateway type.

The following table shows example prices based on a snapshot of East US pricing and are for illustration purposes only.

Fixed Cost (East US region pricing)

APPLICATION GATEWAY TYPE	COSTS ($/HR)
Small	NA
Medium	$0.126
Large	$0.448

Monthly price estimates are based on 730 hours of usage per month.

Variable Cost (East US region pricing)

DATA PROCESSED	SMALL ($/GB)	MEDIUM ($/GB)	LARGE ($/GB)
First 10 TB/month	$0.008	Free	Free
Next 30 TB (10–40 TB)/month	$0.008	$0.007	Free
Over 40 TB/month	$0.008	$0.007	$0.0035

For more pricing information according to your region, see the pricing page.

> **NOTE**
> Outbound data transfers - data going out of Azure data centers from application gateways will be charged at standard data transfer rates.

Example 1 (a) – Standard Application Gateway with 1 instance count

Let's assume you've provisioned a standard Application Gateway of medium type with 1 instance and it processes 500 GB in a month. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Fixed Price = $0.07 * 730 (Hours) = $51.1 Monthly price estimates are based on 730 hours of usage per month.

Variable Costs = Free (Medium tier has no costs for the first 10 TB processed per month) Total Costs = $51.1 + 0 = $51.1

> **NOTE**
> To support high availability scenarios, it is required to setup a minimum of 2 instances for V1 SKUs. See SLA for Application Gateway

Example 1 (b) – Standard Application Gateway with > 1 instance count

Let's assume you've provisioned a standard Application Gateway of medium type with five instances and it processes 500 GB in a month. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Fixed Price = 5 (Instance count) * $0.07 * 730 (Hours) = $255.5 Monthly price estimates are based on 730 hours of usage per month.

Variable Costs = Free (Medium tier has no costs for the first 10 TB processed per month) Total Costs = $255.5 + 0 = $255.5

Example 2 – WAF Application Gateway

Let's assume you've provisioned a small type standard Application Gateway and a large type WAF Application Gateway for the first 15 days of the month. The small application gateway processes 15 TB in the duration that it is active and the large WAF application gateway processes 100 TB in the duration that it is active. Your Application Gateway costs using the pricing mentioned above would be calculated as follows:

Small instance Standard Application Gateway

24 Hours * 15 Days = 360 Hours

Fixed Price = $0.025 * 360 (Hours) = $9

Variable Costs = 10 * 1000 * $0.008/GB + 5 * 1000 * $0.008/GB = $120

Total Costs = $9 + $120 = $129

Large instance WAF Application Gateway

24 Hours * 15 Days = 360 Hours

Fixed Price = $0.448 * 360 (Hours) = $161.28

Variable Costs = 60 * 1000 * $0.0035/GB = $210 (Large tier has no costs for the first 40 TB processed per month)

Total Costs = $161.28 + $210 = $371.28

Monitoring Billed Usage

You can view the amount of consumption for different parameters (compute unit, throughput & persistent connections) as well as the Capacity Units being utilized as part of the Application Gateway metrics under the **Monitoring** section.

Useful metrics for cost estimation

- Current capacity units

  Count of capacity units consumed to load balance the traffic across the three parameters - Current connections, Throughput and Compute unit

- Fixed Billable Capacity Units

  The minimum number of capacity units kept provisioned as per the minimum instance count setting (one instance translates to 10 capacity units) in the Application Gateway configuration.

- Estimated Billed Capacity units

  Estimated Billed Capacity units indicates the number of capacity units using which the billing is estimated. This is calculated as the greater value between Current capacity units (capacity units required to load balance the traffic) and Fixed billable capacity units (minimum capacity units kept provisioned).

More metrics such as throughput, current connections and compute units are also available to understand bottlenecks and estimate the number of capacity units required. Detailed information is available at Application Gateway Metrics

Example - Estimating Capacity Units being utilized

Observed Metrics:

- Compute Units = 17.38
- Throughput = 1.37M Bytes/sec - 10.96 Mbps
- Current Connections = 123.08k
- Capacity Units calculated = max(17.38, 10.96/2.22, 123.08k/2500) = 49.232

Observed Capacity Units in metrics = 49.23

Next steps

See the following articles to learn more about how pricing works in Azure Application Gateway:

- Azure Application Gateway pricing page
- Azure Application Gateway pricing calculator

Manage web traffic with an application gateway using Azure PowerShell

8/22/2022 • 6 minutes to read • Edit Online

Application gateway is used to manage and secure web traffic to servers that you maintain. You can use Azure PowerShell to create an application gateway that uses a virtual machine scale set for backend servers to manage web traffic. In this example, the scale set contains two virtual machine instances that are added to the default backend pool of the application gateway.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Create a virtual machine scale set with the default backend pool

If you prefer, you can complete this procedure using Azure CLI.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.

3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.

4. Select **Enter** to run the code.

If you choose to install and use PowerShell locally, this article requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az`. If you need to upgrade, see Install Azure PowerShell module. If you are running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Configure the subnets named *myBackendSubnet* and *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24

$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24

$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig

$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Static `
  -Sku Standard
```

Create an application gateway

In this section you create resources that support the application gateway, and then finally create it. The resources that you create include:

- *IP configurations and frontend port* - Associates the subnet that you previously created to the application gateway and assigns a port to use to access it.
- *Default pool* - All application gateways must have at least one backend pool of servers.
- *Default listener and rule* - The default listener listens for traffic on the port that was assigned and the default rule sends traffic to the default pool.

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$subnet=$vnet.Subnets[1]

$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet

$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip

$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the backend pool and settings

Create the backend pool named *appGatewayBackendPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend address pools using New-AzApplicationGatewayBackendHttpSettings.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool

$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the default listener and rule

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this example, you create a basic listener that listens for traffic at the root URL.

Create a listener named *mydefaultListener* using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *rule1* using New-AzApplicationGatewayRequestRoutingRule.

```
$defaultlistener = New-AzApplicationGatewayHttpListener `
  -Name mydefaultListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport

$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultlistener `
  -BackendAddressPool $defaultPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_v2 `
  -Tier Standard_v2 `
  -Capacity 2

$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultlistener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku
```

Create a virtual machine scale set

In this example, you create a virtual machine scale set to provide servers for the backend pool in the application gateway. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool `
  -ApplicationGateway $appgw

$ipConfig = New-AzVmssIpConfig `
  -Name myVmssIPConfig `
  -SubnetId $vnet.Subnets[0].Id `
  -ApplicationGatewayBackendAddressPoolsId $backendPool.Id

$vmssConfig = New-AzVmssConfig `
  -Location eastus `
  -SkuCapacity 2 `
  -SkuName Standard_DS2_v2 `
  -UpgradePolicyMode Automatic

Set-AzVmssStorageProfile $vmssConfig `
  -ImageReferencePublisher MicrosoftWindowsServer `
  -ImageReferenceOffer WindowsServer `
  -ImageReferenceSku 2016-Datacenter `
  -ImageReferenceVersion latest `
  -OsDiskCreateOption FromImage

Set-AzVmssOsProfile $vmssConfig `
  -AdminUsername azureuser `
  -AdminPassword "Azure123456!" `
  -ComputerNamePrefix myvmss

Add-AzVmssNetworkInterfaceConfiguration `
  -VirtualMachineScaleSet $vmssConfig `
  -Name myVmssNetConfig `
  -Primary $true `
  -IPConfiguration $ipConfig

New-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmssConfig
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss

Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
  -Name "customScript" `
  -Publisher "Microsoft.Compute" `
  -Type "CustomScriptExtension" `
  -TypeHandlerVersion 1.8 `
  -Setting $publicSettings

Update-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmss
```

Test the application gateway

Use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Restrict web traffic with a web application firewall

Manage web traffic with an application gateway using the Azure CLI

8/22/2022 • 4 minutes to read • Edit Online

Application gateway is used to manage and secure web traffic to servers that you maintain. You can use the Azure CLI to create an application gateway that uses a virtual machine scale set for backend servers. In this example, the scale set contains two virtual machine instances. The scale set is added to the default backend pool of the application gateway.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Create a virtual machine scale set with the default backend pool

If you prefer, you can complete this procedure using Azure PowerShell.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet named *myBackendSubnet* that's needed by the backend servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24

az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard
```

Create an application gateway

Use az network application-gateway create to create the application gateway named *myAppGateway*. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myPublicIPAddress* that you previously created.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_v2 \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 80 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you'll see these new features:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Create a Virtual Machine Scale Set

In this example, you create a virtual machine scale set that provides servers for the backend pool in the application gateway. The virtual machines in the scale set are associated with *myBackendSubnet* and *appGatewayBackendPool*. To create the scale set, use az vmss create.

```
az vmss create \
  --name myvmss \
  --resource-group myResourceGroupAG \
  --image UbuntuLTS \
  --admin-username azureuser \
  --admin-password Azure123456! \
  --instance-count 2 \
  --vnet-name myVNet \
  --subnet myBackendSubnet \
  --vm-sku Standard_DS2 \
  --upgrade-policy-mode Automatic \
  --app-gateway myAppGateway \
  --backend-pool-name appGatewayBackendPool
```

Install NGINX

Now you can install NGINX on the virtual machine scale set so you can test HTTP connectivity to the backend pool.

```
az vmss extension set \
  --publisher Microsoft.Azure.Extensions \
  --version 2.0 \
  --name CustomScript \
  --resource-group myResourceGroupAG \
  --vmss-name myvmss \
  --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"], "commandToExecute": "./install_nginx.sh" }'
```

Test the application gateway

To get the public IP address of the application gateway, use az network public-ip show. Copy the public IP address, and then paste it into the address bar of your browser.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroupAG --location eastus
```

Next steps

Restrict web traffic with a web application firewall

Configure an application gateway with an internal load balancer (ILB) endpoint

8/22/2022 • 5 minutes to read • Edit Online

Azure Application Gateway Standard v1 can be configured with an Internet-facing VIP or with an internal endpoint that isn't exposed to the Internet. An internal endpoint uses a private IP address for the frontend, which is also known as an *internal load balancer (ILB) endpoint*.

Configuring the gateway using a frontend private IP address is useful for internal line-of-business applications that aren't exposed to the Internet. It's also useful for services and tiers within a multi-tier application that are in a security boundary that isn't exposed to the Internet but:

- still require round-robin load distribution
- session stickiness
- or Transport Layer Security (TLS) termination (previously known as Secure Sockets Layer (SSL)).

This article guides you through the steps to configure a Standard v1 Application Gateway with an ILB using the Azure portal.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Sign in to Azure

Sign in to the Azure portal.

Create an application gateway

For Azure to communicate between the resources that you create, it needs a virtual network. Either create a new virtual network or use an existing one.

In this example, you create a new virtual network. You can create a virtual network at the same time that you create the application gateway. Application Gateway instances are created in separate subnets. There are two subnets in this example: one for the application gateway, and another for the backend servers.

1. Expand the portal menu and select **Create a resource**.
2. Select **Networking** and then select **Application Gateway** in the Featured list.
3. Enter *myAppGateway* for the name of the application gateway and *myResourceGroupAG* for the new resource group.
4. For **Region**, select **Central US**.
5. For **Tier**, select **Standard**.
6. Under **Configure virtual network** select **Create new**, and then enter these values for the virtual network:

- *myVNet* - for the name of the virtual network.
- *10.0.0.0/16* - for the virtual network address space.
- *myAGSubnet* - for the subnet name.
- *10.0.0.0/24* - for the subnet address space.
- *myBackendSubnet* - for the backend subnet name.
- *10.0.1.0/24* - for the backend subnet address space.

7. Select **OK** to create the virtual network and subnets.

8. Select **Next : Frontends**.

9. For **Frontend IP address type**, select **Private**.

   By default, it's a dynamic IP address assignment. The first available address of the configured subnet is assigned as the frontend IP address.

   > **NOTE**
   >
   > Once allocated, the IP address type (static or dynamic) cannot be changed later.

10. Select **Next:Backends**.

11. Select **Add a backend pool**.

12. For **Name**, type *appGatewayBackendPool*.

13. For **Add backend pool without targets**, select **Yes**. You'll add the targets later.

14. Select **Add**.

15. Select **Next:Configuration**.

16. Under **Routing rules**, select **Add a routing rule**.

17. For **Rule name**, type *Rrule-01*.

18. For **Listener name**, type *Listener-01*.

19. For **Frontend IP**, select **Private**.
20. Accept the remaining defaults and select the **Backend targets** tab.
21. For **Target type**, select **Backend pool**, and then select **appGatewayBackendPool**.
22. For **HTTP setting**, select **Add new**.
23. For **HTTP setting name**, type *http-setting-01*.
24. For **Backend protocol**, select **HTTP**.
25. For **Backend port**, type *80*.
26. Accept the remaining defaults, and select **Add**.
27. On the **Add a routing rule** page, select **Add**.
28. Select **Next: Tags**.
29. Select **Next: Review + create**.
30. Review the settings on the summary page, and then select **Create** to create the network resources and the application gateway. It may take several minutes to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

Add backend pool

The backend pool is used to route requests to the backend servers that serve the request. The backend can be composed of NICs, virtual machine scale sets, public IP addresses, internal IP addresses, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you use virtual machines as the target backend. You can either use existing virtual machines or create new ones. In this example, you create two virtual machines that Azure uses as backend servers for the application gateway.

To do this, you:

1. Create two new virtual machines, *myVM* and *myVM2*, used as backend servers.
2. Install IIS on the virtual machines to verify that the application gateway was created successfully.
3. Add the backend servers to the backend pool.

Create a virtual machine

1. Select **Create a resource**.
2. Select **Compute** and then select **Virtual machine**.
3. Enter these values for the virtual machine:
   - Select your subscription.
   - Select *myResourceGroupAG* for **Resource group**.
   - Type *myVM* for **Virtual machine name**.
   - Select **Windows Server 2019 Datacenter** for **Image**.
   - Type a valid **Username**.
   - Type a valid **Password**.
4. Accept the remaining defaults and select **Next: Disks**.
5. Accept the defaults and select **Next : Networking**.
6. Ensure that **myVNet** is selected for the virtual network and the subnet is **myBackendSubnet**.
7. Accept the remaining defaults, and select **Next : Management**.
8. Select **Disable** to disable boot diagnostics.
9. Select **Review + create**.

10. Review the settings on the summary page, and then select **Create**. It may take several minutes to create the VM. Wait until the deployment finishes successfully before moving on to the next section.

Install IIS

1. Open the Cloud Shell and ensure that it's set to **PowerShell**.

2. Run the following command to install IIS on the virtual machine:

```
Set-AzVMExtension `
     -ResourceGroupName myResourceGroupAG `
     -ExtensionName IIS `
     -VMName myVM `
     -Publisher Microsoft.Compute `
     -ExtensionType CustomScriptExtension `
     -TypeHandlerVersion 1.4 `
     -SettingString '{"commandToExecute":"powershell Add-WindowsFeature Web-Server; powershell Add-
Content -Path \"C:\\inetpub\\wwwroot\\Default.htm\" -Value $($env:computername)"}' `
      -Location CentralUS
```

3. Create a second virtual machine and install IIS using the steps that you just finished. Use myVM2 for the virtual machine name and for `VMName` in `Set-AzVMExtension`.

Add backend servers to backend pool

1. Select **All resources**, and then select **myAppGateway**.
2. Select **Backend pools**, and then select **appGatewayBackendPool**.
3. Under **Target type** select **Virtual machine** and under **Target**, select the vNIC associated with myVM.
4. Repeat to add MyVM2.

5. Select **Save**.

Create a client virtual machine

The client virtual machine is used to connect to the application gateway backend pool.

- Create a third virtual machine using the previous steps. Use myVM3 for the virtual machine name.

Test the application gateway

1. On the myAppGateway page, select **Frontend IP Configurations** to note the frontend private IP address.

2. Copy the private IP address, and then paste it into the browser address bar on myVM3 to access the application gateway backend pool.

Next steps

If you want to monitor the health of your backend pool, see Back-end health and diagnostic logs for Application Gateway.

Create an application gateway with an internal load balancer (ILB)

8/22/2022 • 7 minutes to read • Edit Online

Azure Application Gateway Standard v1 can be configured with an Internet-facing VIP or with an internal endpoint that is not exposed to the Internet, also known as an internal load balancer (ILB) endpoint. Configuring the gateway with an ILB is useful for internal line-of-business applications that are not exposed to the Internet. It's also useful for services and tiers within a multi-tier application that sit in a security boundary that is not exposed to the Internet but still require round-robin load distribution, session stickiness, or Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL), termination.

This article walks you through the steps to configure a Standard v1 Application Gateway with an ILB.

Before you begin

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

1. Install the latest version of the Azure PowerShell module by following the install instructions.
2. You create a virtual network and a subnet for Application Gateway. Make sure that no virtual machines or cloud deployments are using the subnet. Application Gateway must be by itself in a virtual network subnet.
3. The servers that you configure to use the application gateway must exist or have their endpoints created either in the virtual network or with a public IP/VIP assigned.

What is required to create an application gateway?

- **Back-end server pool:** The list of IP addresses of the back-end servers. The IP addresses listed should either belong to the virtual network but in a different subnet for the application gateway or should be a public IP/VIP.
- **Back-end server pool settings:** Every pool has settings like port, protocol, and cookie-based affinity. These settings are tied to a pool and are applied to all servers within the pool.
- **Front-end port:** This port is the public port that is opened on the application gateway. Traffic hits this port, and then gets redirected to one of the back-end servers.
- **Listener:** The listener has a front-end port, a protocol (Http or Https, these are case-sensitive), and the SSL certificate name (if configuring SSL offload).
- **Rule:** The rule binds the listener and the back-end server pool and defines which back-end server pool the traffic should be directed to when it hits a particular listener. Currently, only the *basic* rule is supported. The *basic* rule is round-robin load distribution.

Create an application gateway

The difference between using Azure Classic and Azure Resource Manager is the order in which you create the application gateway and the items that need to be configured. With Resource Manager, all items that make an application gateway is configured individually and then put together to create the application gateway resource.

Here are the steps that are needed to create an application gateway:

1. Create a resource group for Resource Manager
2. Create a virtual network and a subnet for the application gateway
3. Create an application gateway configuration object
4. Create an application gateway resource

Create a resource group for Resource Manager

Make sure that you switch PowerShell mode to use the Azure Resource Manager cmdlets. More info is available at Using Windows PowerShell with Resource Manager.

Step 1

```
Connect-AzAccount
```

Step 2

Check the subscriptions for the account.

```
Get-AzSubscription
```

You are prompted to authenticate with your credentials.

Step 3

Choose which of your Azure subscriptions to use.

```
Select-AzSubscription -Subscriptionid "GUID of subscription"
```

Step 4

Create a new resource group (skip this step if you're using an existing resource group).

```
New-AzResourceGroup -Name appgw-rg -location "West US"
```

Azure Resource Manager requires that all resource groups specify a location. This is used as the default location for resources in that resource group. Make sure that all commands to create an application gateway uses the same resource group.

In the preceding example, we created a resource group called "appgw-rg" and location "West US".

Create a virtual network and a subnet for the application gateway

The following example shows how to create a virtual network by using Resource Manager:

Step 1

```
$subnetconfig = New-AzVirtualNetworkSubnetConfig -Name subnet01 -AddressPrefix 10.0.0.0/24
```

This step assigns the address range 10.0.0.0/24 to a subnet variable to be used to create a virtual network.

Step 2

```
$vnet = New-AzVirtualNetwork -Name appgwvnet -ResourceGroupName appgw-rg -Location "West US" -AddressPrefix
10.0.0.0/16 -Subnet $subnetconfig
```

This step creates a virtual network named "appgwvnet" in resource group "appgw-rg" for the West US region using the prefix 10.0.0.0/16 with subnet 10.0.0.0/24.

Step 3

```
$subnet = $vnet.subnets[0]
```

This step assigns the subnet object to variable $subnet for the next steps.

Create an application gateway configuration object

Step 1

```
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name gatewayIP01 -Subnet $subnet
```

This step creates an application gateway IP configuration named "gatewayIP01". When Application Gateway starts, it picks up an IP address from the subnet configured and route network traffic to the IP addresses in the back-end IP pool. Keep in mind that each instance takes one IP address.

Step 2

```
$pool = New-AzApplicationGatewayBackendAddressPool -Name pool01 -BackendIPAddresses
10.1.1.8,10.1.1.9,10.1.1.10
```

This step configures the back-end IP address pool named "pool01" with IP addresses "10.1.1.8, 10.1.1.9, 10.1.1.10". Those are the IP addresses that receive the network traffic that comes from the front-end IP endpoint. You replace the preceding IP addresses to add your own application IP address endpoints.

Step 3

```
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name poolsetting01 -Port 80 -Protocol Http -
CookieBasedAffinity Disabled
```

This step configures application gateway setting "poolsetting01" for the load balanced network traffic in the back-end pool.

Step 4

```
$fp = New-AzApplicationGatewayFrontendPort -Name frontendport01  -Port 80
```

This step configures the front-end IP port named "frontendport01" for the ILB.

Step 5

```
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name fipconfig01 -Subnet $subnet
```

This step creates the front-end IP configuration called "fipconfig01" and associates it with a private IP from the current virtual network subnet.

Step 6

```
$listener = New-AzApplicationGatewayHttpListener -Name listener01  -Protocol Http -FrontendIPConfiguration
$fipconfig -FrontendPort $fp
```

This step creates the listener called "listener01" and associates the front-end port to the front-end IP configuration.

Step 7

```
$rule = New-AzApplicationGatewayRequestRoutingRule -Name rule01 -RuleType Basic -BackendHttpSettings
$poolSetting -HttpListener $listener -BackendAddressPool $pool
```

This step creates the load balancer routing rule called "rule01" that configures the load balancer behavior.

Step 8

```
$sku = New-AzApplicationGatewaySku -Name Standard_Small -Tier Standard -Capacity 2
```

This step configures the instance size of the application gateway.

> **NOTE**
> The default value for Capacity is 2. For Sku Name, you can choose between Standard_Small, Standard_Medium, and Standard_Large.

Create an application gateway by using New-AzureApplicationGateway

Creates an application gateway with all configuration items from the preceding steps. In this example, the application gateway is called "appgwtest".

```
$appgw = New-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg -Location "West US" -
BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting -FrontendIpConfigurations $fipconfig
-GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners $listener -RequestRoutingRules $rule -
Sku $sku
```

This step creates an application gateway with all configuration items from the preceding steps. In the example, the application gateway is called "appgwtest".

Delete an application gateway

To delete an application gateway, you need to do the following steps in order:

1. Use the `Stop-AzApplicationGateway` cmdlet to stop the gateway.
2. Use the `Remove-AzApplicationGateway` cmdlet to remove the gateway.
3. Verify that the gateway has been removed by using the `Get-AzureApplicationGateway` cmdlet.

Step 1

Get the application gateway object and associate it to a variable "$getgw".

```
$getgw =  Get-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg
```

Step 2

Use `Stop-AzApplicationGateway` to stop the application gateway. This sample shows the `Stop-AzApplicationGateway` cmdlet on the first line, followed by the output.

```
Stop-AzApplicationGateway -ApplicationGateway $getgw
```

```
VERBOSE: 9:49:34 PM - Begin Operation: Stop-AzureApplicationGateway
VERBOSE: 10:10:06 PM - Completed Operation: Stop-AzureApplicationGateway
Name       HTTP Status Code     Operation ID                             Error
----       ----------------     ------------                             ----
Successful OK                   ce6c6c95-77b4-2118-9d65-e29defadffb8
```

Once the application gateway is in a stopped state, use the `Remove-AzApplicationGateway` cmdlet to remove the service.

```
Remove-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg -Force
```

```
VERBOSE: 10:49:34 PM - Begin Operation: Remove-AzureApplicationGateway
VERBOSE: 10:50:36 PM - Completed Operation: Remove-AzureApplicationGateway
Name       HTTP Status Code     Operation ID                             Error
----       ----------------     ------------                             ----
Successful OK                   055f3a96-8681-2094-a304-8d9a11ad8301
```

> **NOTE**
>
> The **-force** switch can be used to suppress the remove confirmation message.

To verify that the service has been removed, you can use the `Get-AzApplicationGateway` cmdlet. This step is not required.

```
Get-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg
```

```
VERBOSE: 10:52:46 PM - Begin Operation: Get-AzureApplicationGateway

Get-AzureApplicationGateway : ResourceNotFound: The gateway does not exist.
```

Next steps

If you want to configure SSL offload, see Configure an application gateway for SSL offload.

If you want more information about load balancing options in general, see:

- Azure Load Balancer
- Azure Traffic Manager

Create an application gateway with TLS termination using Azure PowerShell

8/22/2022 • 5 minutes to read • Edit Online

You can use Azure PowerShell to create an application gateway with a certificate for TLS/SSL termination that uses a virtual machine scale set for backend servers. In this example, the scale set contains two virtual machine instances that are added to the default backend pool of the application gateway.

In this article, you learn how to:

- Create a self-signed certificate
- Set up a network
- Create an application gateway with the certificate
- Create a virtual machine scale set with the default backend pool

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This article requires the Azure PowerShell module version 1.0.0 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. If you're running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a self-signed certificate

For production use, you should import a valid certificate signed by trusted provider. For this article, you create a self-signed certificate using New-SelfSignedCertificate. You can use Export-PfxCertificate with the Thumbprint that was returned to export a pfx file from the certificate.

```
New-SelfSignedCertificate `
  -certstorelocation cert:\localmachine\my `
  -dnsname www.contoso.com
```

You should see something like this result:

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject
----------                                -------
E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630  CN=www.contoso.com
```

Use the thumbprint to create the pfx file:

```
$pwd = ConvertTo-SecureString -String "Azure123456!" -Force -AsPlainText

Export-PfxCertificate `
  -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `
  -FilePath c:\appgwcert.pfx `
  -Password $pwd
```

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group named *myResourceGroupAG* with New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Configure the subnets named *myBackendSubnet* and *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24

$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24

$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig

$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Static `
  -Sku Standard
```

Create an application gateway

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$subnet=$vnet.Subnets[0]

$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet

$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip

$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 443
```

Create the backend pool and settings

Create the backend pool named *appGatewayBackendPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend pool using New-AzApplicationGatewayBackendHttpSettings.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool

$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the default listener and rule

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this example, you create a basic listener that listens for HTTPS traffic at the root URL.

Create a certificate object using New-AzApplicationGatewaySslCertificate and then create a listener named *mydefaultListener* using New-AzApplicationGatewayHttpListener with the frontend configuration, frontend port, and certificate that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *rule1* using New-AzApplicationGatewayRequestRoutingRule.

```
$pwd = ConvertTo-SecureString `
  -String "Azure123456!" `
  -Force `
  -AsPlainText

$cert = New-AzApplicationGatewaySslCertificate `
  -Name "appgwcert" `
  -CertificateFile "c:\appgwcert.pfx" `
  -Password $pwd

$defaultlistener = New-AzApplicationGatewayHttpListener `
  -Name mydefaultListener `
  -Protocol Https `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport `
  -SslCertificate $cert

$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultlistener `
  -BackendAddressPool $defaultPool `
  -BackendHttpSettings $poolSettings `
  -priority 100
```

Create the application gateway with the certificate

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway with the certificate.

Create the application gateway

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_v2 `
  -Tier Standard_v2 `
  -Capacity 2

$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultlistener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku `
  -SslCertificates $cert
```

Create a virtual machine scale set

In this example, you create a virtual machine scale set to provide servers for the backend pool in the application gateway. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool `
  -ApplicationGateway $appgw

$ipConfig = New-AzVmssIpConfig `
  -Name myVmssIPConfig `
  -SubnetId $vnet.Subnets[1].Id `
  -ApplicationGatewayBackendAddressPoolsId $backendPool.Id

$vmssConfig = New-AzVmssConfig `
  -Location eastus `
  -SkuCapacity 2 `
  -SkuName Standard_DS2 `
  -UpgradePolicyMode Automatic

Set-AzVmssStorageProfile $vmssConfig `
  -ImageReferencePublisher MicrosoftWindowsServer `
  -ImageReferenceOffer WindowsServer `
  -ImageReferenceSku 2016-Datacenter `
  -ImageReferenceVersion latest `
  -OsDiskCreateOption FromImage

Set-AzVmssOsProfile $vmssConfig `
  -AdminUsername azureuser `
  -AdminPassword "Azure123456!" `
  -ComputerNamePrefix myvmss

Add-AzVmssNetworkInterfaceConfiguration `
  -VirtualMachineScaleSet $vmssConfig `
  -Name myVmssNetConfig `
  -Primary $true `
  -IPConfiguration $ipConfig

New-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmssConfig
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss

Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
  -Name "customScript" `
  -Publisher "Microsoft.Compute" `
  -Type "CustomScriptExtension" `
  -TypeHandlerVersion 1.8 `
  -Setting $publicSettings

Update-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmss
```

Test the application gateway

You can use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

To accept the security warning if you used a self-signed certificate, select **Details** and then **Go on to the webpage**. Your secured IIS website is then displayed as in the following example:

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Create an application gateway that hosts multiple web sites

Create an application gateway with TLS termination using the Azure CLI

8/22/2022 • 4 minutes to read • Edit Online

You can use the Azure CLI to create an application gateway with a certificate for TLS termination. For backend servers, you can use a virtual machine scale set. In this example, the scale set contains two virtual machine instances that are added to the default backend pool of the application gateway.

In this article, you learn how to:

- Create a self-signed certificate
- Set up a network
- Create an application gateway with the certificate
- Create a virtual machine scale set with the default backend pool

If you prefer, you can complete this procedure using Azure PowerShell.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a self-signed certificate

For production use, you should import a valid certificate signed by trusted provider. For this article, you create a self-signed certificate and pfx file using the openssl command.

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out appgwcert.crt
```

Enter values that make sense for your certificate. You can accept the default values.

```
openssl pkcs12 -export -out appgwcert.pfx -inkey privateKey.key -in appgwcert.crt
```

Enter the password for the certificate. In this example, *Azure123456!* is being used.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet named *myBackendSubnet* that's needed by the backend servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24

az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard \
  --location eastus
```

Create the application gateway

You can use az network application-gateway create to create the application gateway. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings.

The application gateway is assigned to *myAGSubnet* and *myAGPublicIPAddress* that you previously created. In this example, you associate the certificate that you created and its password when you create the application gateway.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_v2 \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 443 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --priority "1" \
  --public-ip-address myAGPublicIPAddress \
  --cert-file appgwcert.pfx \
  --cert-password "Azure123456!"
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features of it:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Create a virtual machine scale set

In this example, you create a virtual machine scale set that provides servers for the default backend pool in the application gateway. The virtual machines in the scale set are associated with *myBackendSubnet* and *appGatewayBackendPool*. To create the scale set, you can use az vmss create.

```
az vmss create \
  --name myvmss \
  --resource-group myResourceGroupAG \
  --image UbuntuLTS \
  --admin-username azureuser \
  --admin-password Azure123456! \
  --instance-count 2 \
  --vnet-name myVNet \
  --subnet myBackendSubnet \
  --vm-sku Standard_DS2 \
  --upgrade-policy-mode Automatic \
  --app-gateway myAppGateway \
  --backend-pool-name appGatewayBackendPool
```

Install NGINX

```
az vmss extension set \
  --publisher Microsoft.Azure.Extensions \
  --version 2.0 \
  --name CustomScript \
  --resource-group myResourceGroupAG \
  --vmss-name myvmss \
  --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"],
  "commandToExecute": "./install_nginx.sh" }'
```

Test the application gateway

To get the public IP address of the application gateway, you can use az network public-ip show.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

Copy the public IP address, and then paste it into the address bar of your browser. For this example, the URL is: **https://52.170.203.149**.

To accept the security warning if you used a self-signed certificate, select **Details** and then **Go on to the webpage**. Your secured NGINX site is then displayed as in the following example:

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroupAG --location eastus
```

Next steps

Create an application gateway that hosts multiple web sites

Configure TLS termination with Key Vault certificates using Azure PowerShell

8/22/2022 • 2 minutes to read • Edit Online

Azure Key Vault is a platform-managed secret store that you can use to safeguard secrets, keys, and TLS/SSL certificates. Azure Application Gateway supports integration with Key Vault for server certificates that are attached to HTTPS-enabled listeners. This support is limited to the Application Gateway v2 SKU.

For more information, see TLS termination with Key Vault certificates.

This article shows you how to use an Azure PowerShell script to integrate your key vault with your application gateway for TLS/SSL termination certificates.

This article requires Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az`. If you need to upgrade, see Install Azure PowerShell module. To run the commands in this article, you also need to create a connection with Azure by running `Connect-AzAccount`.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

Before you begin, you must have the ManagedServiceIdentity module installed:

```
Install-Module -Name Az.ManagedServiceIdentity
Connect-AzAccount
Select-AzSubscription -Subscription <your subscription>
```

Example script

Set up variables

```
$rgname = "KeyVaultTest"
$location = "East US"
$kv = "<your key vault name>"
$appgwName = "AppGwKVIntegration"
```

> **IMPORTANT**
>
> The key vault name must be universally unique.

Create a resource group and a user-managed identity

```
$resourceGroup = New-AzResourceGroup -Name $rgname -Location $location
$identity = New-AzUserAssignedIdentity -Name "appgwKeyVaultIdentity" `
  -Location $location -ResourceGroupName $rgname
```

Create a key vault, policy, and certificate to be used by the application gateway

```
$keyVault = New-AzKeyVault -Name $kv -ResourceGroupName $rgname -Location $location
Set-AzKeyVaultAccessPolicy -VaultName $kv -PermissionsToSecrets get -ObjectId $identity.PrincipalId

$policy = New-AzKeyVaultCertificatePolicy -ValidityInMonths 12 `
  -SubjectName "CN=www.contoso11.com" -IssuerName self `
  -RenewAtNumberOfDaysBeforeExpiry 30
Set-AzKeyVaultAccessPolicy -VaultName $kv -EmailAddress <your email address> -PermissionsToCertificates
create,get,list
$certificate = Add-AzKeyVaultCertificate -VaultName $kv -Name "cert1" -CertificatePolicy $policy
$certificate = Get-AzKeyVaultCertificate -VaultName $kv -Name "cert1"
$secretId = $certificate.SecretId.Replace($certificate.Version, "")
```

Create a virtual network

```
$sub1 = New-AzVirtualNetworkSubnetConfig -Name "appgwSubnet" -AddressPrefix "10.0.0.0/24"
$sub2 = New-AzVirtualNetworkSubnetConfig -Name "backendSubnet" -AddressPrefix "10.0.1.0/24"
$vnet = New-AzvirtualNetwork -Name "Vnet1" -ResourceGroupName $rgname -Location $location `
  -AddressPrefix "10.0.0.0/16" -Subnet @($sub1, $sub2)
```

Create a static public virtual IP (VIP) address

```
$publicip = New-AzPublicIpAddress -ResourceGroupName $rgname -name "AppGwIP" `
  -location $location -AllocationMethod Static -Sku Standard
```

Create pool and front-end ports

```
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name "appgwSubnet" -VirtualNetwork $vnet

$gipconfig = New-AzApplicationGatewayIPConfiguration -Name "AppGwIpConfig" -Subnet $gwSubnet
$fipconfig01 = New-AzApplicationGatewayFrontendIPConfig -Name "fipconfig" -PublicIPAddress $publicip
$pool = New-AzApplicationGatewayBackendAddressPool -Name "pool1" `
  -BackendIPAddresses testbackend1.westus.cloudapp.azure.com, testbackend2.westus.cloudapp.azure.com
$fp01 = New-AzApplicationGatewayFrontendPort -Name "port1" -Port 443
$fp02 = New-AzApplicationGatewayFrontendPort -Name "port2" -Port 80
```

Point the TLS/SSL certificate to your key vault

```
$sslCert01 = New-AzApplicationGatewaySslCertificate -Name "SSLCert1" -KeyVaultSecretId $secretId
```

Create listeners, rules, and autoscale

```
$listener01 = New-AzApplicationGatewayHttpListener -Name "listener1" -Protocol Https `
  -FrontendIPConfiguration $fipconfig01 -FrontendPort $fp01 -SslCertificate $sslCert01
$listener02 = New-AzApplicationGatewayHttpListener -Name "listener2" -Protocol Http `
  -FrontendIPConfiguration $fipconfig01 -FrontendPort $fp02
$poolSetting01 = New-AzApplicationGatewayBackendHttpSetting -Name "setting1" -Port 80 `
  -Protocol Http -CookieBasedAffinity Disabled
$rule01 = New-AzApplicationGatewayRequestRoutingRule -Name "rule1" -RuleType basic `
  -BackendHttpSettings $poolSetting01 -HttpListener $listener01 -BackendAddressPool $pool
$rule02 = New-AzApplicationGatewayRequestRoutingRule -Name "rule2" -RuleType basic `
  -BackendHttpSettings $poolSetting01 -HttpListener $listener02 -BackendAddressPool $pool
$autoscaleConfig = New-AzApplicationGatewayAutoscaleConfiguration -MinCapacity 3
$sku = New-AzApplicationGatewaySku -Name Standard_v2 -Tier Standard_v2
```

Assign the user-managed identity to the application gateway

```
$appgwIdentity = New-AzApplicationGatewayIdentity -UserAssignedIdentityId $identity.Id
```

Create the application gateway

```
$appgw = New-AzApplicationGateway -Name $appgwName -Identity $appgwIdentity -ResourceGroupName $rgname `
  -Location $location -BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting01 `
  -GatewayIpConfigurations $gipconfig -FrontendIpConfigurations $fipconfig01 `
  -FrontendPorts @($fp01, $fp02) -HttpListeners @($listener01, $listener02) `
  -RequestRoutingRules @($rule01, $rule02) -Sku $sku `
  -SslCertificates $sslCert01 -AutoscaleConfiguration $autoscaleConfig
```

Next steps

Learn more about TLS termination

Configure TLS termination with Key Vault certificates using Azure portal

8/22/2022 • 8 minutes to read • Edit Online

Azure Key Vault is a platform-managed secret store that you can use to safeguard secrets, keys, and TLS/SSL certificates. Azure Application Gateway supports integration with Key Vault for server certificates that are attached to HTTPS-enabled listeners. This support is limited to the Application Gateway v2 SKU.

For more information, see TLS termination with Key Vault certificates.

This article shows you how to use Azure portal to integrate your key vault with your application gateway for TLS/SSL termination certificates.

Prerequisites

Before you begin, you must have an Azure subscription. If not, create a free account before you begin.

Configuration steps

Create a resource group

1. Sign in to the Azure portal.
2. Select **Resource groups**
3. Select **Add**.
4. Enter the following values:
   - **Subscription**: Select your Azure subscription.
   - **Resource group**: Enter a new resource group name.
   - **Region**: Select an Azure location, such as **East US**.
5. Select **Review + Create**
6. Select **Create**. It takes a few seconds to create a resource group.
7. Select **Refresh** from the top menu to refresh the resource group list, and then select the newly created resource group to open it. Or select **Notification** (the bell icon) from the top, and then select **Go to resource group** to open the newly created resource group

Create a Key Vault

> **IMPORTANT**
>
> The key vault name must be universally unique.

1. In the **Search** box, enter **Key Vault**.
2. From the results list, choose **Key Vault**.
3. On the Key Vault section, choose **Create**.
4. On the **Create key vault** section provide the following information:

- **Name**: A unique name is required.
- **Subscription**: Choose a subscription.
- **Select Resource Group** previously created.
- In the **Location** pull-down menu, choose a **East US**. Leave the other options to their defaults.

5. After providing the information above, select **create**.

At this point, your Azure account is the only one authorized to perform operations on this new vault.

Create Certificate and Issuance Policy

Certificate

1. In the Key Vault that was created earlier, navigate to **Certificate**.

2. Under **Certificate** select your desired method of creation, whether you want to **Import** or **Generate**. If you choose to generate the certificate, you would need to create the issuance policy.

3. Input your Certificate name and complete all the required fields.

4. After providing the information above, select **Create**.

> **IMPORTANT**
>
> Issuance policies only affect certificates that will be issued in the future. Modifying this issuance policy will not affect any existing certificates.

Create a Virtual Network

1. In the **Search** box, enter **Virtual Network**.
2. In the **Virtual Network** page, select **Create**.
3. In **Create virtual network**, enter or select this information in the **Basics tab**:
   - **Subscription**: Choose a subscription.
   - **Select Resource Group** previously created.
   - **Name**: A unique name is required.
   - In the **Location** pull-down menu, choose a **East US**.
4. Select **OK**.
5. Select the **IP Addresses** tab, or select the **Next: IP Addresses** button at the bottom of the page.
6. In IPv4 address space, select the existing address space and change it to 10.1.0.0/16.
7. Select **+ Add subnet**, then enter *MySubnet* for Subnet name and 10.1.0.0/24 for Subnet address range.
8. Select **Add**.
9. Select **OK**.
10. Select the **Review + create** button.

Create an Application Gateway

You'll create the application gateway using the tabs on the **Create an application gateway** page.

1. On the Azure portal menu or from the Home page, select **Create a resource**. The New window appears.
2. Select **Networking** and then select Application Gateway in the Featured list.

Basics tab

1. On the Basics tab, enter these values for the following application gateway settings:

   Resource group: Select **myResourceGroupAG** for the resource group. If it doesn't exist, select **Create new** to create it.

   Application gateway name: Enter *myAppGateway* for the name of the application gateway.

2. For Azure to communicate between the resources that you create, it needs a virtual network. You can either create a new virtual network or use an existing one. In this example, you'll create a new virtual

network at the same time that you create the application gateway. Application Gateway instances are created in separate subnets. You create two subnets in this example: one for the application gateway, and another for the backend servers.

> **NOTE**
> Virtual network service endpoint policies are currently not supported in an Application Gateway subnet.

Under **Configure virtual network**, create a new virtual network by selecting **Create new**. In the Create virtual network window that opens, enter the following values to create the virtual network and two subnets:

Name: Enter myVNet for the name of the virtual network.

Subnet name (Application Gateway subnet): The Subnets grid will show a subnet named Default. Change the name of this subnet to myAGSubnet. The application gateway subnet can contain only application gateways. No other resources are allowed.

Subnet name (backend server subnet): In the second row of the Subnets grid, enter myBackendSubnet in the Subnet name column.

Address range (backend server subnet): In the second row of the Subnets Grid, enter an address range that doesn't overlap with the address range of myAGSubnet. For example, if the address range of myAGSubnet is 10.0.0.0/24, enter 10.0.1.0/24 for the address range of myBackendSubnet.

Select **OK** to close the Create virtual network window and save the virtual network settings.

3. On the **Basics** tab, accept the default values for the other settings and then select **Next: Frontends**.

Frontends tab

1. On the **Frontends** tab, verify Frontend IP address type is set to Public. You can configure the Frontend IP to be Public or Private as per your use case. In this example, you'll choose a Public Frontend IP.

> **NOTE**
> For the Application Gateway v2 SKU, there must be a Public frontend IP configuration. You can still have both a Public and a Private frontend IP configuration, but Private only frontend IP configuration (Only ILB mode) is currently not enabled for the v2 SKU.

2. Select **Add new** for the Public IP address and enter myAGPublicIPAddress for the public IP address name, and then select OK.

3. Select **Next: Backends**.

Backends tab

The backend pool is used to route requests to the backend servers that serve the request. Backend pools can be composed of NICs, virtual machine scale sets, public IP addresses, internal IP addresses, fully qualified domain names (FQDN), and multi-tenant back-ends like Azure App Service. In this example, you'll create an empty backend pool with your application gateway and then add backend targets to the backend pool.

1. On the **Backends** tab, select **Add a backend pool**.

2. In the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:

   Name: Enter myBackendPool for the name of the backend pool.

Add backend pool without targets: Select Yes to create a backend pool with no targets. You'll add backend targets after creating the application gateway.

3. In the **Add a backend pool** window, select **Add** to save the backend pool configuration and return to the Backends tab.

4. On the **Backends** tab, select **Next: Configuration**.

Configuration tab

On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule.

1. Select **Add a routing rule** in the **Routing rules** column.

2. In the **Add a routing rule** window that opens, enter **myRoutingRule** for the Rule name.

3. A routing rule requires a listener. On the **Listener** tab within the **Add a routing rule** window, enter the following values for the listener:

   Listener name: Enter *myListener* for the name of the listener.

   Frontend IP: Select **Public** to choose the public IP you created for the frontend.

   Accept the default values for the other settings on the **Listener** tab, then select the **Backend targets** tab to configure the rest of the routing rule.

4. On the **Backend targets** tab, select **myBackendPool** for the **Backend target**.

5. For the **HTTP setting**, select **Add new** to add a new HTTP setting. The HTTP setting will determine the behavior of the routing rule. In the Add an HTTP setting window that opens, enter *myHTTPSetting* for the HTTP setting name and 80 for the Backend port. Accept the default values for the other settings in the **Add an HTTP setting** window, then select **Add** to return to the **Add a routing rule** window.

6. On the **Add a routing rule** window, select **Add** to save the routing rule and return to the **Configuration** tab.

7. Select **Next: Tags** and then **Next: Review + create**.

Review + create tab

Review the settings on the **Review + create** tab, and then select **Create** to configure the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

Add backend targets to your Backend Pool.

Create user-assigned managed identity

Search for **Managed Identity**

1. Create user assigned managed identity.

2. Go back to Key Vault previously created.

3. Select **Access Policy**.

4. Select **Add access policy**.

5. Fill in the required level of permissions you need.

6. Select **principal** - then choose **None selected**.

7. Search for the managed identity name previously created and use that one. Skip Application identity.

For additional information on configuring Access Policy permissions and Key Vault Firewall permissions, read below articles.

Delegate user-assigned managed identity to Key Vault

Verify Firewall Permissions to Key Vault

Point the TLS/SSL certificate to your key vault by assigning the user-managed identity to the application gateway

1. Go back to the application gateway previously configured
2. Select **Listeners** and then choose **Add listener** (this will add the HTTPS listener)
3. Complete the following fields:
   - **Listener name**: unique-name
   - **Frontend IP**: Public
   - **Port**: 443
   - **Protocol**: HTTPS
   - **Https Settings**: Choose a certificate from Key Vault
   - **Cert name**: unique-name
   - **Managed identity**: select the name of the managed identity previously created from above
   - **Key vault**: select name of key vault previously configured
   - **Certificate**: auto-populates the certificate(s) found in key vault Leave remaining selections as default

Next steps

Learn more about TLS termination

Learn more about Key Vault

Configure end-to-end TLS by using Application Gateway with the portal

8/22/2022 • 4 minutes to read • Edit Online

This article describes how to use the Azure portal to configure end-to-end Transport Layer Security (TLS) encryption, previously known as Secure Sockets Layer (SSL) encryption, through Azure Application Gateway v1 SKU.

> **NOTE**
>
> Application Gateway v2 SKU requires trusted root certificates for enabling end-to-end configuration.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

To configure end-to-end TLS with an application gateway, you need a certificate for the gateway. Certificates are also required for the back-end servers. The gateway certificate is used to derive a symmetric key in compliance with the TLS protocol specification. The symmetric key is then used to encrypt and decrypt the traffic sent to the gateway.

For end-to-end TLS encryption, the right back-end servers must be allowed in the application gateway. To allow this access, upload the public certificate of the back-end servers, also known as Authentication Certificates (v1) or Trusted Root Certificates (v2), to the application gateway. Adding the certificate ensures that the application gateway communicates only with known back-end instances. This configuration further secures end-to-end communication.

To learn more, see Overview of TLS termination and end to end TLS with Application Gateway.

Create a new application gateway with end-to-end TLS

To create a new application gateway with end-to-end TLS encryption, you'll need to first enable TLS termination while creating a new application gateway. This action enables TLS encryption for communication between the client and application gateway. Then, you'll need to put on the Safe Recipients list the certificates for the back-end servers in the HTTP settings. This configuration enables TLS encryption for communication between the application gateway and the back-end servers. That accomplishes end-to-end TLS encryption.

Enable TLS termination while creating a new application gateway

To learn more, see enable TLS termination while creating a new application gateway.

Add authentication/root certificates of back-end servers

1. Select **All resources**, and then select **myAppGateway**.
2. Select **HTTP settings** from the left-side menu. Azure automatically created a default HTTP setting, **appGatewayBackendHttpSettings**, when you created the application gateway.
3. Select **appGatewayBackendHttpSettings**.
4. Under **Protocol**, select **HTTPS**. A pane for **Backend authentication certificates or Trusted root certificates** appears.

5. Select **Create new**.

6. In the **Name** field, enter a suitable name.

7. Select the certificate file in the **Upload CER certificate** box.

   For Standard and WAF (v1) application gateways, you should upload the public key of your back-end server certificate in .cer format.

   For Standard_v2 and WAF_v2 application gateways, you should upload the root certificate of the back-end server certificate in .cer format. If the back-end certificate is issued by a well-known certificate authority (CA), you can select the **Use Well Known CA Certificate** check box, and then you don't have to upload a certificate.

Home > appgwv2 - HTTP settings > appGatewayBackendHttpSettings

appGatewayBackendHttpSettings

appgwv2

Save Discard Delete

Name

appGatewayBackendHttpSettings

* Cookie based affinity

Disabled **Enabled**

Affinity cookie name

ApplicationGatewayAffinity

* Connection draining

Disabled Enabled

* Protocol

HTTP **HTTPS**

☐ Use for App service

☐ Use Well Known CA Certificate

Trusted Root certificates

◉ Create new ○ Select existing

* Name

* Upload CER certificate

Select a file

+ Add certificate

* Port

443

* Request timeout (seconds)

30

8. Select **Save**.

Enable end-to-end TLS for an existing application gateway

To configure an existing application gateway with end-to-end TLS encryption, you must first enable TLS termination in the listener. This action enables TLS encryption for communication between the client and the application gateway. Then, put those certificates for back-end servers in the HTTP settings on the Safe Recipients list. This configuration enables TLS encryption for communication between the application gateway and the back-end servers. That accomplishes end-to-end TLS encryption.

You'll need to use a listener with the HTTPS protocol and a certificate for enabling TLS termination. You can either use an existing listener that meets those conditions or create a new listener. If you choose the former option, you can ignore the following "Enable TLS termination in an existing application gateway" section and move directly to the "Add authentication/trusted root certificates for backend servers" section.

If you choose the latter option, apply the steps in the following procedure.

Enable TLS termination in an existing application gateway

1. Select **All resources**, and then select **myAppGateway**.
2. Select **Listeners** from the left-side menu.
3. Select either **Basic** or **Multi-site** listener depending on your requirements.
4. Under **Protocol**, select **HTTPS**. A pane for **Certificate** appears.
5. Upload the PFX certificate you intend to use for TLS termination between the client and the application gateway.

> **NOTE**
>
> For testing purposes, you can use a self-signed certificate. However, this is not advised for production workloads, because they're harder to manage and aren't completely secure. For more info, see create a self-signed certificate.

6. Add other required settings for the **Listener**, depending on your requirements.
7. Select **OK** to save.

Add authentication/trusted root certificates of back-end servers

1. Select **All resources**, and then select **myAppGateway**.
2. Select **HTTP settings** from the left-side menu. You can either put certificates in an existing back-end HTTP setting on the Safe Recipients list or create a new HTTP setting. (In the next step, the certificate for the default HTTP setting, **appGatewayBackendHttpSettings**, is added to the Safe Recipients list.)
3. Select **appGatewayBackendHttpSettings**.
4. Under **Protocol**, select **HTTPS**. A pane for **Backend authentication certificates or Trusted root certificates** appears.
5. Select **Create new**.
6. In the **Name** field, enter a suitable name.
7. Select the certificate file in the **Upload CER certificate** box.

   For Standard and WAF (v1) application gateways, you should upload the public key of your back-end server certificate in .cer format.

For Standard_v2 and WAF_v2 application gateways, you should upload the root certificate of the back-end server certificate in .cer format. If the back-end certificate is issued by a well-known CA, you can select the **Use Well Known CA Certificate** check box, and then you don't have to upload a certificate.

8. Select **Save**.

Next steps

Manage web traffic with an application gateway using the Azure CLI

Configure end to end TLS by using Application Gateway with PowerShell

8/22/2022 • 11 minutes to read • Edit Online

Overview

Azure Application Gateway supports end-to-end encryption of traffic. Application Gateway terminates the TLS/SSL connection at the application gateway. The gateway then applies the routing rules to the traffic, re-encrypts the packet, and forwards the packet to the appropriate back-end server based on the routing rules defined. Any response from the web server goes through the same process back to the end user.

Application Gateway supports defining custom TLS options. It also supports disabling the following protocol versions: **TLSv1.0**, **TLSv1.1**, and **TLSv1.2**, as well defining which cipher suites to use and the order of preference. To learn more about configurable TLS options, see the TLS policy overview.

> **NOTE**
>
> SSL 2.0 and SSL 3.0 are disabled by default and cannot be enabled. They are considered unsecure and cannot be used with Application Gateway.

Scenario

In this scenario, you learn how to create an application gateway by using end-to-end TLS with PowerShell.

This scenario will:

- Create a resource group named **appgw-rg**.
- Create a virtual network named **appgwvnet** with an address space of **10.0.0.0/16**.
- Create two subnets called **appgwsubnet** and **appsubnet**.
- Create a small application gateway supporting end-to-end TLS encryption that limits TLS protocol versions

and cipher suites.

Before you begin

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

To configure end-to-end TLS with an application gateway, a certificate is required for the gateway and certificates are required for the back-end servers. The gateway certificate is used to derive a symmetric key as per TLS protocol specification. The symmetric key is then used encrypt and decrypt the traffic sent to the gateway. The gateway certificate needs to be in Personal Information Exchange (PFX) format. This file format allows you to export the private key that is required by the application gateway to perform the encryption and decryption of traffic.

For end-to-end TLS encryption, the back end must be explicitly allowed by the application gateway. Upload the public certificate of the back-end servers to the application gateway. Adding the certificate ensures that the application gateway only communicates with known back-end instances. This further secures the end-to-end communication.

The configuration process is described in the following sections.

Create the resource group

This section walks you through creating a resource group that contains the application gateway.

1. Sign in to your Azure account.

```
Connect-AzAccount
```

2. Select the subscription to use for this scenario.

```
Select-Azsubscription -SubscriptionName "<Subscription name>"
```

3. Create a resource group. (Skip this step if you're using an existing resource group.)

```
New-AzResourceGroup -Name appgw-rg -Location "West US"
```

Create a virtual network and a subnet for the application gateway

The following example creates a virtual network and two subnets. One subnet is used to hold the application gateway. The other subnet is used for the back ends that host the web application.

1. Assign an address range for the subnet to be used for the application gateway.

```
$gwSubnet = New-AzVirtualNetworkSubnetConfig -Name 'appgwsubnet' -AddressPrefix 10.0.0.0/24
```

> **NOTE**
> Subnets configured for an application gateway should be properly sized. An application gateway can be configured for up to 10 instances. Each instance takes one IP address from the subnet. Too small of a subnet can adversely affect scaling out an application gateway.

2. Assign an address range to be used for the back-end address pool.

```
$nicSubnet = New-AzVirtualNetworkSubnetConfig  -Name 'appsubnet' -AddressPrefix 10.0.2.0/24
```

3. Create a virtual network with the subnets defined in the preceding steps.

```
$vnet = New-AzvirtualNetwork -Name 'appgwvnet' -ResourceGroupName appgw-rg -Location "West US" -
AddressPrefix 10.0.0.0/16 -Subnet $gwSubnet, $nicSubnet
```

4. Retrieve the virtual network resource and subnet resources to be used in the steps that follow.

```
$vnet = Get-AzvirtualNetwork -Name 'appgwvnet' -ResourceGroupName appgw-rg
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name 'appgwsubnet' -VirtualNetwork $vnet
$nicSubnet = Get-AzVirtualNetworkSubnetConfig -Name 'appsubnet' -VirtualNetwork $vnet
```

Create a public IP address for the front-end configuration

Create a public IP resource to be used for the application gateway. This public IP address is used in one of the steps that follow.

```
$publicip = New-AzPublicIpAddress -ResourceGroupName appgw-rg -Name 'publicIP01' -Location "West US" -
AllocationMethod Dynamic
```

> **IMPORTANT**
> Application Gateway does not support the use of a public IP address created with a defined domain label. Only a public IP address with a dynamically created domain label is supported. If you require a friendly DNS name for the application gateway, we recommend you use a CNAME record as an alias.

Create an application gateway configuration object

All configuration items are set before creating the application gateway. The following steps create the configuration items that are needed for an application gateway resource.

1. Create an application gateway IP configuration. This setting configures which of the subnets the application gateway uses. When application gateway starts, it picks up an IP address from the configured subnet and routes network traffic to the IP addresses in the back-end IP pool. Keep in mind that each instance takes one IP address.

```
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name 'gwconfig' -Subnet $gwSubnet
```

2. Create a front-end IP configuration. This setting maps a private or public IP address to the front end of the application gateway. The following step associates the public IP address in the preceding step with the front-end IP configuration.

```
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name 'fip01' -PublicIPAddress $publicip
```

3. Configure the back-end IP address pool with the IP addresses of the back-end web servers. These IP addresses are the IP addresses that receive the network traffic that comes from the front-end IP endpoint. Replace the IP addresses in the sample with your own application IP address endpoints.

```
$pool = New-AzApplicationGatewayBackendAddressPool -Name 'pool01' -BackendIPAddresses 1.1.1.1,
2.2.2.2, 3.3.3.3
```

> **NOTE**
>
> A fully qualified domain name (FQDN) is also a valid value to use in place of an IP address for the back-end servers. You enable it by using the **-BackendFqdns** switch.

4. Configure the front-end IP port for the public IP endpoint. This port is the port that end users connect to.

```
$fp = New-AzApplicationGatewayFrontendPort -Name 'port01'  -Port 443
```

5. Configure the certificate for the application gateway. This certificate is used to decrypt and reencrypt the traffic on the application gateway.

```
$passwd = ConvertTo-SecureString  <certificate file password> -AsPlainText -Force
$cert = New-AzApplicationGatewaySSLCertificate -Name cert01 -CertificateFile <full path to .pfx file>
-Password $passwd
```

> **NOTE**
>
> This sample configures the certificate used for the TLS connection. The certificate needs to be in .pfx format.

6. Create the HTTP listener for the application gateway. Assign the front-end IP configuration, port, and TLS/SSL certificate to use.

```
$listener = New-AzApplicationGatewayHttpListener -Name listener01 -Protocol Https -
FrontendIPConfiguration $fipconfig -FrontendPort $fp -SSLCertificate $cert
```

7. Upload the certificate to be used on the TLS-enabled back-end pool resources.

> **NOTE**
>
> The default probe gets the public key from the *default* TLS binding on the back-end's IP address and compares the public key value it receives to the public key value you provide here.
>
> If you are using host headers and Server Name Indication (SNI) on the back end, the retrieved public key might not be the intended site to which traffic flows. If you're in doubt, visit https://127.0.0.1/ on the back-end servers to confirm which certificate is used for the *default* TLS binding. Use the public key from that request in this section. If you are using host-headers and SNI on HTTPS bindings and you do not receive a response and certificate from a manual browser request to https://127.0.0.1/ on the back-end servers, you must set up a default TLS binding on the them. If you do not do so, probes fail and the back end is not allowed.

For more information about SNI in Application Gateway, see Overview of TLS termination and end to end

TLS with Application Gateway.

```
$authcert = New-AzApplicationGatewayAuthenticationCertificate -Name 'allowlistcert1' -CertificateFile
C:\cert.cer
```

> **NOTE**
> The certificate provided in the previous step should be the public key of the .pfx certificate present on the back end. Export the certificate (not the root certificate) installed on the back-end server in Claim, Evidence, and Reasoning (CER) format and use it in this step. This step allows the back end with the application gateway.

If you are using the Application Gateway v2 SKU, then create a trusted root certificate instead of an authentication certificate. For more information, see Overview of end to end TLS with Application Gateway:

```
$trustedRootCert01 = New-AzApplicationGatewayTrustedRootCertificate -Name "test1" -CertificateFile
<path to root cert file>
```

8. Configure the HTTP settings for the application gateway back end. Assign the certificate uploaded in the preceding step to the HTTP settings.

```
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name 'setting01' -Port 443 -Protocol
Https -CookieBasedAffinity Enabled -AuthenticationCertificates $authcert
```

For the Application Gateway v2 SKU, use the following command:

```
$poolSetting01 = New-AzApplicationGatewayBackendHttpSettings -Name “setting01” -Port 443 -Protocol
Https -CookieBasedAffinity Disabled -TrustedRootCertificate $trustedRootCert01 -HostName "test1"
```

9. Create a load-balancer routing rule that configures the load balancer behavior. In this example, a basic round-robin rule is created.

```
$rule = New-AzApplicationGatewayRequestRoutingRule -Name 'rule01' -RuleType basic -
BackendHttpSettings $poolSetting -HttpListener $listener -BackendAddressPool $pool
```

10. Configure the instance size of the application gateway. The available sizes are **Standard_Small**, **Standard_Medium**, and **Standard_Large**. For capacity, the available values are **1** through **10**.

```
$sku = New-AzApplicationGatewaySku -Name Standard_Small -Tier Standard -Capacity 2
```

> **NOTE**
> An instance count of 1 can be chosen for testing purposes. It is important to know that any instance count under two instances is not covered by the SLA and is therefore not recommended. Small gateways are to be used for dev test and not for production purposes.

11. Configure the TLS policy to be used on the application gateway. Application Gateway supports the ability to set a minimum version for TLS protocol versions.

The following values are a list of protocol versions that can be defined:

- TLSV1_0
- TLSV1_1
- TLSV1_2

The following example sets the minimum protocol version to TLSv1_2 and enables TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, and TLS_RSA_WITH_AES_128_GCM_SHA256 only.

```
$SSLPolicy = New-AzApplicationGatewaySSLPolicy -MinProtocolVersion TLSv1_2 -CipherSuite
"TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256", "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
"TLS_RSA_WITH_AES_128_GCM_SHA256" -PolicyType Custom
```

Create the application gateway

Using all the preceding steps, create the application gateway. The creation of the gateway is a process that takes a long time to run.

For V1 SKU use the below command

```
$appgw = New-AzApplicationGateway -Name appgateway -SSLCertificates $cert -ResourceGroupName "appgw-rg" -
Location "West US" -BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting -
FrontendIpConfigurations $fipconfig -GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners
$listener -RequestRoutingRules $rule -Sku $sku -SSLPolicy $SSLPolicy -AuthenticationCertificates $authcert -
Verbose
```

For V2 SKU use the below command

```
$appgw = New-AzApplicationGateway -Name appgateway -SSLCertificates $cert -ResourceGroupName "appgw-rg" -
Location "West US" -BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting01 -
FrontendIpConfigurations $fipconfig -GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners
$listener -RequestRoutingRules $rule -Sku $sku -SSLPolicy $SSLPolicy -TrustedRootCertificate
$trustedRootCert01 -Verbose
```

Apply a new certificate if the back-end certificate is expired

Use this procedure to apply a new certificate if the back-end certificate is expired.

1. Retrieve the application gateway to update.

   ```
   $gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroupName AdatumAppGatewayRG
   ```

2. Add the new certificate resource from the .cer file, which contains the public key of the certificate and can also be the same certificate added to the listener for TLS termination at the application gateway.

   ```
   Add-AzApplicationGatewayAuthenticationCertificate -ApplicationGateway $gw -Name 'NewCert' -
   CertificateFile "appgw_NewCert.cer"
   ```

3. Get the new authentication certificate object into a variable (TypeName: Microsoft.Azure.Commands.Network.Models.PSApplicationGatewayAuthenticationCertificate).

   ```
   $AuthCert = Get-AzApplicationGatewayAuthenticationCertificate -ApplicationGateway $gw -Name NewCert
   ```

4. Assign the new certificate into the **BackendHttp** Setting and refer it with the $AuthCert variable. (Specify the HTTP setting name that you want to change.)

```
$out= Set-AzApplicationGatewayBackendHttpSetting -ApplicationGateway $gw -Name "HTTP1" -Port 443 -Protocol
"Https" -CookieBasedAffinity Disabled -AuthenticationCertificates $Authcert
```

5. Commit the change into the application gateway and pass the new configuration contained into the $out variable.

```
Set-AzApplicationGateway -ApplicationGateway $gw
```

Remove an unused expired certificate from HTTP Settings

Use this procedure to remove an unused expired certificate from HTTP Settings.

1. Retrieve the application gateway to update.

```
$gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroupName AdatumAppGatewayRG
```

2. List the name of the authentication certificate that you want to remove.

```
Get-AzApplicationGatewayAuthenticationCertificate -ApplicationGateway $gw | select name
```

3. Remove the authentication certificate from an application gateway.

```
$gw=Remove-AzApplicationGatewayAuthenticationCertificate -ApplicationGateway $gw -Name ExpiredCert
```

4. Commit the change.

```
Set-AzApplicationGateway -ApplicationGateway $gw
```

Limit TLS protocol versions on an existing application gateway

The preceding steps took you through creating an application with end-to-end TLS and disabling certain TLS protocol versions. The following example disables certain TLS policies on an existing application gateway.

1. Retrieve the application gateway to update.

```
$gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroupName AdatumAppGatewayRG
```

2. Define a TLS policy. In the following example, **TLSv1.0** and **TLSv1.1** are disabled and the cipher suites **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**, **TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**, and **TLS_RSA_WITH_AES_128_GCM_SHA256** are the only ones allowed.

```
Set-AzApplicationGatewaySSLPolicy -MinProtocolVersion TLSv1_2 -PolicyType Custom -CipherSuite
"TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256", "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
"TLS_RSA_WITH_AES_128_GCM_SHA256" -ApplicationGateway $gw
```

3. Finally, update the gateway. This last step is a long-running task. When it is done, end-to-end TLS is configured on the application gateway.

```
$gw | Set-AzApplicationGateway
```

Get an application gateway DNS name

After the gateway is created, the next step is to configure the front end for communication. Application Gateway requires a dynamically assigned DNS name when using a public IP, which is not friendly. To ensure end users can hit the application gateway, you can use a CNAME record to point to the public endpoint of the application gateway. For more information, see Configuring a custom domain name for in Azure.

To configure an alias, retrieve details of the application gateway and its associated IP/DNS name by using the **PublicIPAddress** element attached to the application gateway. Use the application gateway's DNS name to create a CNAME record that points the two web applications to this DNS name. We don't recommend the use of A-records, because the VIP can change on restart of the application gateway.

```
Get-AzPublicIpAddress -ResourceGroupName appgw-RG -Name publicIP01
```

```
Name                     : publicIP01
ResourceGroupName        : appgw-RG
Location                 : westus
Id                       : /subscriptions/<subscription_id>/resourceGroups/appgw-
RG/providers/Microsoft.Network/publicIPAddresses/publicIP01
Etag                     : W/"00000d5b-54ed-4907-bae8-99bd5766d0e5"
ResourceGuid             : 00000000-0000-0000-0000-000000000000
ProvisioningState        : Succeeded
Tags                     :
PublicIpAllocationMethod : Dynamic
IpAddress                : xx.xx.xxx.xx
PublicIpAddressVersion   : IPv4
IdleTimeoutInMinutes     : 4
IpConfiguration          : {
                                "Id": "/subscriptions/<subscription_id>/resourceGroups/appgw-
RG/providers/Microsoft.Network/applicationGateways/appgwtest/frontendIP
                           Configurations/frontend1"
                           }
DnsSettings              : {
                                "Fqdn": "00000000-0000-xxxx-xxxx-xxxxxxxxxxxx.cloudapp.net"
                           }
```

Next steps

For more information about hardening the security of your web applications with Web Application Firewall through Application Gateway, see the Web application firewall overview.

Configure mutual authentication with Application Gateway through portal

8/22/2022 • 3 minutes to read • Edit Online

This article describes how to use the Azure portal to configure mutual authentication on your Application Gateway. Mutual authentication means Application Gateway authenticates the client sending the request using the client certificate you upload onto the Application Gateway.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

To configure mutual authentication with an Application Gateway, you need a client certificate to upload to the gateway. The client certificate will be used to validate the certificate the client will present to Application Gateway. For testing purposes, you can use a self-signed certificate. However, this is not advised for production workloads, because they're harder to manage and aren't completely secure.

To learn more, especially about what kind of client certificates you can upload, see Overview of mutual authentication with Application Gateway.

Create a new Application Gateway

First create a new Application Gateway as you would usually through the portal - there are no additional steps needed in the creation to enable mutual authentication. For more information on how to create an Application Gateway in portal, check out our portal quickstart tutorial.

Configure mutual authentication

To configure an existing Application Gateway with mutual authentication, you'll need to first go to the **SSL settings** tab in the Portal and create a new SSL profile. When you create an SSL profile, you'll see two tabs: **Client Authentication** and **SSL Policy**. The **Client Authentication** tab is where you'll upload your client certificate(s). The **SSL Policy** tab is to configure a listener specific SSL policy - for more information, check out Configuring a listener specific SSL policy.

> **IMPORTANT**
>
> Please ensure that you upload the entire client CA certificate chain in one file, and only one chain per file.

1. Search for **Application Gateway** in portal, select **Application gateways**, and click on your existing Application Gateway.
2. Select **SSL settings** from the left-side menu.
3. Click on the plus sign next to **SSL Profiles** at the top to create a new SSL profile.
4. Enter a name under **SSL Profile Name**. In this example, we call our SSL profile *applicationGatewaySSLProfile*.
5. Stay in the **Client Authentication** tab. Upload the PEM certificate you intend to use for mutual authentication between the client and the Application Gateway using the **Upload a new certificate** button.

For more information on how to extract trusted client CA certificate chains to upload here, see how to extract trusted client CA certificate chains.

> **NOTE**
>
> If this isn't your first SSL profile and you've uploaded other client certificates onto your Application Gateway, you can choose to reuse an existing certificate on your gateway through the dropdown menu.

6. Check the **Verify client certificate issuer's DN** box only if you want Application Gateway to verify the client certificate's immediate issuer Distinguished Name.

7. Consider adding a listener specific policy. See instructions at setting up listener specific SSL policies.

8. Select **Add** to save.

Associate the SSL profile with a listener

Now that we've created an SSL profile with mutual authentication configured, we need to associate the SSL profile to the listener to complete the set up of mutual authentication.

1. Navigate to your existing Application Gateway. If you just completed the steps above, you don't need to do anything here.

2. Select **Listeners** from the left-side menu.

3. Click on **Add listener** if you don't already have an HTTPS listener set up. If you already have an HTTPS listener, click on it from the list.

4. Fill out the **Listener name**, **Frontend IP**, **Port**, **Protocol**, and other **HTTPS Settings** to fit your requirements.

5. Check the **Enable SSL Profile** checkbox so that you can select which SSL Profile to associate with the listener.

6. Select the SSL profile you just created from the dropdown list. In this example, we choose the SSL profile we created from the earlier steps: *applicationGatewaySSLProfile*.

7. Continue configuring the remainder of the listener to fit your requirements.

8. Click **Add** to save your new listener with the SSL profile associated to it.

Renew expired client CA certificates

In the case that your client CA certificate has expired, you can update the certificate on your gateway through the following steps:

1. Navigate to your Application Gateway and go to the **SSL settings** tab in the left-hand menu.

2. Select the existing SSL profile(s) with the expired client certificate.
3. Select **Upload a new certificate** in the **Client Authentication** tab and upload your new client certificate.
4. Select the trash can icon next to the expired certificate. This will remove the association of that certificate from the SSL profile.
5. Repeat steps 2-4 above with any other SSL profile that was using the same expired client certificate. You will be able to choose the new certificate you uploaded in step 3 from the dropdown menu in other SSL profiles.

Next steps

- Manage web traffic with an application gateway using the Azure CLI

Configure mutual authentication with Application Gateway through PowerShell

8/22/2022 • 4 minutes to read • Edit Online

This article describes how to use the PowerShell to configure mutual authentication on your Application Gateway. Mutual authentication means Application Gateway authenticates the client sending the request using the client certificate you upload onto the Application Gateway.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This article requires the Azure PowerShell module version 1.0.0 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. If you're running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Before you begin

To configure mutual authentication with an Application Gateway, you need a client certificate to upload to the gateway. The client certificate will be used to validate the certificate the client will present to Application Gateway. For testing purposes, you can use a self-signed certificate. However, this is not advised for production workloads, because they're harder to manage and aren't completely secure.

To learn more, especially about what kind of client certificates you can upload, see Overview of mutual authentication with Application Gateway.

Create a resource group

First create a new resource group in your subscription.

```
$resourceGroup = New-AzResourceGroup -Name $rgname -Location $location -Tags @{ testtag = "APPGw tag"}
```

Create a virtual network

Deploy a virtual network for your Application Gateway to be deployed in.

```
$gwSubnet = New-AzVirtualNetworkSubnetConfig -Name $gwSubnetName -AddressPrefix 10.0.0.0/24
$vnet = New-AzVirtualNetwork -Name $vnetName -ResourceGroupName $rgname -Location $location -AddressPrefix
10.0.0.0/16 -Subnet $gwSubnet
$vnet = Get-AzVirtualNetwork -Name $vnetName -ResourceGroupName $rgname
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name $gwSubnetName -VirtualNetwork $vnet
```

Create a public IP

Create a public IP to use with your Application Gateway.

```
$publicip = New-AzPublicIpAddress -ResourceGroupName $rgname -name $publicIpName -location $location -
AllocationMethod Static -sku Standard
```

Create the Application Gateway IP configuration

Create the IP configurations and frontend port.

```
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name $gipconfigname -Subnet $gwSubnet
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name $fipconfigName -PublicIPAddress $publicip
$port = New-AzApplicationGatewayFrontendPort -Name $frontendPortName  -Port 443
```

Configure frontend SSL

Configure the SSL certificates for your Application Gateway.

```
$password = ConvertTo-SecureString "P@ssw0rd" -AsPlainText -Force
$sslCertPath = $basedir + "/ScenarioTests/Data/ApplicationGatewaySslCert1.pfx"
$sslCert = New-AzApplicationGatewaySslCertificate -Name $sslCertName -CertificateFile $sslCertPath -Password
$password
```

Configure client authentication

Configure client authentication on your Application Gateway. For more information on how to extract trusted client CA certificate chains to use here, see how to extract trusted client CA certificate chains.

> **IMPORTANT**
>
> Please ensure that you upload the entire client CA certificate chain in one file, and only one chain per file.

> **NOTE**
>
> We recommend using TLS 1.2 with mutual authentication as TLS 1.2 will be mandated in the future.

```
$clientCertFilePath = $basedir + "/ScenarioTests/Data/TrustedClientCertificate.cer"
$trustedClient01 = New-AzApplicationGatewayTrustedClientCertificate -Name $trustedClientCert01Name -
CertificateFile $clientCertFilePath
$sslPolicy = New-AzApplicationGatewaySslPolicy -PolicyType Predefined -PolicyName "AppGwSslPolicy20170401S"
$clientAuthConfig = New-AzApplicationGatewayClientAuthConfiguration -VerifyClientCertIssuerDN
$sslProfile01 = New-AzApplicationGatewaySslProfile -Name $sslProfile01Name -SslPolicy $sslPolicy -
ClientAuthConfiguration $clientAuthConfig -TrustedClientCertificates $trustedClient01
$listener = New-AzApplicationGatewayHttpListener -Name $listenerName -Protocol Https -SslCertificate
$sslCert -FrontendIPConfiguration $fipconfig -FrontendPort $port -SslProfile $sslProfile01
```

Configure the backend pool and settings

Set up backend pool and settings for your Application Gateway. Optionally, set up the backend trusted root certificate for end-to-end SSL encryption.

```
$certFilePath = $basedir + "/ScenarioTests/Data/ApplicationGatewayAuthCert.cer"
$trustedRoot = New-AzApplicationGatewayTrustedRootCertificate -Name $trustedRootCertName -CertificateFile
$certFilePath
$pool = New-AzApplicationGatewayBackendAddressPool -Name $poolName -BackendIPAddresses www.microsoft.com,
www.bing.com
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name $poolSettingName -Port 443 -Protocol Https
-CookieBasedAffinity Enabled -PickHostNameFromBackendAddress -TrustedRootCertificate $trustedRoot
```

Configure the rule

Set up a rule on your Application Gateway.

```
$rule = New-AzApplicationGatewayRequestRoutingRule -Name $ruleName -RuleType basic -BackendHttpSettings
$poolSetting -HttpListener $listener -BackendAddressPool $pool
```

Set up default SSL policy for future listeners

You've set up a listener specific SSL policy while setting up mutual authentication. In this step, you can optionally set the default SSL policy for future listeners you create.

```
$sslPolicyGlobal = New-AzApplicationGatewaySslPolicy -PolicyType Predefined -PolicyName
"AppGwSslPolicy20170401"
```

Create the Application Gateway

Using everything we created above, deploy your Application Gateway.

```
$sku = New-AzApplicationGatewaySku -Name Standard_v2 -Tier Standard_v2
$appgw = New-AzApplicationGateway -Name $appgwName -ResourceGroupName $rgname -Zone 1,2 -Location $location
-BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting -FrontendIpConfigurations $fipconfig
-GatewayIpConfigurations $gipconfig -FrontendPorts $port -HttpListeners $listener -RequestRoutingRules $rule
-Sku $sku -SslPolicy $sslPolicyGlobal -TrustedRootCertificate $trustedRoot -AutoscaleConfiguration
$autoscaleConfig -TrustedClientCertificates $trustedClient01 -SslProfiles $sslProfile01 -SslCertificates
$sslCert
```

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name $rgname
```

Renew expired client CA certificates

In the case that your client CA certificate has expired, you can update the certificate on your gateway through the following steps:

1. Sign in to Azure

   ```
   Connect-AzAccount
   Select-AzSubscription -Subscription "<sub name>"
   ```

2. Get your Application Gateway configuration

```
$gateway = Get-AzApplicationGateway -Name "<gateway-name>" -ResourceGroupName "<resource-group-name>"
```

3. Remove the trusted client certificate from the gateway

```
Remove-AzApplicationGatewayTrustedClientCertificate -Name "<name-of-client-certificate>" -
ApplicationGateway $gateway
```

4. Add the new certificate onto the gateway

```
Add-AzApplicationGatewayTrustedClientCertificate -ApplicationGateway $gateway -Name "<name-of-new-
cert>" -CertificateFile "<path-to-certificate-file>"
```

5. Update the gateway with the new certificate

```
Set-AzApplicationGateway -ApplicationGateway $gateway
```

Next steps

- Manage web traffic with an application gateway using the Azure CLI

Create certificates to allow the backend with Azure Application Gateway

8/22/2022 • 3 minutes to read • Edit Online

To do end to end TLS, Application Gateway requires the backend instances to be allowed by uploading authentication/trusted root certificates. For the v1 SKU, authentication certificates are required, but for the v2 SKU trusted root certificates are required to allow the certificates.

In this article, you learn how to:

- Export authentication certificate from a backend certificate (for v1 SKU)
- Export trusted root certificate from a backend certificate (for v2 SKU)

Prerequisites

An existing backend certificate is required to generate the authentication certificates or trusted root certificates required for allowing backend instances with Application Gateway. The backend certificate can be the same as the TLS/SSL certificate or different for added security. Application Gateway doesn't provide you any mechanism to create or purchase a TLS/SSL certificate. For testing purposes, you can create a self-signed certificate but you shouldn't use it for production workloads.

Export authentication certificate (for v1 SKU)

An authentication certificate is required to allow backend instances in Application Gateway v1 SKU. The authentication certificate is the public key of backend server certificates in Base-64 encoded X.509(.CER) format. In this example, you'll use a TLS/SSL certificate for the backend certificate and export its public key to be used as authentication certification. Also, in this example, you'll use the Windows Certificate Manager tool to export the required certificates. You can choose to use any other tool that is convenient.

From your TLS/SSL certificate, export the public key .cer file (not the private key). The following steps help you export the .cer file in Base-64 encoded X.509(.CER) format for your certificate:

1. To obtain a .cer file from the certificate, open **Manage user certificates**. Locate the certificate, typically in 'Certificates - Current User\Personal\Certificates', and right-click. Click **All Tasks**, and then click **Export**. This opens the **Certificate Export Wizard**. If you can't find the certificate under Current User\Personal\Certificates, you may have accidentally opened "Certificates - Local Computer", rather than "Certificates - Current User"). If you want to open Certificate Manager in current user scope using PowerShell, you type *certmgr* in the console window.

2. In the Wizard, click **Next**.

3. Select **No, do not export the private key**, and then click **Next**.

4. On the **Export File Format** page, select **Base-64 encoded X.509 (.CER).**, and then click **Next**.

5. For **File to Export**, **Browse** to the location to which you want to export the certificate. For **File name**, name the certificate file. Then, click **Next**.

6. Click **Finish** to export the certificate.

7. Your certificate is successfully exported.

The exported certificate looks similar to this:

8. If you open the exported certificate using Notepad, you see something similar to this example. The section in blue contains the information that is uploaded to application gateway. If you open your

certificate with Notepad and it doesn't look similar to this, typically this means you didn't export it using the Base-64 encoded X.509(.CER) format. Additionally, if you want to use a different text editor, understand that some editors can introduce unintended formatting in the background. This can create problems when uploaded the text from this certificate to Azure.

Export trusted root certificate (for v2 SKU)

Trusted root certificate is required to allow backend instances in application gateway v2 SKU. The root certificate is a Base-64 encoded X.509(.CER) format root certificate from the backend server certificates. In this example, we will use a TLS/SSL certificate for the backend certificate, export its public key and then export the root certificate of the trusted CA from the public key in base64 encoded format to get the trusted root certificate. The intermediate certificate(s) should be bundled with server certificate and installed on the backend server.

The following steps help you export the .cer file for your certificate:

1. Use the steps 1 - 8 mentioned in the previous section Export authentication certificate (for v1 SKU) to export the public key from your backend certificate.

2. Once the public key has been exported, open the file.

3. Move to the Certification Path view to view the certification authority.

4. Select the root certificate and click on **View Certificate**.

You should see the root certificate details.

5. Move to the **Details** view and click **Copy to File...**

6. At this point, you've extracted the details of the root certificate from the backend certificate. You'll see the

Certificate Export Wizard. Now use steps 2-9 mentioned in the section **Export authentication certificate from a backend certificate (for v1 SKU)** above to export the trusted root certificate in the Base-64 encoded X.509(.CER) format.

Next steps

Now you have the authentication certificate/trusted root certificate in Base-64 encoded X.509(.CER) format. You can add this to the application gateway to allow your backend servers for end to end TLS encryption. See Configure end to end TLS by using Application Gateway with PowerShell.

Renew Application Gateway certificates

8/22/2022 • 2 minutes to read • Edit Online

At some point, you'll need to renew your certificates if you configured your application gateway for TLS/SSL encryption.

There are two locations where certificates may exist: certificates stored in Azure Key Vault, or certificates uploaded to an application gateway.

Certificates on Azure Key Vault

When Application Gateway is configured to use Key Vault certificates, its instances retrieve the certificate from Key Vault and install them locally for TLS termination. The instances poll Key Vault at four-hour intervals to retrieve a renewed version of the certificate if it exists. If an updated certificate is found, the TLS/SSL certificate that's currently associated with the HTTPS listener is automatically rotated.

> **TIP**
>
> Any change to Application Gateway will force a check against Key Vault to see if any new versions of certificates are available. This includes, but is not limited to, changes to Frontend IP Configurations, Listeners, Rules, Backend Pools, Resource Tags, and more. If an updated certificate is found, the new certificate will immediately be presented.

Application Gateway uses a secret identifier in Key Vault to reference the certificates. For Azure PowerShell, the Azure CLI, or Azure Resource Manager, we strongly recommend that you use a secret identifier that doesn't specify a version. This way, Application Gateway will automatically rotate the certificate if a newer version is available in your key vault. An example of a secret URI without a version is `https://myvault.vault.azure.net/secrets/mysecret/`.

Certificates on an application gateway

Application Gateway supports certificate upload without the need to configure Azure Key Vault. To renew the uploaded certificates, use the following steps for the Azure portal, Azure PowerShell, or Azure CLI.

Azure portal

To renew a listener certificate from the portal, navigate to your application gateway listeners. Select the listener that has a certificate that needs to be renewed, and then select **Renew or edit selected certificate**.

Upload your new PFX certificate, give it a name, type the password, and then select **Save**.

Azure PowerShell

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

To renew your certificate using Azure PowerShell, use the following script:

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName <ResourceGroup> `
  -Name <AppGatewayName>

$password = ConvertTo-SecureString `
  -String "<password>" `
  -Force `
  -AsPlainText

set-AzApplicationGatewaySSLCertificate -Name <oldcertname> `
-ApplicationGateway $appgw -CertificateFile <newcertPath> -Password $password

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Azure CLI

```
az network application-gateway ssl-cert update \
  -n "<CertName>" \
  --gateway-name "<AppGatewayName>" \
  -g "ResourceGroupName>" \
  --cert-file <PathToCerFile> \
  --cert-password "<password>"
```

Next steps

To learn how to configure TLS Offloading with Azure Application Gateway, see Configure TLS Offload.

Generate an Azure Application Gateway self-signed certificate with a custom root CA

8/22/2022 • 6 minutes to read • Edit Online

The Application Gateway v2 SKU introduces the use of Trusted Root Certificates to allow backend servers. This removes authentication certificates that were required in the v1 SKU. The *root certificate* is a Base-64 encoded X.509(.CER) format root certificate from the backend certificate server. It identifies the root certificate authority (CA) that issued the server certificate and the server certificate is then used for the TLS/SSL communication.

Application Gateway trusts your website's certificate by default if it's signed by a well-known CA (for example, GoDaddy or DigiCert). You don't need to explicitly upload the root certificate in that case. For more information, see Overview of TLS termination and end to end TLS with Application Gateway. However, if you have a dev/test environment and don't want to purchase a verified CA signed certificate, you can create your own custom CA and create a self-signed certificate with it.

> **NOTE**
>
> Self-signed certificates are not trusted by default and they can be difficult to maintain. Also, they may use outdated hash and cipher suites that may not be strong. For better security, purchase a certificate signed by a well-known certificate authority.

In this article, you will learn how to:

- Create your own custom Certificate Authority
- Create a self-signed certificate signed by your custom CA
- Upload a self-signed root certificate to an Application Gateway to authenticate the backend server

Prerequisites

- **OpenSSL on a computer running Windows or Linux**

  While there could be other tools available for certificate management, this tutorial uses OpenSSL. You can find OpenSSL bundled with many Linux distributions, such as Ubuntu.

- **A web server**

  For example, Apache, IIS, or NGINX to test the certificates.

- **An Application Gateway v2 SKU**

  If you don't have an existing application gateway, see Quickstart: Direct web traffic with Azure Application Gateway - Azure portal.

Create a root CA certificate

Create your root CA certificate using OpenSSL.

Create the root key

1. Sign in to your computer where OpenSSL is installed and run the following command. This creates an encrypted key.

```
openssl ecparam -out contoso.key -name prime256v1 -genkey
```

Create a Root Certificate and self-sign it

1. Use the following command to generate the Certificate Signing Request (CSR).

```
openssl req -new -sha256 -key contoso.key -out contoso.csr
```

2. When prompted, type the password for the root key, and the organizational information for the custom CA such as Country/Region, State, Org, OU, and the fully qualified domain name (this is the domain of the issuer).

3. Use the following command to generate the Root Certificate.

```
openssl x509 -req -sha256 -days 365 -in contoso.csr -signkey contoso.key -out contoso.crt
```

The previous commands create the root certificate. You'll use this to sign your server certificate.

Create a server certificate

Next, you'll create a server certificate using OpenSSL.

Create the certificate's key

Use the following command to generate the key for the server certificate.

```
openssl ecparam -out fabrikam.key -name prime256v1 -genkey
```

Create the CSR (Certificate Signing Request)

The CSR is a public key that is given to a CA when requesting a certificate. The CA issues the certificate for this specific request.

> **NOTE**
> The CN (Common Name) for the server certificate must be different from the issuer's domain. For example, in this case, the CN for the issuer is `www.contoso.com` and the server certificate's CN is `www.fabrikam.com`.

1. Use the following command to generate the CSR:

```
openssl req -new -sha256 -key fabrikam.key -out fabrikam.csr
```

2. When prompted, type the password for the root key, and the organizational information for the custom CA: Country/Region, State, Org, OU, and the fully qualified domain name. This is the domain of the website and it should be different from the issuer.

```
surajmb@testlboutvm1:~$ openssl req -new -sha256 -key fabrikam.key -out fabrikam.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Telangana
Locality Name (eg, city) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MS
Organizational Unit Name (eg, section) []:IDC
Common Name (e.g. server FQDN or YOUR name) []:www.fabrikam.com
Email Address []:user@fabrikam.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:a1b2c3d4
```

Generate the certificate with the CSR and the key and sign it with the CA's root key

1. Use the following command to create the certificate:

```
openssl x509 -req -in fabrikam.csr -CA  contoso.crt -CAkey contoso.key -CAcreateserial -out
fabrikam.crt -days 365 -sha256
```

Verify the newly created certificate

1. Use the following command to print the output of the CRT file and verify its content:

```
openssl x509 -in fabrikam.crt -text -noout
```

```
surajmb@testlboutvm1:~$ openssl x509 -in fabrikam.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            96:73:88:67:fc:a6:d4:4c
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = US, ST = Texas, L = Austin, O = MS, OU = DC, CN = www.contoso.com, emailAddress = user@contoso.com
        Validity
            Not Before: May 20 04:49:30 2019 GMT
            Not After : May 19 04:49:30 2020 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = MS, OU = IDC, CN = www.fabrikam.com, emailAddress = user@fabrikam.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:56:38:e4:07:ca:bc:72:24:26:98:f2:68:17:21:
```

2. Verify the files in your directory, and ensure you have the following files:

- contoso.crt
- contoso.key
- fabrikam.crt
- fabrikam.key

Configure the certificate in your web server's TLS settings

In your web server, configure TLS using the fabrikam.crt and fabrikam.key files. If your web server can't take two files, you can combine them to a single .pem or .pfx file using OpenSSL commands.

IIS

For instructions on how to import certificate and upload them as server certificate on IIS, see HOW TO: Install Imported Certificates on a Web Server in Windows Server 2003.

For TLS binding instructions, see How to Set Up SSL on IIS 7.

Apache

The following configuration is an example virtual host configured for SSL in Apache:

```
<VirtualHost www.fabrikam:443>
      DocumentRoot /var/www/fabrikam
      ServerName www.fabrikam.com
      SSLEngine on
      SSLCertificateFile /home/user/fabrikam.crt
      SSLCertificateKeyFile /home/user/fabrikam.key
</VirtualHost>
```

NGINX

The following configuration is an example NGINX server block with TLS configuration:

```
 listen 443 ssl default_server;
listen [::]:443 ssl default_server;
ssl_certificate     /home/surajmb/fabrikam.crt;
ssl_certificate_key /home/surajmb/fabrikam.key;
ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers         HIGH:!aNULL:!MD5;
```

Access the server to verify the configuration

1. Add the root certificate to your machine's trusted root store. When you access the website, ensure the entire certificate chain is seen in the browser.

> **NOTE**
>
> It's assumed that DNS has been configured to point the web server name (in this example, www.fabrikam.com) to your web server's IP address. If not, you can edit the hosts file to resolve the name.

2. Browse to your website, and click the lock icon on your browser's address box to verify the site and certificate information.

Verify the configuration with OpenSSL

Or, you can use OpenSSL to verify the certificate.

```
openssl s_client -connect localhost:443 -servername www.fabrikam.com -showcerts
```

```
surajmb@testlboutvm1:~$ openssl s_client -connect localhost:443 -servername www.fabrikam.com -showcerts
CONNECTED(00000003)
depth=1 C = US, ST = Texas, L = Austin, O = MS, OU = DC, CN = www.contoso.com, emailAddress = user@contoso.com
verify return:1
depth=0 C = IN, ST = Telangana, L = Hyderabad, O = MS, OU = IDC, CN = www.fabrikam.com, emailAddress = user@fabrikam.comverify return:1
---
Certificate chain
 0 s:/C=IN/ST=Telangana/L=Hyderabad/O=MS/OU=IDC/CN=www.fabrikam.com/emailAddress=user@fabrikam.com
   i:/C=US/ST=Texas/L=Austin/O=MS/OU=DC/CN=www.contoso.com/emailAddress=user@contoso.com
-----BEGIN CERTIFICATE-----
MIICAzCCAagCCQCWc4hn/KbUTDAKBggqhkjOPQQDAjCBgzELMAkGA1UEBhMCVVMx
DjAMBgNVBAgMBVRleGFzMQ8wDQYDVQQHDAZBdXN0aW4xCzAJBgNVBAoMAk1TMQsw
```

Upload the root certificate to Application Gateway's HTTP Settings

To upload the certificate in Application Gateway, you must export the .crt certificate into a .cer format Base-64 encoded. Since .crt already contains the public key in the base-64 encoded format, just rename the file extension from .crt to .cer.

Azure portal

To upload the trusted root certificate from the portal, select the **HTTP Settings** and choose the **HTTPS** protocol.

Azure PowerShell

Or, you can use Azure CLI or Azure PowerShell to upload the root certificate. The following code is an Azure PowerShell sample.

> **NOTE**
>
> The following sample adds a trusted root certificate to the application gateway, creates a new HTTP setting and adds a new rule, assuming the backend pool and the listener exist already.

```
## Add the trusted root certificate to the Application Gateway

$gw=Get-AzApplicationGateway -Name appgwv2 -ResourceGroupName rgOne

Add-AzApplicationGatewayTrustedRootCertificate `
   -ApplicationGateway $gw `
   -Name CustomCARoot `
   -CertificateFile "C:\Users\surmb\Downloads\contoso.cer"

$trustedroot = Get-AzApplicationGatewayTrustedRootCertificate `
   -Name CustomCARoot `
   -ApplicationGateway $gw

## Get the listener, backend pool and probe

$listener = Get-AzApplicationGatewayHttpListener `
   -Name basichttps `
   -ApplicationGateway $gw

$bepool = Get-AzApplicationGatewayBackendAddressPool `
  -Name testbackendpool `
  -ApplicationGateway $gw

Add-AzApplicationGatewayProbeConfig `
  -ApplicationGateway $gw `
  -Name testprobe `
  -Protocol Https `
  -HostName "www.fabrikam.com" `
  -Path "/" `
  -Interval 15 `
  -Timeout 20 `
  -UnhealthyThreshold 3

$probe = Get-AzApplicationGatewayProbeConfig `
  -Name testprobe `
  -ApplicationGateway $gw

## Add the configuration to the HTTP Setting and don't forget to set the "hostname" field
## to the domain name of the server certificate as this will be set as the SNI header and
## will be used to verify the backend server's certificate. Note that TLS handshake will
## fail otherwise and might lead to backend servers being deemed as Unhealthy by the probes

Add-AzApplicationGatewayBackendHttpSettings `
  -ApplicationGateway $gw `
  -Name testbackend `
  -Port 443 `
  -Protocol Https `
  -Probe $probe `
  -TrustedRootCertificate $trustedroot `
  -CookieBasedAffinity Disabled `
  -RequestTimeout 20 `
  -HostName www.fabrikam.com

## Get the configuration and update the Application Gateway

$backendhttp = Get-AzApplicationGatewayBackendHttpSettings `
  -Name testbackend `
  -ApplicationGateway $gw

Add-AzApplicationGatewayRequestRoutingRule `
  -ApplicationGateway $gw `
  -Name testrule `
  -RuleType Basic `
  -BackendHttpSettings $backendhttp `
  -HttpListener $listener `
  -BackendAddressPool $bepool

Set-AzApplicationGateway -ApplicationGateway $gw
```

Verify the application gateway backend health

1. Click the **Backend Health** view of your application gateway to check if the probe is healthy.
2. You should see that the Status is **Healthy** for the HTTPS probe.

Next steps

To learn more about SSL\TLS in Application Gateway, see Overview of TLS termination and end to end TLS with Application Gateway.

Export a trusted client CA certificate chain to use with client authentication

8/22/2022 • 3 minutes to read • Edit Online

In order to configure mutual authentication with the client, or client authentication, Application Gateway requires a trusted client CA certificate chain to be uploaded to the gateway. If you have multiple certificate chains, you'll need to create the chains separately and upload them as different files on the Application Gateway. In this article, you'll learn how to export a trusted client CA certificate chain that you can use in your client authentication configuration on your gateway.

Prerequisites

An existing client certificate is required to generate the trusted client CA certificate chain.

Export trusted client CA certificate

Trusted client CA certificate is required to allow client authentication on Application Gateway. In this example, we will use a TLS/SSL certificate for the client certificate, export its public key and then export the CA certificates from the public key to get the trusted client CA certificates. We'll then concatenate all the client CA certificates into one trusted client CA certificate chain.

The following steps help you export the .pem or .cer file for your certificate:

Export public certificate

1. To obtain a .cer file from the certificate, open **Manage user certificates**. Locate the certificate, typically in 'Certificates - Current User\Personal\Certificates', and right-click. Click **All Tasks**, and then click **Export**. This opens the **Certificate Export Wizard**. If you can't find the certificate under Current User\Personal\Certificates, you may have accidentally opened "Certificates - Local Computer", rather than "Certificates - Current User"). If you want to open Certificate Manager in current user scope using PowerShell, you type *certmgr* in the console window.

2. In the Wizard, click **Next**.

3. Select **No, do not export the private key**, and then click **Next**.

4. On the **Export File Format** page, select **Base-64 encoded X.509 (.CER)**., and then click **Next**.

5. For **File to Export**, **Browse** to the location to which you want to export the certificate. For **File name**, name the certificate file. Then, click **Next**.

6. Click **Finish** to export the certificate.

7. Your certificate is successfully exported.

The exported certificate looks similar to this:

Export CA certificate(s) from the public certificate

Now that you've exported your public certificate, you will now export the CA certificate(s) from your public

certificate. If you only have a root CA, you'll only need to export that certificate. However, if you have 1+ intermediate CAs, you'll need to export each of those as well.

1. Once the public key has been exported, open the file.

2. Select the Certification Path tab to view the certification authority.

3. Select the root certificate and click on **View Certificate**.

You should see the root certificate details.

4. Select the **Details** tab and click **Copy to File...**

5. At this point, you've extracted the details of the root CA certificate from the public certificate. You'll see the **Certificate Export Wizard**. Follow steps 2-7 from the previous section (Export public certificate) to complete the Certificate Export Wizard.

6. Now repeat steps 2-6 from this current section (Export CA certificate(s) from the public certificate) for all intermediate CAs to export all intermediate CA certificates in the Base-64 encoded X.509(.CER) format.

For example, you would repeat steps 2-6 from this section on the *MSIT CAZ2* intermediate CA to extract it as its own certificate.

Concatenate all your CA certificates into one file

1. Run the following command with all the CA certificates you extracted earlier.

   Windows:

   ```
   type intermediateCA.cer rootCA.cer > combined.cer
   ```

   Linux:

   ```
   cat intermediateCA.cer rootCA.cer >> combined.cer
   ```

   Your resulting combined certificate should look something like the following:

combined.cer - Notepad

File Edit Format View Help


```
-----BEGIN CERTIFICATE-----
MIIHGjCCBQKgAwIBAgITZQAAAF7VTdBaviGEtQAAAAAAXjANBgkqhkiG9w0BAQsF
ADAsMSowKAYDVQQDEyFNaWNyb3NvZnQgSW50ZXJuYWwgQ29ycG9yYXRlIFJvb3Qw
HhcNMjAwNDIzMTgxNzQ5WhcNMjQwNDI0MTgxNzQ5WjAVMRMwEQYDVQQDEwpNU0lU
IENBIFoyMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsIfN3ufSMGNv
mBErdTfWAIsKFZfybdlucHBaG5iJaXAMwfOT70ix8H6KT6yzXe2H35/n8F0eEKrs
tgEZn3p7LMGxL0O+fVWeT82VVznFo6E0e2iePSF7apE7eKmxRhnSEJWHSO9nEJxe
bFCawweS2HZ/NwhS9feZLUWei74u+Sh0AYOLIWu7RZXmDEspdTvs2LxpRUy8lfbc
yy0HW4LH2CX8dgZBEjSv4LNiBlBr5uDqOHXYyIHMM9ReEaVppDmXAMZnBtXGHzhQ
RRVXJASucvhV4lKHt6IkHizQsniBPN9y8eFp8p2kAHYiB4XBns+o+NbHHYqHdpBV
1DhmSY0KihhgCyTRI+8W7fOh++73UO2l9wziDPV9F1HjW3gguTdZ4ieDVl0MMhR1
F7E7PMK+axGewGO2eguOinmWOmXgO0k00RSNru61sNObxxGV2de/NAnKIIi/70CU
yasNxxsDRnTOkunFdZh0RxFrTlR4xNNyTtROfWGRIACATBjvBJsJpx+Ce66Kzold
EUKzHKdtflLzBFdIggiEY7rQuQqjj4zibNgKDAKwNj0oDrPFkBuAlm+0ftDYtowT
p78BxgsrUDXFCs2ZUkt8GADgzONoQakyAjAdIDqt0f66zBEleNHOLB1FpEp4pJDL
4+hIm2WG5RXpzZxbq/OnQchkJuuk9ZcCAwEAAaOCAkowggJGMBIGCSsGAQQBgjcV
AQQFAgMCAAIwIwYJKwYBBAGCNxUCBBYEFDerFe2jVmREjAD7MPAKdth/6ahVMB0G
A1UdDgQWBBTSMoT753PGyeDd5Ju5RZlDZBgUeDBpBgNVHSUEYjBgBggrBgEFBQcD
AwYIKwYBBQUHAw4GCCsGAQUFBwMBBgcrBgEFAgMFBgorBgEEAYI3FAICBgkrBgEE
AYI3FQUGCCsGAQUFBwMCBgorBgEEAYI3KgIFBgorBgEEAYI3KgIGMBkGCSsGAQQB
gjcUAgQMHgoAUwB1AGIAQwBBMAsGA1UdDwQEAwIBhjASBgNVHRMBAf8ECDAGAQH/
AgEAMB8GA1UdIwQYMBaAFNvsCmSozsAO6YhUDunI3sR4S3phMIGmBgNVHR8EgZ4w
gZswgZiggZWggZKGIGh0dHA6Ly9jb3JwcGtpL2NybC9tc2ludGNyY2EuY3Jshjdo
dHRwOi8vbXNjcmwubWljcm9zb2Z0LmNvbS9wa2kvbXNjb3JwL2NybC9tc2ludGNy
Y2EuY3JshjVodHRwOi8vY3JsLm1pY3Jvc29mdC5jb20vcGtpL21zY29ycC9jcmwv
bXNpbnRjcmNhLmNybDB7BggrBgEFBQcBAQRvMG0wLAYIKwYBBQUHMAKGIGh0dHA6
Ly9jb3JwcGtpL2FpYS9tc2ludGNyY2EuY3J0MD0GCCsGAQUFBzAChjFodHRwOi8v
d3d3Lm1pY3Jvc29mdC5jb20vcGtpL21zY29ycC9tc2ludGNyY2EuY3J0MA0GCSqG
SIb3DQEBCwUAA4ICAQAwkaN12OgzXkqjLDsafH0xfRPR741b4l+7CWDtpl3i8EhN
00E+RwxMW8CpJOt4R3JUfX2KpFujiPchRPrPf2QljDU+iM78opr8Ab7iv/ufP2wH
RNeqCKNp57wHbMmqvNf3C98B057mrjbtWW1fwPVhqAzvXDIWy5mOqG4ls7wsvNCw
2RrgLbhzgftqIlWDVKkX7KUpKC1+TTXBBJ2QcK+6JePb48dDjlvAwkRU7xZUMnQW
P3mCrdOTLAtGnUkKNGNS3t15qDCIWofZlDQipuqQ0oQSQaZGHp1wo4A94/bV7EH+
sx6GaqSyi++V/Zw3HYnB3N69tIrNPFZ6GSZvg7or7ZEaXdNhu1CnEF+aFB1MO3ox
M4lOxhkZmRUZCs4HB2YS6BcZ9gIepa5wSkidR6NqWBhk6wX6d6BdzeMsOjVNoXH2
e4rbvIu+WcEwnl2aL4PdFFaPfTT2EeKYBD2ksVKBLlJIxAhWP0UC7MNKx70/KBhy
N6oFvFmRmu6jW3XIRsABAU6bQNbLnSaKFasZ5vWoZO9eeaMUQjXGpsSUi/mO2PPL
V8ecVNheg/cmihviW10Mkf1bWMd5W+bN4TutDiF3vcF+OlS54cL7W2vRDrAe3YDd
6qISXP8UkJf7B4D5qm+iroDzV9gIZv4OOD0rEo38xItoGlazuwpQl2dLO/UArQ==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFsDCCA5igAwIBAgIQCx8BEem5Cp1K2NZtNZ9wADANBgkqhkiG9w0BAQsFADAs
MSowKAYDVQQDEyFNaWNyb3NvZnQgSW50ZXJuYWwgQ29ycG9yYXRlIFJvb3QwHhcN
MTIwNDA1MjE1NDU0WhcNMzcwNDA1MjIwMTA1WjAsMSowKAYDVQQDEyFNaWNyb3Nv
ZnQgSW50ZXJuYWwgQ29ycG9yYXRlIFJvb3QwggIiMA0GCSqGSIb3DQEBAQUAA4IC
DwAwggIKAoICAQCV2ldlrpoImQLgnSMRmPCan4PHEw+Gxd2JecDmnas8ZF1VMj5M
7qhKnahwIJCoasPjnYORprpbkUZrAMM+yp3aQ+Ob7qn2d1EVxRduozSnQ1siu5DS
tXGIPv1z19h2wJ2T6uZEde59Vy3tyBfekYkaq9+d4Zh+ie+j6mvaufPG0Job7FZs
G9B2KfjggLXiT5OnOqGgB/ODk+OtWJQyUCXyCUHgUwg8d+ZhHTZqfKZL5E6riz7f
tgo707/EiySCJe8L4uXJSOqfCBgrzQyOik0Uh811DFmGPF0pT42RWUZGLSmAGxuM
pSRZcCkvuhfeSczeR/hwCQiiLcoM1c6apoFtipq7B0NhpL29SE9KyGFQ0BwlH+HI
```


Next steps

Now you have the trusted client CA certificate chain. You can add this to your client authentication configuration on the Application Gateway to allow mutual authentication with your gateway. See configure mutual authentication using Application Gateway with Portal or configure mutual authentication using Application Gateway with PowerShell.

Configure TLS policy versions and cipher suites on Application Gateway

8/22/2022 • 5 minutes to read • Edit Online

Learn how to configure TLS/SSL policy versions and cipher suites on Application Gateway. You can select from a list of predefined policies that contain different configurations of TLS policy versions and enabled cipher suites. You also have the ability to define a custom TLS policy based on your requirements.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

> **NOTE**
>
> We recommend using TLS 1.2 as your minimum TLS protocol version for better security on your Application Gateway.

Get available TLS options

The `Get-AzApplicationGatewayAvailableSslOptions` cmdlet provides a listing of available pre-defined policies, available cipher suites, and protocol versions that can be configured. The following example shows an example output from running the cmdlet.

```
DefaultPolicy: AppGwSslPolicy20150501
PredefinedPolicies:
    /subscriptions/xxx-
xxx/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/Applic
ationGatewaySslPredefinedPolicy/AppGwSslPolicy20150501
    /subscriptions/xxx-
xxx/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/Applic
ationGatewaySslPredefinedPolicy/AppGwSslPolicy20170401
    /subscriptions/xxx-
xxx/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/Applic
ationGatewaySslPredefinedPolicy/AppGwSslPolicy20170401S
    /subscriptions/xxx-
xxx/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/Applic
ationGatewaySslPredefinedPolicy/AppGwSslPolicy20220101
    /subscriptions/xxx-
xxx/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/Applic
ationGatewaySslPredefinedPolicy/AppGwSslPolicy20220101S

AvailableCipherSuites:
    TLS_AES_128_GCM_SHA256
    TLS_AES_256_GCM_SHA384
    TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_DHE_RSA_WITH_AES_256_CBC_SHA
    TLS_DHE_RSA_WITH_AES_128_CBC_SHA
    TLS_RSA_WITH_AES_256_GCM_SHA384
    TLS_RSA_WITH_AES_128_GCM_SHA256
    TLS_RSA_WITH_AES_256_CBC_SHA256
    TLS_RSA_WITH_AES_128_CBC_SHA256
    TLS_RSA_WITH_AES_256_CBC_SHA
    TLS_RSA_WITH_AES_128_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
    TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
    TLS_DHE_DSS_WITH_AES_256_CBC_SHA
    TLS_DHE_DSS_WITH_AES_128_CBC_SHA
    TLS_RSA_WITH_3DES_EDE_CBC_SHA
    TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

AvailableProtocols:
    TLSv1_0
    TLSv1_1
    TLSv1_2
    TLSv1_3
```

List pre-defined TLS Policies

Application gateway comes with multiple pre-defined policies that can be used. The `Get-AzApplicationGatewaySslPredefinedPolicy` cmdlet retrieves these policies. Each policy has different protocol versions and cipher suites enabled. These pre-defined policies can be used to quickly configure a TLS policy on your application gateway. By default **AppGwSslPolicy20150501** is selected if no specific TLS policy is defined.

The following output is an example of running `Get-AzApplicationGatewaySslPredefinedPolicy`.

```
Name: AppGwSslPolicy20150501
MinProtocolVersion: TLSv1_0
CipherSuites:
    TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_DHE_RSA_WITH_AES_256_CBC_SHA
    TLS_DHE_RSA_WITH_AES_128_CBC_SHA
    TLS_RSA_WITH_AES_256_GCM_SHA384
 ...
Name: AppGwSslPolicy20170401
MinProtocolVersion: TLSv1_1
CipherSuites:
    TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
...
```

Configure a custom TLS policy

When configuring a custom TLS policy, you pass the following parameters: PolicyType, MinProtocolVersion, CipherSuite, and ApplicationGateway. If you attempt to pass other parameters, you get an error when creating or updating the Application Gateway. The following example sets a custom TLS policy on an application gateway. It sets the minimum protocol version to `TLSv1_1` and enables the following cipher suites:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

```
# get an application gateway resource
$gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroup AdatumAppGatewayRG

# set the TLS policy on the application gateway
Set-AzApplicationGatewaySslPolicy -ApplicationGateway $gw -PolicyType Custom -MinProtocolVersion TLSv1_1 -
CipherSuite "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256", "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"

# validate the TLS policy locally
Get-AzApplicationGatewaySslPolicy -ApplicationGateway $gw

# update the gateway with validated TLS policy
Set-AzApplicationGateway -ApplicationGateway $gw
```

IMPORTANT

- If you are using a custom SSL policy in Application Gateway v1 SKU (Standard or WAF), make sure that you add the mandatory cipher "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256" to the list. This cipher is required to enable metrics and logging in the Application Gateway v1 SKU. This is not mandatory for Application Gateway v2 SKU (Standard_v2 or WAF_v2).
- Cipher suites "TLS_AES_128_GCM_SHA256" and "TLS_AES_256_GCM_SHA384" with TLSv1.3 are not customizable and included by default when setting a CustomV2 policy with a minimum TLS version of 1.2 or 1.3. These two cipher suites will not appear in the Get Details output, with an exception of Portal.

To set minimum protocol version to 1.3, you must use the following command:

```
Set-AzApplicationGatewaySslPolicy -ApplicationGateway $AppGW -MinProtocolVersion TLSv1_3 -PolicyType
CustomV2 -CipherSuite @()
```

This illustration further explains the usage of CustomV2 policy with minimum protocol versions 1.2 and 1.3.

Examples:

Set-AzApplicationGatewaySslPolicy -ApplicationGateway $AppGW

-MinProtocolVersion **TLSv1_2** -PolicyType **CustomV2** -CipherSuite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
OR
-MinProtocolVersion **TLSv1_3** -PolicyType **CustomV2** -CipherSuite @()

Do not specify mandatory ciphers

Create an application gateway with a pre-defined TLS policy

When configuring a Predefined TLS policy, you pass the following parameters: PolicyType, PolicyName, and ApplicationGateway. If you attempt to pass other parameters, you get an error when creating or updating the Application Gateway.

The following example creates a new application gateway with a pre-defined TLS policy.

```
# Create a resource group
$rg = New-AzResourceGroup -Name ContosoRG -Location "East US"

# Create a subnet for the application gateway
$subnet = New-AzVirtualNetworkSubnetConfig -Name subnet01 -AddressPrefix 10.0.0.0/24

# Create a virtual network with a 10.0.0.0/16 address space
$vnet = New-AzVirtualNetwork -Name appgwvnet -ResourceGroupName $rg.ResourceGroupName -Location "East US" -
AddressPrefix 10.0.0.0/16 -Subnet $subnet

# Retrieve the subnet object for later use
$subnet = $vnet.Subnets[0]

# Create a public IP address
$publicip = New-AzPublicIpAddress -ResourceGroupName $rg.ResourceGroupName -name publicIP01 -location "East
US" -AllocationMethod Dynamic

# Create an ip configuration object
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name gatewayIP01 -Subnet $subnet

# Create a backend pool for backend web servers
$pool = New-AzApplicationGatewayBackendAddressPool -Name pool01 -BackendIPAddresses 134.170.185.46,
134.170.188.221,134.170.185.50

# Define the backend http settings to be used.
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name poolsetting01 -Port 80 -Protocol Http -
CookieBasedAffinity Enabled

# Create a new port for TLS
$fp = New-AzApplicationGatewayFrontendPort -Name frontendport01  -Port 443

# Upload an existing pfx certificate for TLS offload
$password = ConvertTo-SecureString -String "P@ssw0rd" -AsPlainText -Force
$cert = New-AzApplicationGatewaySslCertificate -Name cert01 -CertificateFile C:\folder\contoso.pfx -Password
$password

# Create a frontend IP configuration for the public IP address
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name fipconfig01 -PublicIPAddress $publicip

# Create a new listener with the certificate, port, and frontend ip.
$listener = New-AzApplicationGatewayHttpListener -Name listener01  -Protocol Https -FrontendIPConfiguration
$fipconfig -FrontendPort $fp -SslCertificate $cert

# Create a new rule for backend traffic routing
$rule = New-AzApplicationGatewayRequestRoutingRule -Name rule01 -RuleType Basic -BackendHttpSettings
$poolSetting -HttpListener $listener -BackendAddressPool $pool

# Define the size of the application gateway
$sku = New-AzApplicationGatewaySku -Name Standard_Small -Tier Standard -Capacity 2

# Configure the TLS policy to use a different pre-defined policy
$policy = New-AzApplicationGatewaySslPolicy -PolicyType Predefined -PolicyName AppGwSslPolicy20170401S

# Create the application gateway.
$appgw = New-AzApplicationGateway -Name appgwtest -ResourceGroupName $rg.ResourceGroupName -Location "East
US" -BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting -FrontendIpConfigurations
$fipconfig  -GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners $listener -
RequestRoutingRules $rule -Sku $sku -SslCertificates $cert -SslPolicy $policy
```

Update an existing application gateway with a pre-defined TLS policy

To set a custom TLS policy, pass the following parameters: **PolicyType**, **MinProtocolVersion**, **CipherSuite**, and **ApplicationGateway**. To set a Predefined TLS policy, pass the following parameters: **PolicyType**, **PolicyName**, and **ApplicationGateway**. If you attempt to pass other parameters, you get an error when

creating or updating the Application Gateway.

> **NOTE**
> Using a new Predefined or Customv2 policy enhances SSL security and performance posture of the entire gateway (SSL Policy and SSL Profile). Hence, both old and new policies cannot co-exist. You are required to use any of the older predefined or custom policies across the gateway, in case there are clients requiring older TLS version or ciphers (for example, TLS v1.0).

In the following example, there are code samples for both Custom Policy and Predefined Policy. Uncomment the policy you want to use.

```
# You have to change these parameters to match your environment.
$AppGWname = "YourAppGwName"
$RG = "YourResourceGroupName"

$AppGw = get-Azapplicationgateway -Name $AppGWname -ResourceGroupName $RG

# Choose either custom policy or predefined policy and uncomment the one you want to use.

# TLS Custom Policy
# Set-AzApplicationGatewaySslPolicy -PolicyType Custom -MinProtocolVersion TLSv1_2 -CipherSuite
"TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256", "TLS_RSA_WITH_AES_128_CBC_SHA256" -ApplicationGateway $AppGw

# TLS Predefined Policy
# Set-AzApplicationGatewaySslPolicy -PolicyType Predefined -PolicyName "AppGwSslPolicy20170401S" -
ApplicationGateway $AppGW

# Update AppGW
# The TLS policy options are not validated or updated on the Application Gateway until this cmdlet is
executed.
$SetGW = Set-AzApplicationGateway -ApplicationGateway $AppGW
```

Next steps

Visit Application Gateway redirect overview to learn how to redirect HTTP traffic to an HTTPS endpoint.

Check out setting up listener specific SSL policies at setting up SSL listener specific policy through Portal

Configure listener-specific SSL policies on Application Gateway through portal

8/22/2022 • 3 minutes to read • Edit Online

This article describes how to use the Azure portal to configure listener-specific SSL policies on your Application Gateway. Listener-specific SSL policies allow you to configure specific listeners to use different SSL policies from each other. You'll still be able to set a default SSL policy that all listeners will use unless overwritten by the listener-specific SSL policy.

> **NOTE**
>
> Only Standard_v2 and WAF_v2 SKUs support listener specific policies as listener specific policies are part of SSL profiles, and SSL profiles are only supported on v2 gateways.

If you don't have an Azure subscription, create a free account before you begin.

Create a new Application Gateway

First create a new Application Gateway as you would usually through the portal - there are no additional steps needed in the creation to configure listener-specific SSL policies. For more information on how to create an Application Gateway in portal, check out our portal quickstart tutorial.

Set up a listener-specific SSL policy

Before you proceed, here are some important points related to listener-specific SSL policy.

- We recommend using TLS 1.2 as this version will be mandated in the future.
- You don't have to configure client authentication on an SSL profile to associate it to a listener. You can have only client authentication or listener-specific SSL policy configured, or both configured in your SSL profile.
- Using a new Predefined or Customv2 policy enhances SSL security and performance for the entire gateway (SSL Policy and SSL Profile). Therefore, you cannot have different listeners on both old as well as new SSL (predefined or custom) policies.

  Consider this example, you are currently using SSL Policy and SSL Profile with "older" policies/ciphers. To use a "new" Predefined or Customv2 policy for any one of them will also require you to upgrade the other configuration. You may use the new predefined policies, or customv2 policy, or combination of these across the gateway.

To set up a listener-specific SSL policy, you'll need to first go to the **SSL settings** tab in the Portal and create a new SSL profile. When you create an SSL profile, you'll see two tabs: **Client Authentication** and **SSL Policy**. The **SSL Policy** tab is to configure a listener-specific SSL policy. The **Client Authentication** tab is where to upload a client certificate(s) for mutual authentication - for more information, check out Configuring a mutual authentication.

1. Search for **Application Gateway** in portal, select **Application gateways**, and click on your existing Application Gateway.
2. Select **SSL settings** from the left-side menu.

3. Click on the plus sign next to **SSL Profiles** at the top to create a new SSL profile.
4. Enter a name under **SSL Profile Name**. In this example, we call our SSL profile *applicationGatewaySSLProfile*.
5. Go to the **SSL Policy** tab and check the **Enable listener-specific SSL Policy** box.
6. Set up your listener-specific SSL policy given your requirements. You can choose between predefined SSL policies and customizing your own SSL policy. For more information on SSL policies, visit SSL policy overview. We recommend using TLS 1.2
7. Select **Add** to save.

Associate the SSL profile with a listener

Now that we've created an SSL profile with a listener-specific SSL policy, we need to associate the SSL profile to the listener to put the listener-specific policy in action.

1. Navigate to your existing Application Gateway. If you just completed the steps above, you don't need to do anything here.

2. Select **Listeners** from the left-side menu.
3. Click on **Add listener** if you don't already have an HTTPS listener set up. If you already have an HTTPS listener, click on it from the list.
4. Fill out the **Listener name**, **Frontend IP**, **Port**, **Protocol**, and other **HTTPS Settings** to fit your requirements.
5. Check the **Enable SSL Profile** checkbox so that you can select which SSL Profile to associate with the listener.
6. Select the SSL profile you created from the dropdown list. In this example, we choose the SSL profile we created from the earlier steps: *applicationGatewaySSLProfile*.
7. Continue configuring the remainder of the listener to fit your requirements.
8. Click **Add** to save your new listener with the SSL profile associated to it.

Limitations

There is a limitation right now on Application Gateway where different listeners using the same port cannot have SSL policies (predefined or custom) with different TLS protocol versions. Choosing the same TLS version for different listeners will work for configuring cipher suite preference for each listener. However, to use different TLS protocol versions for separate listeners, you will need to use distinct ports for each.

Next steps

Manage web traffic with an application gateway using the Azure CLI

Install an Application Gateway Ingress Controller (AGIC) using an existing Application Gateway

8/22/2022 • 9 minutes to read • Edit Online

The Application Gateway Ingress Controller (AGIC) is a pod within your Kubernetes cluster. AGIC monitors the Kubernetes Ingress resources, and creates and applies Application Gateway config based on the status of the Kubernetes cluster.

Outline:

- Prerequisites
- Azure Resource Manager Authentication (ARM)
  - Option 1: Set up aad-pod-identity and create Azure Identity on ARMs
  - Option 2: Using a Service Principal
- Install Ingress Controller using Helm
- Multi-cluster / Shared Application Gateway: Install AGIC in an environment, where Application Gateway is shared between one or more AKS clusters and/or other Azure components.

Prerequisites

This document assumes you already have the following tools and infrastructure installed:

- AKS with Azure Container Networking Interface (CNI)
- Application Gateway v2 in the same virtual network as AKS
- AAD Pod Identity installed on your AKS cluster
- Cloud Shell is the Azure shell environment, which has `az` CLI, `kubectl`, and `helm` installed. These tools are required for the commands below.

Please **backup your Application Gateway's configuration** before installing AGIC:

1. using Azure portal navigate to your `Application Gateway` instance
2. from `Export template` click `Download`

The zip file you downloaded will have JSON templates, bash, and PowerShell scripts you could use to restore App Gateway should that become necessary

Install Helm

Helm is a package manager for Kubernetes. We will leverage it to install the `application-gateway-kubernetes-ingress` package. Use Cloud Shell to install Helm:

1. Install Helm and run the following to add `application-gateway-kubernetes-ingress` helm package:

   - *Kubernetes RBAC enabled* AKS cluster

```
kubectl create serviceaccount --namespace kube-system tiller-sa
kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
serviceaccount=kube-system:tiller-sa
helm init --tiller-namespace kube-system --service-account tiller-sa
```

- *Kubernetes RBAC disabled* AKS cluster

```
helm init
```

2. Add the AGIC Helm repository:

```
helm repo add application-gateway-kubernetes-ingress
https://appgwingress.blob.core.windows.net/ingress-azure-helm-package/
helm repo update
```

Azure Resource Manager Authentication

AGIC communicates with the Kubernetes API server and the Azure Resource Manager. It requires an identity to access these APIs.

Set up AAD Pod Identity

AAD Pod Identity is a controller, similar to AGIC, which also runs on your AKS. It binds Azure Active Directory identities to your Kubernetes pods. Identity is required for an application in a Kubernetes pod to be able to communicate with other Azure components. In the particular case here, we need authorization for the AGIC pod to make HTTP requests to ARM.

Follow the AAD Pod Identity installation instructions to add this component to your AKS.

Next we need to create an Azure identity and give it permissions ARM. Use Cloud Shell to run all of the following commands and create an identity:

1. Create an Azure identity **in the same resource group as the AKS nodes**. Picking the correct resource group is important. The resource group required in the command below is *not* the one referenced on the AKS portal pane. This is the resource group of the `aks-agentpool` virtual machines. Typically that resource group starts with `MC_` and contains the name of your AKS. For instance: `MC_resourceGroup_aksABCD_westus`

```
az identity create -g <agent-pool-resource-group> -n <identity-name>
```

2. For the role assignment commands below we need to obtain `principalId` for the newly created identity:

```
az identity show -g <resourcegroup> -n <identity-name>
```

3. Give the identity `Contributor` access to your Application Gateway. For this you need the ID of the Application Gateway, which will look something like this: `/subscriptions/A/resourceGroups/B/providers/Microsoft.Network/applicationGateways/C`

   Get the list of Application Gateway IDs in your subscription with: `az network application-gateway list --query '[].id'`

```
az role assignment create \
    --role Contributor \
    --assignee <principalId> \
    --scope <App-Gateway-ID>
```

4. Give the identity `Reader` access to the Application Gateway resource group. The resource group ID would

look like: `/subscriptions/A/resourceGroups/B`. You can get all resource groups with: `az group list --query '[].id'`

```
az role assignment create \
    --role Reader \
    --assignee <principalId> \
    --scope <App-Gateway-Resource-Group-ID>
```

Using a Service Principal

It is also possible to provide AGIC access to ARM via a Kubernetes secret.

1. Create an Active Directory Service Principal and encode with base64. The base64 encoding is required for the JSON blob to be saved to Kubernetes.

```
az ad sp create-for-rbac --role Contributor --sdk-auth | base64 -w0
```

2. Add the base64 encoded JSON blob to the `helm-config.yaml` file. More information on `helm-config.yaml` is in the next section.

```
armAuth:
    type: servicePrincipal
    secretJSON: <Base64-Encoded-Credentials>
```

Install Ingress Controller as a Helm Chart

In the first few steps, we install Helm's Tiller on your Kubernetes cluster. Use Cloud Shell to install the AGIC Helm package:

1. Add the `application-gateway-kubernetes-ingress` helm repo and perform a helm update

```
helm repo add application-gateway-kubernetes-ingress
https://appgwingress.blob.core.windows.net/ingress-azure-helm-package/
helm repo update
```

2. Download helm-config.yaml, which will configure AGIC:

```
wget https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-
ingress/master/docs/examples/sample-helm-config.yaml -O helm-config.yaml
```

Or copy the YAML file below:

```
# This file contains the essential configs for the ingress controller helm chart

# Verbosity level of the App Gateway Ingress Controller
verbosityLevel: 3

################################################################################
# Specify which application gateway the ingress controller will manage
#
appgw:
    subscriptionId: <subscriptionId>
    resourceGroup: <resourceGroupName>
    name: <applicationGatewayName>

    # Setting appgw.shared to "true" will create an AzureIngressProhibitedTarget CRD.
    # This prohibits AGIC from applying config for any host/path.
    # Use "kubectl get AzureIngressProhibitedTargets" to view and change this.
    shared: false

################################################################################
# Specify which kubernetes namespace the ingress controller will watch
# Default value is "default"
# Leaving this variable out or setting it to blank or empty string would
# result in Ingress Controller observing all acessible namespaces.
#
# kubernetes:
#   watchNamespace: <namespace>

################################################################################
# Specify the authentication with Azure Resource Manager
#
# Two authentication methods are available:
# - Option 1: AAD-Pod-Identity (https://github.com/Azure/aad-pod-identity)
armAuth:
    type: aadPodIdentity
    identityResourceID: <identityResourceId>
    identityClientID:  <identityClientId>

## Alternatively you can use Service Principal credentials
# armAuth:
#    type: servicePrincipal
#    secretJSON: <<Generate this value with: "az ad sp create-for-rbac --role Contributor --sdk-auth
| base64 -w0" >>

################################################################################
# Specify if the cluster is Kubernetes RBAC enabled or not
rbac:
    enabled: false # true/false

# Specify aks cluster related information. THIS IS BEING DEPRECATED.
aksClusterConfiguration:
    apiServerAddress: <aks-api-server-address>
```

3. Edit helm-config.yaml and fill in the values for `appgw` and `armAuth`.

```
nano helm-config.yaml
```

> **NOTE**
>
> The `<identity-resource-id>` and `<identity-client-id>` are the properties of the Azure AD Identity you setup in the previous section. You can retrieve this information by running the following command: `az identity show -g <resourcegroup> -n <identity-name>`, where `<resourcegroup>` is the resource group in which the top level AKS cluster object, Application Gateway and Managed Identify are deployed.

4. Install Helm chart `application-gateway-kubernetes-ingress` with the `helm-config.yaml` configuration from the previous step

```
helm install -f <helm-config.yaml> application-gateway-kubernetes-ingress/ingress-azure
```

Alternatively you can combine the `helm-config.yaml` and the Helm command in one step:

```
helm install ./helm/ingress-azure \
     --name ingress-azure \
     --namespace default \
     --debug \
     --set appgw.name=applicationgatewayABCD \
     --set appgw.resourceGroup=your-resource-group \
     --set appgw.subscriptionId=subscription-uuid \
     --set appgw.shared=false \
     --set armAuth.type=servicePrincipal \
     --set armAuth.secretJSON=$(az ad sp create-for-rbac --role Contributor --sdk-auth | base64 -w0)
\
     --set rbac.enabled=true \
     --set verbosityLevel=3 \
     --set kubernetes.watchNamespace=default \
     --set aksClusterConfiguration.apiServerAddress=aks-abcdefg.hcp.westus2.azmk8s.io
```

5. Check the log of the newly created pod to verify if it started properly

Refer to this how-to guide to understand how you can expose an AKS service over HTTP or HTTPS, to the internet, using an Azure Application Gateway.

Multi-cluster / Shared Application Gateway

By default AGIC assumes full ownership of the Application Gateway it is linked to. AGIC version 0.8.0 and later can share a single Application Gateway with other Azure components. For instance, we could use the same Application Gateway for an app hosted on Virtual Machine Scale Set as well as an AKS cluster.

Please **backup your Application Gateway's configuration** before enabling this setting:

1. using Azure portal navigate to your `Application Gateway` instance
2. from `Export template` click `Download`

The zip file you downloaded will have JSON templates, bash, and PowerShell scripts you could use to restore Application Gateway

Example Scenario

Let's look at an imaginary Application Gateway, which manages traffic for two web sites:

- `dev.contoso.com` - hosted on a new AKS, using Application Gateway and AGIC
- `prod.contoso.com` - hosted on an Azure Virtual Machine Scale Set

With default settings, AGIC assumes 100% ownership of the Application Gateway it is pointed to. AGIC overwrites all of App Gateway's configuration. If we were to manually create a listener for `prod.contoso.com` (on Application Gateway), without defining it in the Kubernetes Ingress, AGIC will delete the `prod.contoso.com` config within seconds.

To install AGIC and also serve `prod.contoso.com` from our Virtual Machine Scale Set machines, we must constrain AGIC to configuring `dev.contoso.com` only. This is facilitated by instantiating the following CRD:

```
cat <<EOF | kubectl apply -f -
apiVersion: "appgw.ingress.k8s.io/v1"
kind: AzureIngressProhibitedTarget
metadata:
  name: prod-contoso-com
spec:
  hostname: prod.contoso.com
EOF
```

The command above creates an `AzureIngressProhibitedTarget` object. This makes AGIC (version 0.8.0 and later) aware of the existence of Application Gateway config for `prod.contoso.com` and explicitly instructs it to avoid changing any configuration related to that hostname.

Enable with new AGIC installation

To limit AGIC (version 0.8.0 and later) to a subset of the Application Gateway configuration modify the `helm-config.yaml` template. Under the `appgw:` section, add `shared` key and set it to to `true`.

```
appgw:
    subscriptionId: <subscriptionId>    # existing field
    resourceGroup: <resourceGroupName>  # existing field
    name: <applicationGatewayName>      # existing field
    shared: true                        # <<<<< Add this field to enable shared Application Gateway >>>>>
```

Apply the Helm changes:

1. Ensure the `AzureIngressProhibitedTarget` CRD is installed with:

   ```
   kubectl apply -f https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-
   ingress/ae695ef9bd05c8b708cedf6ff545595d0b7022dc/crds/AzureIngressProhibitedTarget.yaml
   ```

2. Update Helm:

   ```
   helm upgrade \
       --recreate-pods \
       -f helm-config.yaml \
       ingress-azure application-gateway-kubernetes-ingress/ingress-azure
   ```

As a result your AKS will have a new instance of `AzureIngressProhibitedTarget` called `prohibit-all-targets`:

```
kubectl get AzureIngressProhibitedTargets prohibit-all-targets -o yaml
```

The object `prohibit-all-targets`, as the name implies, prohibits AGIC from changing config for *any* host and path. Helm install with `appgw.shared=true` will deploy AGIC, but will not make any changes to Application Gateway.

Broaden permissions

Since Helm with `appgw.shared=true` and the default `prohibit-all-targets` blocks AGIC from applying any config.

Broaden AGIC permissions with:

1. Create a new `AzureIngressProhibitedTarget` with your specific setup:

```
cat <<EOF | kubectl apply -f -
apiVersion: "appgw.ingress.k8s.io/v1"
kind: AzureIngressProhibitedTarget
metadata:
  name: your-custom-prohibitions
spec:
  hostname: your.own-hostname.com
EOF
```

2. Only after you have created your own custom prohibition, you can delete the default one, which is too broad:

```
kubectl delete AzureIngressProhibitedTarget prohibit-all-targets
```

Enable for an existing AGIC installation

Let's assume that we already have a working AKS, Application Gateway, and configured AGIC in our cluster. We have an Ingress for `prod.contoso.com` and are successfully serving traffic for it from AKS. We want to add `staging.contoso.com` to our existing Application Gateway, but need to host it on a VM. We are going to reuse the existing Application Gateway and manually configure a listener and backend pools for `staging.contoso.com`. But manually tweaking Application Gateway config (via portal, ARM APIs or Terraform) would conflict with AGIC's assumptions of full ownership. Shortly after we apply changes, AGIC will overwrite or delete them.

We can prohibit AGIC from making changes to a subset of configuration.

1. Create an `AzureIngressProhibitedTarget` object:

```
cat <<EOF | kubectl apply -f -
apiVersion: "appgw.ingress.k8s.io/v1"
kind: AzureIngressProhibitedTarget
metadata:
  name: manually-configured-staging-environment
spec:
  hostname: staging.contoso.com
EOF
```

2. View the newly created object:

```
kubectl get AzureIngressProhibitedTargets
```

3. Modify Application Gateway config via portal - add listeners, routing rules, backends etc. The new object we created (`manually-configured-staging-environment`) will prohibit AGIC from overwriting Application Gateway configuration related to `staging.contoso.com`.

How to Install an Application Gateway Ingress Controller (AGIC) Using a New Application Gateway

8/22/2022 • 6 minutes to read • Edit Online

The instructions below assume Application Gateway Ingress Controller (AGIC) will be installed in an environment with no pre-existing components.

Required Command Line Tools

We recommend the use of Azure Cloud Shell for all command-line operations below. Launch your shell from shell.azure.com or by clicking the link:

Alternatively, launch Cloud Shell from Azure portal using the following icon:

Your Azure Cloud Shell already has all necessary tools. If you choose to use another environment, ensure the following command-line tools are installed:

- `az` - Azure CLI: installation instructions
- `kubectl` - Kubernetes command-line tool: installation instructions
- `helm` - Kubernetes package manager: installation instructions
- `jq` - command-line JSON processor: installation instructions

Create an Identity

Follow the steps below to create an Azure Active Directory (Azure AD) service principal object. Record the `appId`, `password`, and `objectId` values - these values will be used in the following steps.

1. Create AD service principal (Read more about Azure RBAC):

   ```
   az ad sp create-for-rbac --role Contributor --scopes /subscriptions/mySubscriptionID -o json >
   auth.json
   appId=$(jq -r ".appId" auth.json)
   password=$(jq -r ".password" auth.json)
   ```

   The `appId` and `password` values from the JSON output will be used in the following steps

2. Use the `appId` from the previous command's output to get the `objectId` of the new service principal:

   ```
   objectId=$(az ad sp show --id $appId --query "objectId" -o tsv)
   ```

   The output of this command is `objectId`, which will be used in the Azure Resource Manager template below

3. Create the parameter file that will be used in the Azure Resource Manager template deployment later.

```
cat <<EOF > parameters.json
{
  "aksServicePrincipalAppId": { "value": "$appId" },
  "aksServicePrincipalClientSecret": { "value": "$password" },
  "aksServicePrincipalObjectId": { "value": "$objectId" },
  "aksEnableRBAC": { "value": false }
}
EOF
```

To deploy an **Kubernetes RBAC** enabled cluster, set the `aksEnableRBAC` field to `true`

Deploy Components

This step will add the following components to your subscription:

- Azure Kubernetes Service
- Application Gateway v2
- Virtual Network with two subnets
- Public IP Address
- Managed Identity, which will be used by Azure AD Pod Identity

1. Download the Azure Resource Manager template and modify the template as needed.

```
wget https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-
ingress/master/deploy/azuredeploy.json -O template.json
```

2. Deploy the Azure Resource Manager template using `az cli`. The deployment might take up to 5 minutes.

```
resourceGroupName="MyResourceGroup"
location="westus2"
deploymentName="ingress-appgw"

# create a resource group
az group create -n $resourceGroupName -l $location

# modify the template as needed
az deployment group create \
        -g $resourceGroupName \
        -n $deploymentName \
        --template-file template.json \
        --parameters parameters.json
```

3. Once the deployment finished, download the deployment output into a file named `deployment-outputs.json`.

```
az deployment group show -g $resourceGroupName -n $deploymentName --query "properties.outputs" -o
json > deployment-outputs.json
```

Set up Application Gateway Ingress Controller

With the instructions in the previous section, we created and configured a new AKS cluster and an Application Gateway. We're now ready to deploy a sample app and an ingress controller to our new Kubernetes infrastructure.

Setup Kubernetes Credentials

For the following steps, we need setup kubectl command, which we'll use to connect to our new Kubernetes cluster. Cloud Shell has `kubectl` already installed. We'll use `az` CLI to obtain credentials for Kubernetes.

Get credentials for your newly deployed AKS (read more):

```
# use the deployment-outputs.json created after deployment to get the cluster name and resource group name
aksClusterName=$(jq -r ".aksClusterName.value" deployment-outputs.json)
resourceGroupName=$(jq -r ".resourceGroupName.value" deployment-outputs.json)

az aks get-credentials --resource-group $resourceGroupName --name $aksClusterName
```

Install Azure AD Pod Identity

Azure Active Directory Pod Identity provides token-based access to Azure Resource Manager (ARM).

Azure AD Pod Identity will add the following components to your Kubernetes cluster:

- Kubernetes CRDs: `AzureIdentity`, `AzureAssignedIdentity`, `AzureIdentityBinding`
- Managed Identity Controller (MIC) component
- Node Managed Identity (NMI) component

To install Azure AD Pod Identity to your cluster:

- *Kubernetes RBAC enabled* AKS cluster

  ```
  kubectl create -f https://raw.githubusercontent.com/Azure/aad-pod-identity/master/deploy/infra/deployment-rbac.yaml
  ```

- *Kubernetes RBAC disabled* AKS cluster

  ```
  kubectl create -f https://raw.githubusercontent.com/Azure/aad-pod-identity/master/deploy/infra/deployment.yaml
  ```

Install Helm

Helm is a package manager for Kubernetes. We'll use it to install the `application-gateway-kubernetes-ingress` package:

1. Install Helm and run the following to add `application-gateway-kubernetes-ingress` helm package:

   - *Kubernetes RBAC enabled* AKS cluster

     ```
     kubectl create serviceaccount --namespace kube-system tiller-sa
     kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --serviceaccount=kube-system:tiller-sa
     helm init --tiller-namespace kube-system --service-account tiller-sa
     ```

   - *Kubernetes RBAC disabled* AKS cluster

     ```
     helm init
     ```

2. Add the AGIC Helm repository:

```
helm repo add application-gateway-kubernetes-ingress
https://appgwingress.blob.core.windows.net/ingress-azure-helm-package/
helm repo update
```

Install Ingress Controller Helm Chart

1. Use the `deployment-outputs.json` file created above and create the following variables.

```
applicationGatewayName=$(jq -r ".applicationGatewayName.value" deployment-outputs.json)
resourceGroupName=$(jq -r ".resourceGroupName.value" deployment-outputs.json)
subscriptionId=$(jq -r ".subscriptionId.value" deployment-outputs.json)
identityClientId=$(jq -r ".identityClientId.value" deployment-outputs.json)
identityResourceId=$(jq -r ".identityResourceId.value" deployment-outputs.json)
```

2. Download helm-config.yaml, which will configure AGIC:

```
wget https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-
ingress/master/docs/examples/sample-helm-config.yaml -O helm-config.yaml
```

   Or copy the YAML file below:

```
# This file contains the essential configs for the ingress controller helm chart

# Verbosity level of the App Gateway Ingress Controller
verbosityLevel: 3

################################################################################
# Specify which application gateway the ingress controller will manage
#
appgw:
    subscriptionId: <subscriptionId>
    resourceGroup: <resourceGroupName>
    name: <applicationGatewayName>

    # Setting appgw.shared to "true" will create an AzureIngressProhibitedTarget CRD.
    # This prohibits AGIC from applying config for any host/path.
    # Use "kubectl get AzureIngressProhibitedTargets" to view and change this.
    shared: false

################################################################################
# Specify which kubernetes namespace the ingress controller will watch
# Default value is "default"
# Leaving this variable out or setting it to blank or empty string would
# result in Ingress Controller observing all acessible namespaces.
#
# kubernetes:
#   watchNamespace: <namespace>

################################################################################
# Specify the authentication with Azure Resource Manager
#
# Two authentication methods are available:
# - Option 1: AAD-Pod-Identity (https://github.com/Azure/aad-pod-identity)
armAuth:
    type: aadPodIdentity
    identityResourceID: <identityResourceId>
    identityClientID:  <identityClientId>

## Alternatively you can use Service Principal credentials
# armAuth:
#    type: servicePrincipal
#    secretJSON: <<Generate this value with: "az ad sp create-for-rbac --subscription <subscription-
uuid> --role Contributor --sdk-auth | base64 -w0" >>

################################################################################
# Specify if the cluster is Kubernetes RBAC enabled or not
rbac:
    enabled: false # true/false

# Specify aks cluster related information. THIS IS BEING DEPRECATED.
aksClusterConfiguration:
    apiServerAddress: <aks-api-server-address>
```

3. Edit the newly downloaded helm-config.yaml and fill out the sections `appgw` and `armAuth`.

```
sed -i "s|<subscriptionId>|${subscriptionId}|g" helm-config.yaml
sed -i "s|<resourceGroupName>|${resourceGroupName}|g" helm-config.yaml
sed -i "s|<applicationGatewayName>|${applicationGatewayName}|g" helm-config.yaml
sed -i "s|<identityResourceId>|${identityResourceId}|g" helm-config.yaml
sed -i "s|<identityClientId>|${identityClientId}|g" helm-config.yaml

# You can further modify the helm config to enable/disable features
nano helm-config.yaml
```

Values:

- `verbosityLevel`: Sets the verbosity level of the AGIC logging infrastructure. See Logging Levels for possible values.
- `appgw.subscriptionId`: The Azure Subscription ID in which Application Gateway resides. Example: `a123b234-a3b4-557d-b2df-a0bc12de1234`
- `appgw.resourceGroup`: Name of the Azure Resource Group in which Application Gateway was created. Example: `app-gw-resource-group`
- `appgw.name`: Name of the Application Gateway. Example: `applicationgatewayd0f0`
- `appgw.shared`: This boolean flag should be defaulted to `false`. Set to `true` should you need a Shared Application Gateway.
- `kubernetes.watchNamespace`: Specify the namespace that AGIC should watch. The namespace value can be a single string value, or a comma-separated list of namespaces.
- `armAuth.type`: could be `aadPodIdentity` or `servicePrincipal`
- `armAuth.identityResourceID`: Resource ID of the Azure Managed Identity
- `armAuth.identityClientID`: The Client ID of the Identity. More information about **identityClientID** is provided below.
- `armAuth.secretJSON`: Only needed when Service Principal Secret type is chosen (when `armAuth.type` has been set to `servicePrincipal`)

> **NOTE**
>
> The `identityResourceID` and `identityClientID` are values that were created during the Deploy Components steps, and could be obtained again using the following command:
>
> ```
> az identity show -g <resource-group> -n <identity-name>
> ```
>
> `<resource-group>` in the command above is the resource group of your Application Gateway. `<identity-name>` is the name of the created identity. All identities for a given subscription can be listed using: `az identity list`

4. Install the Application Gateway ingress controller package:

```
helm install -f helm-config.yaml application-gateway-kubernetes-ingress/ingress-azure
```

Install a Sample App

Now that we have Application Gateway, AKS, and AGIC installed we can install a sample app via Azure Cloud Shell:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: aspnetapp
  labels:
    app: aspnetapp
spec:
  containers:
  - image: "mcr.microsoft.com/dotnet/core/samples:aspnetapp"
    name: aspnetapp-image
    ports:
    - containerPort: 80
      protocol: TCP

---

apiVersion: v1
kind: Service
metadata:
  name: aspnetapp
spec:
  selector:
    app: aspnetapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80

---

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: aspnetapp
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: aspnetapp
          servicePort: 80
EOF
```

Alternatively you can:

- Download the YAML file above:

```
curl https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-
ingress/master/docs/examples/aspnetapp.yaml -o aspnetapp.yaml
```

- Apply the YAML file:

```
kubectl apply -f aspnetapp.yaml
```

Other Examples

This how-to guide contains more examples on how to expose an AKS service via HTTP or HTTPS, to the Internet

with Application Gateway.

Migrate from AGIC Helm to AGIC add-on

8/22/2022 • 2 minutes to read • Edit Online

If you already have AGIC deployed through Helm but want to migrate to AGIC deployed as an AKS add-on, the following steps will help guide you through the migration process.

Prerequisites

Before you start the migration process, there are a few things to check.

- Are you using any features with AGIC Helm that are currently not supported with AGIC add-on?
- Are you using more than one AGIC Helm deployment per AKS cluster?
- Are you using multiple AGIC Helm deployments to target one Application Gateway?

If you answered yes to any of the questions above, AGIC add-on won't support your use case yet so it will be best to continue using AGIC Helm in the meantime. Otherwise, continue with the migration process below during off-business hours.

Find the Application Gateway resource ID that AGIC Helm is currently targeting

Navigate to the Application Gateway that your AGIC Helm deployment is targeting. Copy and save the resource ID of that Application Gateway. You will need the resource ID in a later step. The resource ID can be found in Portal, under the Properties tab of your Application Gateway or through Azure CLI. The following example saves the Application Gateway resource ID to *appgwId* for a gateway named *myApplicationGateway* in the resource group *myResourceGroup*.

```
appgwId=$(az network application-gateway show -n myApplicationGateway -g myResourceGroup -o tsv --query
"id")
```

Delete AGIC Helm from your AKS cluster

Through Azure CLI, delete your AGIC Helm deployment from your cluster. You'll need to delete the AGIC Helm deployment first before you can enable the AGIC AKS add-on. Please note that any changes that occur within your AKS cluster between the time of deleting your AGIC Helm deployment and the time you enable the AGIC add-on won't be reflected on your Application Gateway, and therefore this migration process should be done outside of business hours to minimize impact. Application Gateway will continue to have the last configuration applied by AGIC so existing routing rules will not be affected.

Enable AGIC add-on using your existing Application Gateway

You can now enable the AGIC add-on in your AKS cluster to target your existing Application Gateway through Azure CLI or Portal. Run the following Azure CLI command to enable the AGIC add-on in your AKS cluster. The example enables the add-on in a cluster called *myCluster*, in a resource group called *myResourceGroup*, using the Application Gateway resource ID *appgwId* we saved above in the earlier step.

```
az aks enable-addons -n myCluster -g myResourceGroup -a ingress-appgw --appgw-id $appgwId
```

Alternatively, you can navigate to your AKS cluster in Portal using this link and enable the AGIC add-on in the Networking tab of your cluster. Select your existing Application Gateway from the dropdown menu when you choose which Application Gateway the add-on should target.

Next Steps

- **Application Gateway Ingress Controller Troubleshooting**: Troubleshooting guide for AGIC
- **Application Gateway Ingress Controller Annotations**: List of annotations on AGIC

Disable and re-enable AGIC add-on for your AKS cluster

8/22/2022 • 2 minutes to read • Edit Online

Application Gateway Ingress Controller (AGIC) deployed as an AKS add-on allows you to enable and disable the add-on with one line in Azure CLI. The life cycle of the Application Gateway will differ when you disable the AGIC add-on, depending on if the Application Gateway was created by the AGIC add-on, or if it was deployed separately from the AGIC add-on. You can run the same command to re-enable the AGIC add-on if you ever disable it, or to enable the AGIC add-on using an existing AKS cluster and Application Gateway.

Disabling AGIC add-on with associated Application Gateway

If the AGIC add-on automatically deployed the Application Gateway for you when you first set everything up, then disabling the AGIC add-on will by default delete the Application Gateway based on a couple criteria. There are two criteria that the AGIC add-on looks for to determine if it should delete the associated Application Gateway when you disable it:

- Is the Application Gateway that the AGIC add-on is associated with deployed in the MC_* node resource group?
- Does the Application Gateway that the AGIC add-on is associated with have the tag "created-by: ingress-appgw"? The tag is used by AGIC to determine if the Application Gateway was deployed by the add-on or not.

If both criteria are met, then the AGIC add-on will delete the Application Gateway it created when the add-on is disabled; however, it won't delete the public IP or the subnet in which the Application Gateway was deployed with/in. If the first criteria is not met, then it won't matter if the Application Gateway has the "created-by: ingress-appgw" tag - disabling the add-on won't delete the Application Gateway. Likewise, if the second criteria is not met, i.e. the Application Gateway lacks that tag, then disabling the add-on won't delete the Application Gateway in the MC_* node resource group.

> **TIP**
> If you don't want the Application Gateway to be deleted when disabling the add-on, but it meets both criteria then remove the "created-by: ingress-appgw" tag to prevent the add-on from deleting your Application Gateway.

To disable the AGIC add-on, run the following command:

```
az aks disable-addons -n <AKS-cluster-name> -g <AKS-resource-group-name> -a ingress-appgw
```

Enable AGIC add-on on existing Application Gateway and AKS Cluster

If you ever disable the AGIC add-on and need to re-enable the add-on, or want to enable the add-on using an existing Application Gateway and AKS cluster, then run the following command:

```
appgwId=$(az network application-gateway show -n <application-gateway-name> -g <resource-group-name> -o tsv
--query "id")
az aks enable-addons -n <AKS-cluster-name> -g <AKS-cluster-resource-group> -a ingress-appgw --appgw-id
$appgwId
```

Next steps

For more details on how to enable the AGIC add-on using an existing Application Gateway and AKS cluster, see AGIC add-on brownfield deployment.

Enable Cookie based affinity with an Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

As outlined in the Azure Application Gateway Documentation, Application Gateway supports cookie based affinity, which means it can direct subsequent traffic from a user session to the same server for processing.

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/cookie-based-affinity: "true"
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: frontend
          servicePort: 80
```

Enable multiple Namespace support in an AKS cluster with Application Gateway Ingress Controller

8/22/2022 • 4 minutes to read • Edit Online

Motivation

Kubernetes Namespaces make it possible for a Kubernetes cluster to be partitioned and allocated to subgroups of a larger team. These subteams can then deploy and manage infrastructure with finer controls of resources, security, configuration etc. Kubernetes allows for one or more ingress resources to be defined independently within each namespace.

As of version 0.7 Azure Application Gateway Kubernetes IngressController (AGIC) can ingest events from and observe multiple namespaces. Should the AKS administrator decide to use App Gateway as an ingress, all namespaces will use the same instance of Application Gateway. A single installation of Ingress Controller will monitor accessible namespaces and will configure the Application Gateway it is associated with.

Version 0.7 of AGIC will continue to exclusively observe the `default` namespace, unless this is explicitly changed to one or more different namespaces in the Helm configuration (see section below).

Enable multiple namespace support

To enable multiple namespace support:

1. modify the helm-config.yaml file in one of the following ways:
   - delete the `watchNamespace` key entirely from helm-config.yaml - AGIC will observe all namespaces
   - set `watchNamespace` to an empty string - AGIC will observe all namespaces
   - add multiple namespaces separated by a comma (`watchNamespace: default,secondNamespace`) - AGIC will observe these namespaces exclusively
2. apply Helm template changes with:
   `helm install -f helm-config.yaml application-gateway-kubernetes-ingress/ingress-azure`

Once deployed with the ability to observe multiple namespaces, AGIC will:

- list ingress resources from all accessible namespaces
- filter to ingress resources annotated with `kubernetes.io/ingress.class: azure/application-gateway`
- compose combined Application Gateway config
- apply the config to the associated Application Gateway via ARM

Conflicting Configurations

Multiple namespaced ingress resources could instruct AGIC to create conflicting configurations for a single Application Gateway. (Two ingresses claiming the same domain for instance.)

At the top of the hierarchy - **listeners** (IP address, port, and host) and **routing rules** (binding listener, backend pool, and HTTP settings) could be created and shared by multiple namespaces/ingresses.

On the other hand - paths, backend pools, HTTP settings, and TLS certificates could be created by one namespace only and duplicates will be removed.

For example, consider the following duplicate ingress resources defined namespaces `staging` and `production`

for `www.contoso.com`:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: websocket-ingress
  namespace: staging
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
    - host: www.contoso.com
      http:
        paths:
          - backend:
              serviceName: web-service
              servicePort: 80
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: websocket-ingress
  namespace: production
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
    - host: www.contoso.com
      http:
        paths:
          - backend:
              serviceName: web-service
              servicePort: 80
```

Despite the two ingress resources demanding traffic for `www.contoso.com` to be routed to the respective Kubernetes namespaces, only one backend can service the traffic. AGIC would create a configuration on "first come, first served" basis for one of the resources. If two ingresses resources are created at the same time, the one earlier in the alphabet will take precedence. From the example above we will only be able to create settings for the `production` ingress. Application Gateway will be configured with the following resources:

- Listener: `fl-www.contoso.com-80`
- Routing Rule: `rr-www.contoso.com-80`
- Backend Pool: `pool-production-contoso-web-service-80-bp-80`
- HTTP Settings: `bp-production-contoso-web-service-80-80-websocket-ingress`
- Health Probe: `pb-production-contoso-web-service-80-websocket-ingress`

Note that except for *listener* and *routing rule*, the Application Gateway resources created include the name of the namespace (`production`) for which they were created.

If the two ingress resources are introduced into the AKS cluster at different points in time, it is likely for AGIC to end up in a scenario where it reconfigures Application Gateway and re-routes traffic from `namespace-B` to `namespace-A`.

For example if you added `staging` first, AGIC will configure Application Gateway to route traffic to the staging backend pool. At a later stage, introducing `production` ingress, will cause AGIC to reprogram Application Gateway, which will start routing traffic to the `production` backend pool.

Restrict Access to Namespaces

By default AGIC will configure Application Gateway based on annotated Ingress within any namespace. Should you want to limit this behavior you have the following options:

- limit the namespaces, by explicitly defining namespaces AGIC should observe via the `watchNamespace` YAML key in helm-config.yaml
- use Role/RoleBinding to limit AGIC to specific namespaces

Sample Helm config file

```
    # This file contains the essential configs for the ingress controller helm chart

    # Verbosity level of the App Gateway Ingress Controller
    verbosityLevel: 3

    ################################################################################
    # Specify which application gateway the ingress controller will manage
    #
    appgw:
        subscriptionId: <subscriptionId>
        resourceGroup: <resourceGroupName>
        name: <applicationGatewayName>

        # Setting appgw.shared to "true" will create an AzureIngressProhibitedTarget CRD.
        # This prohibits AGIC from applying config for any host/path.
        # Use "kubectl get AzureIngressProhibitedTargets" to view and change this.
        shared: false

    ################################################################################
    # Specify which kubernetes namespace the ingress controller will watch
    # Default value is "default"
    # Leaving this variable out or setting it to blank or empty string would
    # result in Ingress Controller observing all acessible namespaces.
    #
    # kubernetes:
    #   watchNamespace: <namespace>

    ################################################################################
    # Specify the authentication with Azure Resource Manager
    #
    # Two authentication methods are available:
    # - Option 1: AAD-Pod-Identity (https://github.com/Azure/aad-pod-identity)
    armAuth:
        type: aadPodIdentity
        identityResourceID: <identityResourceId>
        identityClientID:  <identityClientId>

    ## Alternatively you can use Service Principal credentials
    # armAuth:
    #    type: servicePrincipal
    #    secretJSON: <<Generate this value with: "az ad sp create-for-rbac --subscription <subscription-
uuid> --role Contributor --sdk-auth | base64 -w0" >>

    ################################################################################
    # Specify if the cluster is Kubernetes RBAC enabled or not
    rbac:
        enabled: false # true/false

    # Specify aks cluster related information. THIS IS BEING DEPRECATED.
    aksClusterConfiguration:
        apiServerAddress: <aks-api-server-address>
```

Use private IP for internal routing for an Ingress endpoint

8/22/2022 • 2 minutes to read • Edit Online

This feature allows to expose the ingress endpoint within the `Virtual Network` using a private IP.

Pre-requisites

Application Gateway with a Private IP configuration

There are two ways to configure the controller to use Private IP for ingress,

Assign to a particular ingress

To expose a particular ingress over Private IP, use annotation `appgw.ingress.kubernetes.io/use-private-ip` in Ingress.

Usage

```
appgw.ingress.kubernetes.io/use-private-ip: "true"
```

For Application Gateways without a Private IP, Ingresses annotated with `appgw.ingress.kubernetes.io/use-private-ip: "true"` will be ignored. This will be indicated in the ingress event and AGIC pod log.

- Error as indicated in the Ingress Event

```
Events:
Type     Reason       Age               From
Message
----     ------       ----              ----
-------
Warning  NoPrivateIP  2m (x17 over 2m)  azure/application-gateway, prod-ingress-azure-5c9b6fcd4-bctcb
Ingress default/hello-world-ingress requires Application Gateway
applicationgateway3026 has a private IP address
```

- Error as indicated in AGIC Logs

```
E0730 18:57:37.914749       1 prune.go:65] Ingress default/hello-world-ingress requires Application
Gateway applicationgateway3026 has a private IP address
```

Assign Globally

In case, requirement is to restrict all Ingresses to be exposed over Private IP, use `appgw.usePrivateIP: true` in `helm` config.

Usage

```
appgw:
    subscriptionId: <subscriptionId>
    resourceGroup: <resourceGroupName>
    name: <applicationGatewayName>
    usePrivateIP: true
```

This will make the ingress controller filter the IP address configurations for a Private IP when configuring the frontend listeners on the Application Gateway. AGIC will panic and crash if `usePrivateIP: true` and no Private IP is assigned.

> **NOTE**
>
> Application Gateway v2 SKU requires a Public IP. Should you require Application Gateway to be private, Attach a `Network Security Group` to the Application Gateway's subnet to restrict traffic.

Add Health Probes to your service

8/22/2022 • 2 minutes to read • Edit Online

By default, Ingress controller will provision an HTTP GET probe for the exposed pods. The probe properties can be customized by adding a Readiness or Liveness Probe to your `deployment` / `pod` spec.

With `readinessProbe` or `livenessProbe`

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: aspnetapp
spec:
  replicas: 3
  template:
    metadata:
      labels:
        service: site
    spec:
      containers:
      - name: aspnetapp
        image: mcr.microsoft.com/dotnet/core/samples:aspnetapp
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
        readinessProbe:
          httpGet:
            path: /
            port: 80
          periodSeconds: 3
          timeoutSeconds: 1
```

Kubernetes API Reference:

- Container Probes

NOTE

- `readinessProbe` and `livenessProbe` are supported when configured with `httpGet`.
- Probing on a port other than the one exposed on the pod is currently not supported.
- `HttpHeaders`, `InitialDelaySeconds`, `SuccessThreshold` aren't supported.

Without `readinessProbe` or `livenessProbe`

If the above probes aren't provided, then the Ingress Controller makes an assumption that the service is reachable on the `Path` specified for `backend-path-prefix` annotation, or the `path` specified in the `ingress` definition for the service.

Default Values for Health Probe

For any property that can't be inferred by the readiness/liveness probe, default values are set.

APPLICATION GATEWAY PROBE PROPERTY	DEFAULT VALUE
`Path`	/
`Host`	localhost
`Protocol`	HTTP
`Timeout`	30
`Interval`	30
`UnhealthyThreshold`	3

Expose an AKS service over HTTP or HTTPS using Application Gateway

8/22/2022 • 3 minutes to read • Edit Online

These tutorials help illustrate the usage of Kubernetes Ingress Resources to expose an example Kubernetes service through the Azure Application Gateway over HTTP or HTTPS.

Prerequisites

- Installed `ingress-azure` helm chart.
  - **Greenfield Deployment**: If you're starting from scratch, refer to these installation instructions, which outlines steps to deploy an AKS cluster with Application Gateway and install application gateway ingress controller on the AKS cluster.
  - **Brownfield Deployment**: If you have an existing AKS cluster and Application Gateway, refer to these instructions to install application gateway ingress controller on the AKS cluster.
- If you want to use HTTPS on this application, you'll need an x509 certificate and its private key.

Deploy `guestbook` application

The guestbook application is a canonical Kubernetes application that composes of a Web UI frontend, a backend and a Redis database. By default, `guestbook` exposes its application through a service with name `frontend` on port `80`. Without a Kubernetes Ingress Resource, the service isn't accessible from outside the AKS cluster. We'll use the application and setup Ingress Resources to access the application through HTTP and HTTPS.

Follow the instructions below to deploy the guestbook application.

1. Download `guestbook-all-in-one.yaml` from here
2. Deploy `guestbook-all-in-one.yaml` into your AKS cluster by running

```
kubectl apply -f guestbook-all-in-one.yaml
```

Now, the `guestbook` application has been deployed.

Expose services over HTTP

In order to expose the guestbook application, we'll be using the following ingress resource:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: frontend
          servicePort: 80
```

This ingress will expose the `frontend` service of the `guestbook-all-in-one` deployment as a default backend of the Application Gateway.

Save the above ingress resource as `ing-guestbook.yaml`.

1. Deploy `ing-guestbook.yaml` by running:

```
kubectl apply -f ing-guestbook.yaml
```

2. Check the log of the ingress controller for deployment status.

Now the `guestbook` application should be available. You can check availability by visiting the public address of the Application Gateway.

Expose services over HTTPS

Without specified hostname

Without specifying hostname, the guestbook service will be available on all the host-names pointing to the application gateway.

1. Before deploying ingress, you need to create a kubernetes secret to host the certificate and private key. You can create a kubernetes secret by running

```
kubectl create secret tls <guestbook-secret-name> --key <path-to-key> --cert <path-to-cert>
```

2. Define the following ingress. In the ingress, specify the name of the secret in the `secretName` section.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  tls:
    - secretName: <guestbook-secret-name>
  rules:
  - http:
      paths:
      - backend:
          serviceName: frontend
          servicePort: 80
```

> **NOTE**
>
> Replace `<guestbook-secret-name>` in the above Ingress Resource with the name of your secret. Store the above Ingress Resource in a file name `ing-guestbook-tls.yaml`.

3. Deploy ing-guestbook-tls.yaml by running

```
kubectl apply -f ing-guestbook-tls.yaml
```

4. Check the log of the ingress controller for deployment status.

Now the `guestbook` application will be available on both HTTP and HTTPS.

With specified hostname

You can also specify the hostname on the ingress in order to multiplex TLS configurations and services. By specifying hostname, the guestbook service will only be available on the specified host.

1. Define the following ingress. In the ingress, specify the name of the secret in the `secretName` section and replace the hostname in the `hosts` section accordingly.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  tls:
    - hosts:
      - <guestbook.contoso.com>
      secretName: <guestbook-secret-name>
  rules:
  - host: <guestbook.contoso.com>
    http:
      paths:
      - backend:
          serviceName: frontend
          servicePort: 80
```

2. Deploy `ing-guestbook-tls-sni.yaml` by running

```
kubectl apply -f ing-guestbook-tls-sni.yaml
```

3. Check the log of the ingress controller for deployment status.

Now the `guestbook` application will be available on both HTTP and HTTPS only on the specified host (`<guestbook.contoso.com>` in this example).

Integrate with other services

The following ingress will allow you to add other paths into this ingress and redirect those paths to other services:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
  - http:
      paths:
      - path: </other/*>
        backend:
          serviceName: <other-service>
          servicePort: 80
       - backend:
          serviceName: frontend
          servicePort: 80
```

How to upgrade Application Gateway Ingress Controller using Helm

8/22/2022 • 2 minutes to read • Edit Online

The Azure Application Gateway Ingress Controller for Kubernetes (AGIC) can be upgraded using a Helm repository hosted on Azure Storage.

Before beginning the upgrade procedure, ensure that you've added the required repository:

- View your currently added Helm repositories with:

```
helm repo list
```

- Add the AGIC repo with:

```
helm repo add \
    application-gateway-kubernetes-ingress \
    https://appgwingress.blob.core.windows.net/ingress-azure-helm-package/
```

Upgrade

1. Refresh the AGIC Helm repository to get the latest release:

```
helm repo update
```

2. View available versions of the `application-gateway-kubernetes-ingress` chart:

```
helm search -l application-gateway-kubernetes-ingress
```

   Sample response:

```
NAME                                                    CHART VERSION   APP VERSION     DESCRIPTION
application-gateway-kubernetes-ingress/ingress-azure    0.7.0-rc1       0.7.0-rc1       Use Azure
Application Gateway as the ingress for an Azure...
application-gateway-kubernetes-ingress/ingress-azure    0.6.0           0.6.0           Use Azure
Application Gateway as the ingress for an Azure...
```

   Latest available version from the list above is: `0.7.0-rc1`

3. View the Helm charts currently installed:

```
helm list
```

   Sample response:

```
NAME            REVISION        UPDATED                         STATUS  CHART                   APP
VERSION     NAMESPACE
odd-billygoat   22              Fri Jun 21 15:56:06 2019        FAILED  ingress-azure-0.7.0-rc1
0.7.0-rc1       default
```

The Helm chart installation from the sample response above is named **odd-billygoat**. This name will be used for the commands. Your actual deployment name will be different.

4. Upgrade the Helm deployment to a new version:

```
helm upgrade \
    odd-billygoat \
    application-gateway-kubernetes-ingress/ingress-azure \
    --version 0.9.0-rc2
```

Rollback

If the Helm deployment fails, you can roll back to a previous release.

1. Get the last known healthy release number:

```
helm history odd-billygoat
```

Sample output:

```
REVISION        UPDATED                         STATUS          CHART                   DESCRIPTION
1               Mon Jun 17 13:49:42 2019        DEPLOYED        ingress-azure-0.6.0     Install
complete
2               Fri Jun 21 15:56:06 2019        FAILED          ingress-azure-xx        xxxx
```

Based on the sample output of the **helm history** command, the last successful deployment of our **odd-billygoat** was revision **1**.

2. Roll back to the last successful revision:

```
helm rollback odd-billygoat 1
```

Use certificates with LetsEncrypt.org on Application Gateway for AKS clusters

8/22/2022 • 3 minutes to read • Edit Online

This section configures your AKS to use LetsEncrypt.org and automatically obtain a TLS/SSL certificate for your domain. The certificate will be installed on Application Gateway, which will perform SSL/TLS termination for your AKS cluster. The setup described here uses the cert-manager Kubernetes add-on, which automates the creation and management of certificates.

Follow the steps below to install cert-manager on your existing AKS cluster.

1. Helm Chart

   Run the following script to install the `cert-manager` helm chart. This will:

   - create a new `cert-manager` namespace on your AKS
   - create the following CRDs: Certificate, Challenge, ClusterIssuer, Issuer, Order
   - install cert-manager chart (from docs.cert-manager.io)

```
#!/bin/bash

# Install the CustomResourceDefinition resources separately
kubectl apply -f https://raw.githubusercontent.com/jetstack/cert-manager/release-
0.8/deploy/manifests/00-crds.yaml

# Create the namespace for cert-manager
kubectl create namespace cert-manager

# Label the cert-manager namespace to disable resource validation
kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
# Helm v3+
helm install \
  cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --version v1.0.4 \
  # --set installCRDs=true

# Helm v2
helm install \
  --name cert-manager \
  --namespace cert-manager \
  --version v1.0.4 \
  jetstack/cert-manager \
  # --set installCRDs=true

#To automatically install and manage the CRDs as part of your Helm release,
#   you must add the --set installCRDs=true flag to your Helm installation command.
```

2. ClusterIssuer Resource

Create a `ClusterIssuer` resource. It's required by `cert-manager` to represent the `Lets Encrypt` certificate authority where the signed certificates will be obtained.

Using the non-namespaced `ClusterIssuer` resource, cert-manager will issue certificates that can be consumed from multiple namespaces. `Let's Encrypt` uses the ACME protocol to verify that you control a given domain name and to issue you a certificate. More details on configuring `ClusterIssuer` properties here. `ClusterIssuer` will instruct `cert-manager` to issue certificates using the `Lets Encrypt` staging environment used for testing (the root certificate not present in browser/client trust stores).

The default challenge type in the YAML below is `http01`. Other challenges are documented on letsencrypt.org - Challenge Types

> **IMPORTANT**
> Update `<YOUR.EMAIL@ADDRESS>` in the YAML below

```
#!/bin/bash
kubectl apply -f - <<EOF
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
name: letsencrypt-staging
spec:
acme:
    # You must replace this email address with your own.
    # Let's Encrypt will use this to contact you about expiring
    # certificates, and issues related to your account.
    email: <YOUR.EMAIL@ADDRESS>
    # ACME server URL for Let's Encrypt's staging environment.
    # The staging environment will not issue trusted certificates but is
    # used to ensure that the verification process is working properly
    # before moving to production
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
    # Secret resource used to store the account's private key.
    name: example-issuer-account-key
    # Enable the HTTP-01 challenge provider
    # you prove ownership of a domain by ensuring that a particular
    # file is present at the domain
    http01: {}
EOF
```

3. Deploy App

Create an Ingress resource to Expose the `guestbook` application using the Application Gateway with the Lets Encrypt Certificate.

Ensure your Application Gateway has a public Frontend IP configuration with a DNS name (either using the default `azure.com` domain, or provision a `Azure DNS Zone` service, and assign your own custom domain). Note the annotation `certmanager.k8s.io/cluster-issuer: letsencrypt-staging`, which tells cert-manager to process the tagged Ingress resource.

> **IMPORTANT**
> Update `<PLACEHOLDERS.COM>` in the YAML below with your own domain (or the Application Gateway one, for example 'kh-aks-ingress.westeurope.cloudapp.azure.com')

```
kubectl apply -f - <<EOF
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: guestbook-letsencrypt-staging
annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    certmanager.k8s.io/cluster-issuer: letsencrypt-staging
spec:
tls:
- hosts:
    - <PLACEHOLDERS.COM>
    secretName: guestbook-secret-name
rules:
- host: <PLACEHOLDERS.COM>
    http:
    paths:
    - backend:
        serviceName: frontend
        servicePort: 80
EOF
```

After a few seconds, you can access the `guestbook` service through the Application Gateway HTTPS url using the automatically issued **staging** `Lets Encrypt` certificate. Your browser may warn you of an invalid certificate authority. The staging certificate is issued by `CN=Fake LE Intermediate X1`. This warning is an indication that the system worked as expected and you're ready for your production certificate.

4. Production Certificate

   Once your staging certificate is set up successfully you can switch to a production ACME server:

   a. Replace the staging annotation on your Ingress resource with: `certmanager.k8s.io/cluster-issuer: letsencrypt-prod`
   b. Delete the existing staging `ClusterIssuer` you created in the previous step and create a new one by replacing the ACME server from the ClusterIssuer YAML above with `https://acme-v02.api.letsencrypt.org/directory`

5. Certificate Expiration and Renewal

   Before the `Lets Encrypt` certificate expires, `cert-manager` will automatically update the certificate in the Kubernetes secret store. At that point, Application Gateway Ingress Controller will apply the updated secret referenced in the ingress resources it's using to configure the Application Gateway.

Expose a WebSocket server to Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

As outlined in the Application Gateway v2 documentation - it provides native support for the WebSocket and HTTP/2 protocols. Please note, that for both Application Gateway and the Kubernetes Ingress - there is no user-configurable setting to selectively enable or disable WebSocket support.

The Kubernetes deployment YAML below shows the minimum configuration used to deploy a WebSocket server, which is the same as deploying a regular web server:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: websocket-server
spec:
  selector:
    matchLabels:
      app: ws-app
  replicas: 2
  template:
    metadata:
      labels:
        app: ws-app
    spec:
      containers:
        - name: websocket-app
          imagePullPolicy: Always
          image: your-container-repo.azurecr.io/websockets-app
          ports:
            - containerPort: 8888
      imagePullSecrets:
        - name: azure-container-registry-credentials

---

apiVersion: v1
kind: Service
metadata:
  name: websocket-app-service
spec:
  selector:
    app: ws-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8888

---

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: websocket-repeater
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
    - host: ws.contoso.com
      http:
        paths:
          - backend:
              serviceName: websocket-app-service
              servicePort: 80
```

Given that all the prerequisites are fulfilled, and you have an Application Gateway controlled by a Kubernetes Ingress in your AKS, the deployment above would result in a WebSockets server exposed on port 80 of your Application Gateway's public IP and the `ws.contoso.com` domain.

The following cURL command would test the WebSocket server deployment:

```
curl -i -N -H "Connection: Upgrade" \
        -H "Upgrade: websocket" \
        -H "Origin: http://localhost" \
        -H "Host: ws.contoso.com" \
        -H "Sec-Websocket-Version: 13" \
        -H "Sec-WebSocket-Key: 123" \
        http://1.2.3.4:80/ws
```

WebSocket Health Probes

If your deployment does not explicitly define health probes, Application Gateway would attempt an HTTP GET on your WebSocket server endpoint. Depending on the server implementation (here is one we love) WebSocket specific headers may be required (`Sec-Websocket-Version` for instance). Since Application Gateway does not add WebSocket headers, the Application Gateway's health probe response from your WebSocket server will most likely be `400 Bad Request`. As a result Application Gateway will mark your pods as unhealthy, which will eventually result in a `502 Bad Gateway` for the consumers of the WebSocket server. To avoid this you may need to add an HTTP GET handler for a health check to your server (`/health` for instance, which returns `200 OK`).

Autoscale your AKS pods using Application Gateway Metrics (Beta)

8/22/2022 • 2 minutes to read • Edit Online

As incoming traffic increases, it becomes crucial to scale up your applications based on the demand.

In the following tutorial, we explain how you can use Application Gateway's `AvgRequestCountPerHealthyHost` metric to scale up your application. `AvgRequestCountPerHealthyHost` measures average requests sent to a specific backend pool and backend HTTP setting combination.

We are going to use following two components:

- `Azure Kubernetes Metric Adapter` - We will use the metric adapter to expose Application Gateway metrics through the metric server. The Azure Kubernetes Metric Adapter is an open source project under Azure, similar to the Application Gateway Ingress Controller.
- `Horizontal Pod Autoscaler` - We will use HPA to use Application Gateway metrics and target a deployment for scaling.

Setting up Azure Kubernetes Metric Adapter

1. We will first create an Azure AAD service principal and assign it `Monitoring Reader` access over Application Gateway's resource group.

```
    applicationGatewayGroupName="<application-gateway-group-id>"
    applicationGatewayGroupId=$(az group show -g $applicationGatewayGroupName -o tsv --query "id")
    az ad sp create-for-rbac -n "azure-k8s-metric-adapter-sp" --role "Monitoring Reader" --scopes
applicationGatewayGroupId
```

2. Now, We will deploy the `Azure Kubernetes Metric Adapter` using the AAD service principal created above.

```
kubectl create namespace custom-metrics
# use values from service principal created above to create secret
kubectl create secret generic azure-k8s-metrics-adapter -n custom-metrics \
    --from-literal=azure-tenant-id=<tenantid> \
    --from-literal=azure-client-id=<clientid> \
    --from-literal=azure-client-secret=<secret>
kubectl apply -f kubectl apply -f https://raw.githubusercontent.com/Azure/azure-k8s-metrics-
adapter/master/deploy/adapter.yaml -n custom-metrics
```

3. We will create an `ExternalMetric` resource with name `appgw-request-count-metric`. This resource will instruct the metric adapter to expose `AvgRequestCountPerHealthyHost` metric for `myApplicationGateway` resource in `myResourceGroup` resource group. You can use the `filter` field to target a specific backend pool and backend HTTP setting in the Application Gateway.

```
apiVersion: azure.com/v1alpha2
kind: ExternalMetric
metadata:
name: appgw-request-count-metric
spec:
    type: azuremonitor
    azure:
        resourceGroup: myResourceGroup # replace with your application gateway's resource group name
        resourceName: myApplicationGateway # replace with your application gateway's name
        resourceProviderNamespace: Microsoft.Network
        resourceType: applicationGateways
    metric:
        metricName: AvgRequestCountPerHealthyHost
        aggregation: Average
        filter: BackendSettingsPool eq '<backend-pool-name>~<backend-http-setting-name>' # optional
```

You can now make a request to the metric server to see if our new metric is getting exposed:

```
kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/namespaces/default/appgw-request-count-metric"
# Sample Output
# {
#   "kind": "ExternalMetricValueList",
#   "apiVersion": "external.metrics.k8s.io/v1beta1",
#   "metadata":
#     {
#       "selfLink": "/apis/external.metrics.k8s.io/v1beta1/namespaces/default/appgw-request-count-metric",
#     },
#   "items":
#     [
#       {
#         "metricName": "appgw-request-count-metric",
#         "metricLabels": null,
#         "timestamp": "2019-11-05T00:18:51Z",
#         "value": "30",
#       },
#     ],
# }
```

Using the new metric to scale up the deployment

Once we are able to expose `appgw-request-count-metric` through the metric server, we are ready to use `Horizontal Pod Autoscaler` to scale up our target deployment.

In following example, we will target a sample deployment `aspnet`. We will scale up Pods when `appgw-request-count-metric` > 200 per Pod up to a max of `10` Pods.

Replace your target deployment name and apply the following auto scale configuration:

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: deployment-scaler
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: aspnet # replace with your deployment's name
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: External
    external:
      metricName: appgw-request-count-metric
      targetAverageValue: 200
```

Test your setup by using a load test tool like apache bench:

```
ab -n10000 http://<applicaiton-gateway-ip-address>/
```

Next steps

- **Troubleshoot Ingress Controller issues**: Troubleshoot any issues with the Ingress Controller.

Route web traffic based on the URL using Azure PowerShell

8/22/2022 • 9 minutes to read • Edit Online

You can use Azure PowerShell to configure web traffic routing to specific scalable server pools based on the URL that is used to access your application. In this article, you create an Azure Application Gateway with three backend pools using Virtual Machine Scale Sets. Each of the backend pools serves a specific purpose such as, common data, images, and video. Routing traffic to separate pools ensures that your customers get the information that they need when they need it.

To enable traffic routing, you create routing rules assigned to listeners that listen on specific ports to ensure web traffic arrives at the appropriate servers in the pools.

In this article, you learn how to:

- Set up the network
- Create listeners, URL path map, and rules
- Create scalable backend pools

If you prefer, you can complete this procedure using Azure CLI or the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this article requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az` . If you need to upgrade, see Install Azure PowerShell module. If you're running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Because of the time needed to create resources, it can take up to 90 minutes to complete this procedure.

Create a resource group

Create a resource group that contains all of the resources for your application.

Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Whether you have an existing virtual network or create a new one, you need to make sure that it contains a subnet that is only used for application gateways. In this article, you create a subnet for the application gateway and a subnet for the scale sets. You create a public IP address to enable access to the resources in the application gateway.

Create the subnet configurations *myAGSubnet* and *myBackendSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24

$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24

$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig
$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Static `
  -Sku Standard
```

Create an application gateway

In this section, you create resources that support the application gateway, and then finally create it. The resources you create include:

- *IP configurations and frontend port* - Associates the subnet you previously created to the application gateway and assigns a port to use to access it.
- *Default pool* - All application gateways must have at least one backend pool of servers.
- *Default listener and rule* - The default listener listens for traffic on the port that was assigned and the default rule sends traffic to the default pool.

Create the IP configurations and frontend port

Associate *myAGSubnet* you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$subnet=$vnet.Subnets[0]

$pip = Get-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Name myAGPublicIPAddress

$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet

$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip

$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the default pool and settings

Create the default backend pool named *appGatewayBackendPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend pool using New-AzApplicationGatewayBackendHttpSetting.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool

$poolSettings = New-AzApplicationGatewayBackendHttpSetting `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the default listener and rule

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this article, you create two listeners. The first basic listener that you create listens for traffic at the root URL. The second listener that you create listens for traffic at specific URLs.

Create the default listener named *myDefaultListener* using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created.

A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *rule1* using New-AzApplicationGatewayRequestRoutingRule.

```
$defaultlistener = New-AzApplicationGatewayHttpListener `
  -Name myDefaultListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport

$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultlistener `
  -BackendAddressPool $defaultPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_v2 `
  -Tier Standard_v2 `
  -Capacity 2

$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultlistener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku
```

It may take up to 30 minutes to create the application gateway. Wait until the deployment finishes successfully before moving on to the next section.

At this point, you have an application gateway that listens for traffic on port 80 and sends that traffic to a default server pool.

Add image and video backend pools and port

Add backend pools named *imagesBackendPool* and *videoBackendPool* to your application gateway using Add-AzApplicationGatewayBackendAddressPool. Add the frontend port for the pools using Add-AzApplicationGatewayFrontendPort. Submit the changes to the application gateway using Set-AzApplicationGateway.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

Add-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name imagesBackendPool

Add-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name videoBackendPool

Add-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name bport `
  -Port 8080

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Updating the application gateway can also take up to 20 minutes to finish.

Add backend listener

Add the backend listener named *backendListener* that's needed to route traffic using Add-AzApplicationGatewayHttpListener.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPort = Get-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name bport

$fipconfig = Get-AzApplicationGatewayFrontendIPConfig `
  -ApplicationGateway $appgw

Add-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name backendListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $backendPort

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add URL path map

URL path maps make sure that the URLs sent to your application are routed to specific backend pools. Create URL path maps named *imagePathRule* and *videoPathRule* using New-AzApplicationGatewayPathRuleConfig and Add-AzApplicationGatewayUrlPathMapConfig.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$poolSettings = Get-AzApplicationGatewayBackendHttpSetting `
  -ApplicationGateway $appgw `
  -Name myPoolSettings

$imagePool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name imagesBackendPool

$videoPool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name videoBackendPool

$defaultPool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name appGatewayBackendPool

$imagePathRule = New-AzApplicationGatewayPathRuleConfig `
  -Name imagePathRule `
  -Paths "/images/*" `
  -BackendAddressPool $imagePool `
  -BackendHttpSettings $poolSettings

$videoPathRule = New-AzApplicationGatewayPathRuleConfig `
  -Name videoPathRule `
    -Paths "/video/*" `
    -BackendAddressPool $videoPool `
    -BackendHttpSettings $poolSettings

Add-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name urlpathmap `
  -PathRules $imagePathRule, $videoPathRule `
  -DefaultBackendAddressPool $defaultPool `
  -DefaultBackendHttpSettings $poolSettings

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add routing rule

The routing rule associates the URL map with the listener that you created. Add the rule named *rule2* using Add-AzApplicationGatewayRequestRoutingRule.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendlistener = Get-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name backendListener

$urlPathMap = Get-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name urlpathmap

Add-AzApplicationGatewayRequestRoutingRule `
  -ApplicationGateway $appgw `
  -Name rule2 `
  -RuleType PathBasedRouting `
  -HttpListener $backendlistener `
  -UrlPathMap $urlPathMap

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Create virtual machine scale sets

In this example, you create three virtual machine scale sets that support the three backend pools that you created. The scale sets that you create are named *myvmss1*, *myvmss2*, and *myvmss3*. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool `
  -ApplicationGateway $appgw

$imagesPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name imagesBackendPool `
  -ApplicationGateway $appgw

$videoPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name videoBackendPool `
  -ApplicationGateway $appgw

for ($i=1; $i -le 3; $i++)
{
  if ($i -eq 1)
  {
     $poolId = $backendPool.Id
  }
  if ($i -eq 2)
  {
    $poolId = $imagesPool.Id
  }
  if ($i -eq 3)
  {
    $poolId = $videoPool.Id
  }

  $ipConfig = New-AzVmssIpConfig `
    -Name myVmssIPConfig$i `
```

```
    -SubnetId $vnet.Subnets[1].Id `
    -ApplicationGatewayBackendAddressPoolsId $poolId

  $vmssConfig = New-AzVmssConfig `
    -Location eastus `
    -SkuCapacity 2 `
    -SkuName Standard_DS2_v2 `
    -UpgradePolicyMode Automatic

  Set-AzVmssStorageProfile $vmssConfig `
    -ImageReferencePublisher MicrosoftWindowsServer `
    -ImageReferenceOffer WindowsServer `
    -ImageReferenceSku 2016-Datacenter `
    -ImageReferenceVersion latest `
    -OsDiskCreateOption FromImage

  Set-AzVmssOsProfile $vmssConfig `
    -AdminUsername azureuser `
    -AdminPassword "Azure123456!" `
    -ComputerNamePrefix myvmss$i

  Add-AzVmssNetworkInterfaceConfiguration `
    -VirtualMachineScaleSet $vmssConfig `
    -Name myVmssNetConfig$i `
    -Primary $true `
    -IPConfiguration $ipConfig

  New-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmssConfig
}
```

Install IIS

Each scale set contains two virtual machine instances on which you install IIS. A sample page is created to test if the application gateway is working.

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

for ($i=1; $i -le 3; $i++)
{
  $vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss$i
  Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
    -Name "customScript" `
    -Publisher "Microsoft.Compute" `
    -Type "CustomScriptExtension" `
    -TypeHandlerVersion 1.8 `
    -Setting $publicSettings

  Update-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmss
}
```

Test the application gateway

Use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://52.168.55.24`, `http://52.168.55.24:8080/images/test.htm`, or `http://52.168.55.24:8080/video/test.htm`.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Change the URL to http://<ip-address>:8080/images/test.htm, replacing your IP address for <ip-address>, and you should see something like the following example:

Change the URL to http://<ip-address>:8080/video/test.htm, replacing your IP address for <ip-address>, and you should see something like the following example:

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Redirect web traffic based on the URL

Route web traffic based on the URL using the Azure CLI

8/22/2022 • 6 minutes to read • Edit Online

As an IT administrator managing web traffic, you want to help your customers or users get the information they need as quickly as possible. One way you can optimize their experience is by routing different kinds of web traffic to different server resources. This article shows you how to use the Azure CLI to set up and configure Application Gateway routing for different types of traffic from your application. The routing then directs the traffic to different server pools based on the URL.

In this article, you learn how to:

- Create a resource group for the network resources you'll need
- Create the network resources
- Create an application gateway for the traffic coming from your application
- Specify server pools and routing rules for the different types of traffic
- Create a scale set for each pool so the pool can automatically scale
- Run a test so you can verify that the different types of traffic go to the correct pool

If you prefer, you can complete this procedure using Azure PowerShell or the Azure portal.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.

- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container where Azure resources are deployed and managed. Create a resource group using `az group create`.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using `az network vnet create`. Then add a subnet named *myBackendSubnet* that's needed by the backend servers using `az network vnet subnet create`. Create the public IP address named *myAGPublicIPAddress* using `az network public-ip create`.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24

az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard
```

Create the app gateway with a URL map

Use `az network application-gateway create` to create an application gateway named *myAppGateway*. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myAGPublicIPAddress*.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_v2 \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 80 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes to create the application gateway. After the application gateway is created, you can see these new features:

FEATURE	DESCRIPTION
appGatewayBackendPool	An application gateway must have at least one backend address pool.
appGatewayBackendHttpSettings	Specifies that port 80 and an HTTP protocol is used for communication.
appGatewayHttpListener	The default listener associated with appGatewayBackendPool
appGatewayFrontendIP	Assigns myAGPublicIPAddress to appGatewayHttpListener.
rule1	The default routing rule that is associated with appGatewayHttpListener.

Add image and video backend pools and a port

Add backend pools named *imagesBackendPool* and *videoBackendPool* to your application gateway by using `az network application-gateway address-pool create`. You add the frontend port for the pools using `az network application-gateway frontend-port create`.

```
az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name imagesBackendPool

az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name videoBackendPool

az network application-gateway frontend-port create \
  --port 8080 \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name port8080
```

Add a backend listener

Add the backend listener named *backendListener* that's needed to route traffic using `az network application-gateway http-listener create`.

```
az network application-gateway http-listener create \
  --name backendListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port port8080 \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway
```

Add a URL path map

URL path maps ensure that specific URLs are routed to specific backend pools. Create URL path maps named *imagePathRule* and *videoPathRule* using `az network application-gateway url-path-map create` and `az network application-gateway url-path-map rule create`.

```
az network application-gateway url-path-map create \
  --gateway-name myAppGateway \
  --name myPathMap \
  --paths /images/* \
  --resource-group myResourceGroupAG \
  --address-pool imagesBackendPool \
  --default-address-pool appGatewayBackendPool \
  --default-http-settings appGatewayBackendHttpSettings \
  --http-settings appGatewayBackendHttpSettings \
  --rule-name imagePathRule

az network application-gateway url-path-map rule create \
  --gateway-name myAppGateway \
  --name videoPathRule \
  --resource-group myResourceGroupAG \
  --path-map-name myPathMap \
  --paths /video/* \
  --address-pool videoBackendPool
```

Add a routing rule

The routing rule associates the URL maps with the listener that you created. Add a rule named *rule2* using `az network application-gateway rule create`.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name rule2 \
  --resource-group myResourceGroupAG \
  --http-listener backendListener \
  --rule-type PathBasedRouting \
  --url-path-map myPathMap \
  --address-pool appGatewayBackendPool
```

Create virtual machine scale sets

In this article, you create three virtual machine scale sets that support the three backend pools you created. You create scale sets named *myvmss1*, *myvmss2*, and *myvmss3*. Each scale set contains two virtual machine instances where you install NGINX.

```
for i in `seq 1 3`; do

  if [ $i -eq 1 ]
  then
    poolName="appGatewayBackendPool"
  fi

  if [ $i -eq 2 ]
  then
    poolName="imagesBackendPool"
  fi

  if [ $i -eq 3 ]
  then
    poolName="videoBackendPool"
  fi

  az vmss create \
    --name myvmss$i \
    --resource-group myResourceGroupAG \
    --image UbuntuLTS \
    --admin-username azureuser \
    --admin-password Azure123456! \
    --instance-count 2 \
    --vnet-name myVNet \
    --subnet myBackendSubnet \
    --vm-sku Standard_DS2 \
    --upgrade-policy-mode Automatic \
    --app-gateway myAppGateway \
    --backend-pool-name $poolName
done
```

Install NGINX

```
for i in `seq 1 3`; do
  az vmss extension set \
    --publisher Microsoft.Azure.Extensions \
    --version 2.0 \
    --name CustomScript \
    --resource-group myResourceGroupAG \
    --vmss-name myvmss$i \
    --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"], "commandToExecute": "./install_nginx.sh" }'
done
```

Test the application gateway

To get the public IP address of the application gateway, use az network public-ip show. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://40.121.222.19`, `http://40.121.222.19:8080/images/test.htm`, or `http://40.121.222.19:8080/video/test.htm`.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

Change the URL to http://<ip-address>:8080/images/test.html, replacing your IP address for <ip-address>, and you should see something like the following example:

Change the URL to http://<ip-address>:8080/video/test.html, replacing your IP address for <ip-address>, and you should see something like the following example.

Clean up resources

When they're no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroupAG
```

Next steps

Create an application gateway with URL path-based redirection

Create an application gateway that hosts multiple web sites using Azure PowerShell

8/22/2022 • 7 minutes to read • Edit Online

You can use Azure PowerShell to configure the hosting of multiple web sites when you create an application gateway. In this article, you define backend address pools using virtual machines scale sets. You then configure listeners and rules based on domains that you own to make sure web traffic arrives at the appropriate servers in the pools. This article assumes that you own multiple domains and uses examples of *www.contoso.com* and *www.fabrikam.com*.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Create backend listeners
- Create routing rules
- Create virtual machine scale sets with the backend pools
- Create a CNAME record in your domain

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this article requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az` . If you need to upgrade, see Install Azure PowerShell module. If you're running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Create the subnet configurations using New-AzVirtualNetworkSubnetConfig. Create the virtual network using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24

$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24

$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig

$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Dynamic
```

Create an application gateway

Create the IP configurations and frontend port

Associate the subnet that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign the public IP address to the application gateway using New-AzApplicationGatewayFrontendIPConfig.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$subnet=$vnet.Subnets[0]

$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet

$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip

$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the backend pools and settings

Create the first backend address pool for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the pool using New-AzApplicationGatewayBackendHttpSettings.

```
$contosoPool = New-AzApplicationGatewayBackendAddressPool `
  -Name contosoPool

$fabrikamPool = New-AzApplicationGatewayBackendAddressPool `
  -Name fabrikamPool

$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the listeners and rules

Listeners are required to enable the application gateway to route traffic appropriately to the backend address pools. In this article, you create two listeners for your two domains. Listeners are created for the *contoso.com* and *fabrikam.com* domains.

Create the first listener using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *contosoRule* using New-AzApplicationGatewayRequestRoutingRule.

> **NOTE**
>
> With Application Gateway or WAF v2 SKU, you can also configure up to 5 host names per listener and you can use wildcard characters in the host name. See wildcard host names in listener for more information. To use multiple host names and wildcard characters in a listener using Azure PowerShell, you must use `-HostNames` instead of `-HostName`. With HostNames, you can mention up to 5 host names as comma-separated values. For example, `-HostNames "*.contoso.com","*.fabrikam.com"`

```
$contosolistener = New-AzApplicationGatewayHttpListener `
  -Name contosoListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport `
  -HostName "www.contoso.com"

$fabrikamlistener = New-AzApplicationGatewayHttpListener `
  -Name fabrikamListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport `
  -HostName "www.fabrikam.com"

$contosoRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name contosoRule `
  -RuleType Basic `
  -HttpListener $contosoListener `
  -BackendAddressPool $contosoPool `
  -BackendHttpSettings $poolSettings

$fabrikamRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name fabrikamRule `
  -RuleType Basic `
  -HttpListener $fabrikamListener `
  -BackendAddressPool $fabrikamPool `
  -BackendHttpSettings $poolSettings
```

Add priority to routing rules

```
$contosoRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name wccontosoRule `
  -RuleType Basic `
  -Priority 200 `
  -HttpListener $wccontosoListener `
  -BackendAddressPool $wccontosoPool `
  -BackendHttpSettings $poolSettings

$fabrikamRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name shopcontosoRule `
  -RuleType Basic `
  -Priority 100 `
  -HttpListener $shopcontosoListener `
  -BackendAddressPool $shopcontosoPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_Medium `
  -Tier Standard `
  -Capacity 2

$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $contosoPool, $fabrikamPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $contosoListener, $fabrikamListener `
  -RequestRoutingRules $contosoRule, $fabrikamRule `
  -Sku $sku
```

Create virtual machine scale sets

In this example, you create two virtual machine scale sets that support the two backend pools that you created. The scale sets that you create are named *myvmss1* and *myvmss2*. Each scale set contains two virtual machine instances on which you install IIS. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$contosoPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name contosoPool `
  -ApplicationGateway $appgw

$fabrikamPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name fabrikamPool `
  -ApplicationGateway $appgw

for ($i=1; $i -le 2; $i++)
{
  if ($i -eq 1)
  {
    $poolId = $contosoPool.Id
  }
  if ($i -eq 2)
  {
    $poolId = $fabrikamPool.Id
  }

  $ipConfig = New-AzVmssIpConfig `
    -Name myVmssIPConfig$i `
    -SubnetId $vnet.Subnets[1].Id `
    -ApplicationGatewayBackendAddressPoolsId $poolId

  $vmssConfig = New-AzVmssConfig `
    -Location eastus `
    -SkuCapacity 2 `
    -SkuName Standard_DS2 `
    -UpgradePolicyMode Automatic

  Set-AzVmssStorageProfile $vmssConfig `
    -ImageReferencePublisher MicrosoftWindowsServer `
    -ImageReferenceOffer WindowsServer `
    -ImageReferenceSku 2016-Datacenter `
    -ImageReferenceVersion latest `
    -OsDiskCreateOption FromImage

  Set-AzVmssOsProfile $vmssConfig `
    -AdminUsername azureuser `
    -AdminPassword "Azure123456!" `
    -ComputerNamePrefix myvmss$i

  Add-AzVmssNetworkInterfaceConfiguration `
    -VirtualMachineScaleSet $vmssConfig `
    -Name myVmssNetConfig$i `
    -Primary $true `
    -IPConfiguration $ipConfig

  New-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmssConfig
}
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

for ($i=1; $i -le 2; $i++)
{
  $vmss = Get-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -VMScaleSetName myvmss$i

  Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
    -Name "customScript" `
    -Publisher "Microsoft.Compute" `
    -Type "CustomScriptExtension" `
    -TypeHandlerVersion 1.8 `
    -Setting $publicSettings

  Update-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmss
}
```

Create CNAME record in your domain

After the application gateway is created with its public IP address, you can get the DNS address and use it to create a CNAME record in your domain. You can use Get-AzPublicIPAddress to get the DNS address of the application gateway. Copy the *fqdn* value of the DNSSettings and use it as the value of the CNAME record that you create. Using A-records isn't recommended because the VIP may change when the application gateway is restarted in the V1 SKU.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Test the application gateway

Enter your domain name into the address bar of your browser. Such as, http://www.contoso.com.

Change the address to your other domain and you should see something like the following example:

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Create an application gateway with URL path-based routing rules

Create an application gateway that hosts multiple web sites using the Azure CLI

8/22/2022 • 7 minutes to read • Edit Online

You can use the Azure CLI to configure the hosting of multiple web sites when you create an application gateway. In this article, you define backend address pools using virtual machines scale sets. You then configure listeners and rules based on domains that you own to make sure web traffic arrives at the appropriate servers in the pools. This article assumes that you own multiple domains and uses examples of *www.contoso.com* and *www.fabrikam.com*.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Create backend listeners
- Create routing rules
- Create virtual machine scale sets with the backend pools
- Create a CNAME record in your domain

If you prefer, you can complete this procedure using Azure PowerShell.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.

- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet that's needed by the backend servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24

az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --allocation-method Static \
  --sku Standard
```

Create the application gateway

You can use az network application-gateway create to create the application gateway. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myAGPublicIPAddress* that you previously created.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_v2 \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 80 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features of it:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Add the backend pools

Add the backend pools that are needed to contain the backend servers using az network application-gateway address-pool create

```
az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name contosoPool

az network application-gateway address-pool create \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name fabrikamPool
```

Add listeners

Add listeners that are needed to route traffic using az network application-gateway http-listener create.

> **NOTE**
>
> With Application Gateway or WAF v2 SKU, you can also configure up to 5 host names per listener and you can use wildcard characters in the host name. See wildcard host names in listener for more information. To use multiple host names and wildcard characters in a listener using Azure CLI, you must use `--host-names` instead of `--host-name`. With host-names, you can mention up to five host names as space-separated values. For example, `--host-names "*.contoso.com *.fabrikam.com"`

```
az network application-gateway http-listener create \
  --name contosoListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port appGatewayFrontendPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway \
  --host-name www.contoso.com

az network application-gateway http-listener create \
  --name fabrikamListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port appGatewayFrontendPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway \
  --host-name www.fabrikam.com
```

Add routing rules

Rules are processed in the order they're listed if rule priority field is not used. Traffic is directed using the first rule that matches regardless of specificity. For example, if you have a rule using a basic listener and a rule using a multi-site listener both on the same port, the rule with the multi-site listener must be listed before the rule with the basic listener in order for the multi-site rule to function as expected.

In this example, you create two new rules and delete the default rule created when you deployed the application gateway. You can add the rule using az network application-gateway rule create.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name contosoRule \
  --resource-group myResourceGroupAG \
  --http-listener contosoListener \
  --rule-type Basic \
  --address-pool contosoPool

az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name fabrikamRule \
  --resource-group myResourceGroupAG \
  --http-listener fabrikamListener \
  --rule-type Basic \
  --address-pool fabrikamPool

az network application-gateway rule delete \
  --gateway-name myAppGateway \
  --name rule1 \
  --resource-group myResourceGroupAG
```

Add priority to routing rules

In order to ensure that more specific rules are processed first, use the rule priority field to ensure they have higher priority. Rule priority field must be set for all the existing request routing rules and any new rule that is created later must also have a rule priority value.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name wccontosoRule \
  --resource-group myResourceGroupAG \
  --http-listener wccontosoListener \
  --rule-type Basic \
  --priority 200 \
  --address-pool wccontosoPool

az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name shopcontosoRule \
  --resource-group myResourceGroupAG \
  --http-listener shopcontosoListener \
  --rule-type Basic \
  --priority 100 \
  --address-pool shopcontosoPool
```

Create virtual machine scale sets

In this example, you create three virtual machine scale sets that support the three backend pools in the application gateway. The scale sets that you create are named *myvmss1*, *myvmss2*, and *myvmss3*. Each scale set contains two virtual machine instances on which you install IIS.

```
for i in `seq 1 2`; do

  if [ $i -eq 1 ]
  then
    poolName="contosoPool"
  fi

  if [ $i -eq 2 ]
  then
    poolName="fabrikamPool"
  fi

  az vmss create \
    --name myvmss$i \
    --resource-group myResourceGroupAG \
    --image UbuntuLTS \
    --admin-username azureuser \
    --admin-password Azure123456! \
    --instance-count 2 \
    --vnet-name myVNet \
    --subnet myBackendSubnet \
    --vm-sku Standard_DS2 \
    --upgrade-policy-mode Automatic \
    --app-gateway myAppGateway \
    --backend-pool-name $poolName
done
```

Install NGINX

```
for i in `seq 1 2`; do

  az vmss extension set \
    --publisher Microsoft.Azure.Extensions \
    --version 2.0 \
    --name CustomScript \
    --resource-group myResourceGroupAG \
    --vmss-name myvmss$i \
    --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"],
  "commandToExecute": "./install_nginx.sh" }'

done
```

Create a CNAME record in your domain

After the application gateway is created with its public IP address, you can get the DNS address and use it to create a CNAME record in your domain. You can use az network public-ip show to get the DNS address of the application gateway. Copy the *fqdn* value of the DNSSettings and use it as the value of the CNAME record that you create.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [dnsSettings.fqdn] \
  --output tsv
```

The use of A-records isn't recommended because the VIP may change when the application gateway restarts.

Test the application gateway

Enter your domain name into the address bar of your browser. Such as, http://www.contoso.com.

Change the address to your other domain and you should see something like the following example:

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources.

```
az group delete --name myResourceGroupAG
```

Next steps

Create an application gateway with URL path-based routing rules

Create an application gateway with external redirection using Azure PowerShell

8/22/2022 • 4 minutes to read • Edit Online

You can use Azure PowerShell to configure web traffic redirection when you create an application gateway. In this tutorial, you configure a listener and rule that redirects web traffic that arrives at the application gateway to an external site.

In this article, you learn how to:

- Set up the network
- Create a listener and redirection rule
- Create an application gateway

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux, or by selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this tutorial requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az` . If you need to upgrade, see Install Azure PowerShell module. If you are running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Create the subnet configuration *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configuration. And finally, create the public IP address using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.1.0/24
$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $agSubnetConfig
$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Dynamic
```

Create an application gateway

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign the public IP address to the application gateway using New-AzApplicationGatewayFrontendIPConfig. And then you can create the HTTP port using New-AzApplicationGatewayFrontendPort.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet
$subnet=$vnet.Subnets[0]
$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet
$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip
$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the backend pool and settings

Create the backend pool named *defaultPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the pool using New-AzApplicationGatewayBackendHttpSettings.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name defaultPool
$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the listener and rule

A listener is required to enable the application gateway to appropriately route traffic. Create the listener using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created. A rule is required for the listener to know where to send incoming traffic. Create a basic rule named *redirectRule* using New-AzApplicationGatewayRequestRoutingRule.

```
$defaultListener = New-AzApplicationGatewayHttpListener `
  -Name defaultListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport
$redirectConfig = New-AzApplicationGatewayRedirectConfiguration `
  -Name myredirect `
  -RedirectType Temporary `
  -TargetUrl "https://bing.com"
$redirectRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name redirectRule `
  -RuleType Basic `
  -HttpListener $defaultListener `
  -RedirectConfiguration $redirectConfig
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_Medium `
  -Tier Standard `
  -Capacity 2
$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultListener `
  -RequestRoutingRules $redirectRule `
  -RedirectConfigurations $redirectConfig `
  -Sku $sku
```

Test the application gateway

You can use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

You should see *bing.com* appear in your browser.

Next steps

- Create an application gateway with internal redirection using Azure PowerShell

Create an application gateway with external redirection using the Azure CLI

8/22/2022 • 3 minutes to read • Edit Online

You can use the Azure CLI to configure web traffic redirection when you create an application gateway. In this tutorial, you configure a listener and rule that redirects web traffic that arrives at the application gateway to an external site.

In this article, you learn how to:

- Set up the network
- Create a listener and redirection rule
- Create an application gateway

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24
az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress
```

Create an application gateway

You can use az network application-gateway create to create the application gateway named *myAppGateway*. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myPublicIPAddress* that you previously created.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_Medium \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 8080 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features of it:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Add the redirection configuration

Add the redirection configuration that sends traffic from *www.consoto.org* to the listener for *www.contoso.com* to the application gateway using az network application-gateway redirect-config create.

```
az network application-gateway redirect-config create \
  --name myredirect \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --type Temporary \
  --target-url "https://bing.com"
```

Add a listener and routing rule

A listener is required to enable the application gateway to appropriately route traffic. Create the listener using az network application-gateway http-listener create with the frontend port created with az network application-gateway frontend-port create. A rule is required for the listener to know where to send incoming traffic. Create a basic rule named *redirectRule* using az network application-gateway rule create.

```
az network application-gateway frontend-port create \
  --port 80 \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name redirectPort
az network application-gateway http-listener create \
  --name redirectListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port redirectPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name redirectRule \
  --resource-group myResourceGroupAG \
  --http-listener redirectListener \
  --rule-type Basic \
  --redirect-config myredirect
```

Test the application gateway

To get the public IP address of the application gateway, you can use az network public-ip show. Copy the public IP address, and then paste it into the address bar of your browser.

You should see *bing.com* appear in your browser.

Next steps

- Create an application gateway with internal redirection using the Azure CLI

Create an application gateway with HTTP to HTTPS redirection using the Azure portal

8/22/2022 • 6 minutes to read • Edit Online

You can use the Azure portal to create an application gateway with a certificate for TLS termination. A routing rule is used to redirect HTTP traffic to the HTTPS port in your application gateway. In this example, you also create a virtual machine scale set for the backend pool of the application gateway that contains two virtual machine instances.

In this article, you learn how to:

- Create a self-signed certificate
- Set up a network
- Create an application gateway with the certificate
- Add a listener and redirection rule
- Create a virtual machine scale set with the default backend pool

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This tutorial requires the Azure PowerShell module version 1.0.0 or later to create a certificate and install IIS. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. To run the commands in this tutorial, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a self-signed certificate

For production use, you should import a valid certificate signed by a trusted provider. For this tutorial, you create a self-signed certificate using New-SelfSignedCertificate. You can use Export-PfxCertificate with the Thumbprint that was returned to export a pfx file from the certificate.

```
New-SelfSignedCertificate `
  -certstorelocation cert:\localmachine\my `
  -dnsname www.contoso.com
```

You should see something like this result:

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject
----------                                -------
E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630  CN=www.contoso.com
```

Use the thumbprint to create the pfx file:

```
$pwd = ConvertTo-SecureString -String "Azure123456!" -Force -AsPlainText
Export-PfxCertificate `
  -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `
  -FilePath c:\appgwcert.pfx `
  -Password $pwd
```

Create an application gateway

A virtual network is needed for communication between the resources that you create. Two subnets are created in this example: one for the application gateway, and the other for the backend servers. You can create a virtual network at the same time that you create the application gateway.

1. Sign in to the Azure portal at https://portal.azure.com.
2. Click **Create a resource** found on the upper left-hand corner of the Azure portal.
3. Select **Networking** and then select **Application Gateway** in the Featured list.
4. Enter these values for the application gateway:
   - *myAppGateway* - for the name of the application gateway.
   - *myResourceGroupAG* - for the new resource group.

5. Accept the default values for the other settings and then click **OK**.
6. Click **Choose a virtual network**, click **Create new**, and then enter these values for the virtual network:
   - *myVNet* - for the name of the virtual network.

- *10.0.0.0/16* - for the virtual network address space.
- *myAGSubnet* - for the subnet name.
- *10.0.0.0/24* - for the subnet address space.

7. Click **OK** to create the virtual network and subnet.
8. Under **Frontend IP configuration**, ensure **IP address type** is **Public**, and **Create new** is selected. Enter *myAGPublicIPAddress* for the name. Accept the default values for the other settings and then click **OK**.
9. Under **Listener configuration**, select **HTTPS**, then select **Select a file** and navigate to the *c:\appgwcert.pfx* file and select **Open**.
10. Type *appgwcert* for the cert name and *Azure123456!* for the password.
11. Leave the Web application firewall disabled, and then select **OK**.
12. Review the settings on the summary page, and then select **OK** to create the network resources and the application gateway. It may take several minutes for the application gateway to be created, wait until the deployment finishes successfully before moving on to the next section.

Add a subnet

1. Select **All resources** in the left-hand menu, and then select **myVNet** from the resources list.
2. Select **Subnets**, and then click **Subnet**.

3. Type *myBackendSubnet* for the name of the subnet.

4. Type *10.0.2.0/24* for the address range, and then select **OK**.

Add a listener and redirection rule

Add the listener

First, add the listener named *myListener* for port 80.

1. Open the **myResourceGroupAG** resource group and select **myAppGateway**.
2. Select **Listeners** and then select **+ Basic**.
3. Type *MyListener* for the name.
4. Type *httpPort* for the new frontend port name and *80* for the port.
5. Ensure the protocol is set to **HTTP**, and then select **OK**.

Add a routing rule with a redirection configuration

1. On **myAppGateway**, select **Rules** and then select **+Request routing rule**.
2. For the **Rule name**, type *Rule2*.
3. Ensure **MyListener** is selected for the listener.
4. Click on **Backend targets** tab and select **Target type** as *Redirection*.
5. For **Redirection type**, select **Permanent**.
6. For **Redirection target**, select **Listener**.
7. Ensure the **Target listener** is set to **appGatewayHttpListener**.
8. For the **Include query string** and **Include path** select *Yes*.
9. Select **Add**.

Create a virtual machine scale set

In this example, you create a virtual machine scale set to provide servers for the backend pool in the application gateway.

1. On the portal upper left corner, select **+Create a resource**.
2. Select **Compute**.
3. In the search box, type *scale set* and press Enter.
4. Select **Virtual machine scale set**, and then select **Create**.
5. For **Virtual machine scale set name**, type *myvmss*.
6. For **Operating system disk image**, ensure **Windows Server 2016 Datacenter** is selected.
7. For **Resource group**, select **myResourceGroupAG**.
8. For **User name**, type *azureuser*.
9. For **Password**, type *Azure123456!* and confirm the password.
10. For **Instance count**, ensure the value is **2**.
11. For **Instance size**, select **D2s_v3**.
12. Under **Networking**, ensure **Choose Load balancing options** is set to **Application Gateway**.
13. Ensure **Application gateway** is set to **myAppGateway**.
14. Ensure **Subnet** is set to **myBackendSubnet**.
15. Select **Create**.

Associate the scale set with the proper backend pool

The virtual machine scale set portal UI creates a new backend pool for the scale set, but you want to associate it with your existing appGatewayBackendPool.

1. Open the **myResourceGroupAg** resource group.
2. Select **myAppGateway**.
3. Select **Backend pools**.
4. Select **myAppGatewaymyvmss**.
5. Select **Remove all targets from backend pool**.
6. Select **Save**.
7. After this process completes, select the **myAppGatewaymyvmss** backend pool, select **Delete** and then **OK** to confirm.
8. Select **appGatewayBackendPool**.
9. Under **Targets**, select **VMSS**.
10. Under **VMSS**, select **myvmss**.
11. Under **Network Interface Configurations**, select **myvmssNic**.
12. Select **Save**.

Upgrade the scale set

Finally, you must upgrade the scale set with these changes.

1. Select the **myvmss** scale set.
2. Under **Settings**, select **Instances**.
3. Select both instances, and then select **Upgrade**.
4. Select **Yes** to confirm.
5. After this completes, go back to the **myAppGateway** and select **Backend pools**. You should now see that the **appGatewayBackendPool** has two targets, and **myAppGatewaymyvmss** has zero targets.
6. Select **myAppGatewaymyvmss**, and then select **Delete**.
7. Select **OK** to confirm.

Install IIS

An easy way to install IIS on the scale set is to use PowerShell. From the portal, click the Cloud Shell icon and ensure that **PowerShell** is selected.

Paste the following code into the PowerShell window and press Enter.

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }
$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss
Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
  -Name "customScript" `
  -Publisher "Microsoft.Compute" `
  -Type "CustomScriptExtension" `
  -TypeHandlerVersion 1.8 `
  -Setting $publicSettings
Update-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmss
```

Upgrade the scale set

After changing the instances with IIS , you must once again upgrade the scale set with this change.

1. Select the **myvmss** scale set.
2. Under **Settings**, select **Instances**.
3. Select both instances, and then select **Upgrade**.
4. Select **Yes** to confirm.

Test the application gateway

You can get the application public IP address from the application gateway Overview page.

1. Select **myAppGateway**.
2. On the **Overview** page, note the IP address under **Frontend public IP address**.
3. Copy the public IP address, and then paste it into the address bar of your browser. For example, http://52.170.203.149

4. To accept the security warning if you used a self-signed certificate, select **Details** and then **Go on to the webpage**. Your secured IIS website is then displayed as in the following example:

Next steps

Learn how to Create an application gateway with internal redirection.

Create an application gateway with HTTP to HTTPS redirection using Azure PowerShell

8/22/2022 • 6 minutes to read • Edit Online

You can use the Azure PowerShell to create an application gateway with a certificate for TLS/SSL termination. A routing rule is used to redirect HTTP traffic to the HTTPS port in your application gateway. In this example, you also create a virtual machine scale set for the backend pool of the application gateway that contains two virtual machine instances.

In this article, you learn how to:

- Create a self-signed certificate
- Set up a network
- Create an application gateway with the certificate
- Add a listener and redirection rule
- Create a virtual machine scale set with the default backend pool

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This tutorial requires the Azure PowerShell module version 1.0.0 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. To run the commands in this tutorial, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a self-signed certificate

For production use, you should import a valid certificate signed by a trusted provider. For this tutorial, you create a self-signed certificate using New-SelfSignedCertificate. You can use Export-PfxCertificate with the Thumbprint that was returned to export a pfx file from the certificate.

```
New-SelfSignedCertificate `
  -certstorelocation cert:\localmachine\my `
  -dnsname www.contoso.com
```

You should see something like this result:

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject
----------                                -------
E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630  CN=www.contoso.com
```

Use the thumbprint to create the pfx file:

```
$pwd = ConvertTo-SecureString -String "Azure123456!" -Force -AsPlainText
Export-PfxCertificate `
  -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `
  -FilePath c:\appgwcert.pfx `
  -Password $pwd
```

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group named *myResourceGroupAG* using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Create the subnet configurations for *myBackendSubnet* and *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24
$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24
$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig
$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Dynamic
```

Create an application gateway

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig. And then you can create the HTTPS port using New-AzApplicationGatewayFrontendPort.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet
$subnet=$vnet.Subnets[0]
$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet
$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip
$frontendPort = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 443
```

Create the backend pool and settings

Create the backend pool named *appGatewayBackendPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend pool using New-AzApplicationGatewayBackendHttpSettings.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool
$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the default listener and rule

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this example, you create a basic listener that listens for HTTPS traffic at the root URL.

Create a certificate object using New-AzApplicationGatewaySslCertificate and then create a listener named *appGatewayHttpListener* using New-AzApplicationGatewayHttpListener with the frontend configuration, frontend port, and certificate that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *rule1* using New-AzApplicationGatewayRequestRoutingRule.

```
$pwd = ConvertTo-SecureString `
  -String "Azure123456!" `
  -Force `
  -AsPlainText
$cert = New-AzApplicationGatewaySslCertificate `
  -Name "appgwcert" `
  -CertificateFile "c:\appgwcert.pfx" `
  -Password $pwd
$defaultListener = New-AzApplicationGatewayHttpListener `
  -Name appGatewayHttpListener `
  -Protocol Https `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendPort `
  -SslCertificate $cert
$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultListener `
  -BackendAddressPool $defaultPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway with the certificate.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_Medium `
  -Tier Standard `
  -Capacity 2
$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendPort `
  -HttpListeners $defaultListener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku `
  -SslCertificates $cert
```

Add a listener and redirection rule

Add the HTTP port

Add the HTTP port to the application gateway using Add-AzApplicationGatewayFrontendPort.

```
$appgw = Get-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG
Add-AzApplicationGatewayFrontendPort `
  -Name httpPort  `
  -Port 80 `
  -ApplicationGateway $appgw
```

Add the HTTP listener

Add the HTTP listener named *myListener* to the application gateway using Add-AzApplicationGatewayHttpListener.

```
$fipconfig = Get-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -ApplicationGateway $appgw
$fp = Get-AzApplicationGatewayFrontendPort `
  -Name httpPort `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayHttpListener `
  -Name myListener `
  -Protocol Http `
  -FrontendPort $fp `
  -FrontendIPConfiguration $fipconfig `
  -ApplicationGateway $appgw
```

Add the redirection configuration

Add the HTTP to HTTPS redirection configuration to the application gateway using Add-AzApplicationGatewayRedirectConfiguration.

```
$defaultListener = Get-AzApplicationGatewayHttpListener `
  -Name appGatewayHttpListener `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayRedirectConfiguration -Name httpToHttps `
  -RedirectType Permanent `
  -TargetListener $defaultListener `
  -IncludePath $true `
  -IncludeQueryString $true `
  -ApplicationGateway $appgw
```

Add the routing rule

Add the routing rule with the redirection configuration to the application gateway using Add-AzApplicationGatewayRequestRoutingRule.

```
$myListener = Get-AzApplicationGatewayHttpListener `
  -Name myListener `
  -ApplicationGateway $appgw
$redirectConfig = Get-AzApplicationGatewayRedirectConfiguration `
  -Name httpToHttps `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayRequestRoutingRule `
  -Name rule2 `
  -RuleType Basic `
  -HttpListener $myListener `
  -RedirectConfiguration $redirectConfig `
  -ApplicationGateway $appgw
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Create a virtual machine scale set

In this example, you create a virtual machine scale set to provide servers for the backend pool in the application gateway. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway
$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool `
  -ApplicationGateway $appgw
$ipConfig = New-AzVmssIpConfig `
  -Name myVmssIPConfig `
  -SubnetId $vnet.Subnets[1].Id `
  -ApplicationGatewayBackendAddressPoolsId $backendPool.Id
$vmssConfig = New-AzVmssConfig `
  -Location eastus `
  -SkuCapacity 2 `
  -SkuName Standard_DS2 `
  -UpgradePolicyMode Automatic
Set-AzVmssStorageProfile $vmssConfig `
  -ImageReferencePublisher MicrosoftWindowsServer `
  -ImageReferenceOffer WindowsServer `
  -ImageReferenceSku 2016-Datacenter `
  -ImageReferenceVersion latest `
  -OsDiskCreateOption FromImage
Set-AzVmssOsProfile $vmssConfig `
  -AdminUsername azureuser `
  -AdminPassword "Azure123456!" `
  -ComputerNamePrefix myvmss
Add-AzVmssNetworkInterfaceConfiguration `
  -VirtualMachineScaleSet $vmssConfig `
  -Name myVmssNetConfig `
  -Primary $true `
  -IPConfiguration $ipConfig
New-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmssConfig
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }
$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss
Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
  -Name "customScript" `
  -Publisher "Microsoft.Compute" `
  -Type "CustomScriptExtension" `
  -TypeHandlerVersion 1.8 `
  -Setting $publicSettings
Update-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmss
```

Test the application gateway

You can use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser. For example, http://52.170.203.149

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

To accept the security warning if you used a self-signed certificate, select **Details** and then **Go on to the webpage**. Your secured IIS website is then displayed as in the following example:

Next steps

- Rewrite HTTP headers and URL with Application Gateway

Create an application gateway with HTTP to HTTPS redirection using the Azure CLI

8/22/2022 • 5 minutes to read • Edit Online

You can use the Azure CLI to create an application gateway with a certificate for TLS/SSL termination. A routing rule is used to redirect HTTP traffic to the HTTPS port in your application gateway. In this example, you also create a virtual machine scale set for the backend pool of the application gateway that contains two virtual machine instances.

In this article, you learn how to:

- Create a self-signed certificate
- Set up a network
- Create an application gateway with the certificate
- Add a listener and redirection rule
- Create a virtual machine scale set with the default backend pool

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a self-signed certificate

For production use, you should import a valid certificate signed by a trusted provider. For this tutorial, you create a self-signed certificate and pfx file using the openssl command.

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out appgwcert.crt
```

Enter values that make sense for your certificate. You can accept the default values.

```
openssl pkcs12 -export -out appgwcert.pfx -inkey privateKey.key -in appgwcert.crt
```

Enter the password for the certificate. In this example, *Azure123456!* is being used.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet named *myBackendSubnet* that's needed by the backend servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24
az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24
az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress
```

Create the application gateway

You can use az network application-gateway create to create the application gateway named *myAppGateway*. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings.

The application gateway is assigned to *myAGSubnet* and *myAGPublicIPAddress* that you previously created. In this example, you associate the certificate that you created and its password when you create the application gateway.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_Medium \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 443 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress \
  --cert-file appgwcert.pfx \
  --cert-password "Azure123456!"
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features of it:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Add a listener and redirection rule

Add the HTTP port

You can use az network application-gateway frontend-port create to add the HTTP port to the application gateway.

```
az network application-gateway frontend-port create \
  --port 80 \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --name httpPort
```

Add the HTTP listener

You can use az network application-gateway http-listener create to add the listener named *myListener* to the application gateway.

```
az network application-gateway http-listener create \
  --name myListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port httpPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway
```

Add the redirection configuration

Add the HTTP to HTTPS redirection configuration to the application gateway using az network application-gateway redirect-config create.

```
az network application-gateway redirect-config create \
  --name httpToHttps \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --type Permanent \
  --target-listener appGatewayHttpListener \
  --include-path true \
  --include-query-string true
```

Add the routing rule

Add the routing rule named *rule2* with the redirection configuration to the application gateway using az network application-gateway rule create.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name rule2 \
  --resource-group myResourceGroupAG \
  --http-listener myListener \
  --rule-type Basic \
  --redirect-config httpToHttps
```

Create a virtual machine scale set

In this example, you create a virtual machine scale set named *myvmss* that provides servers for the backend pool in the application gateway. The virtual machines in the scale set are associated with *myBackendSubnet* and *appGatewayBackendPool*. To create the scale set, you can use az vmss create.

```
az vmss create \
  --name myvmss \
  --resource-group myResourceGroupAG \
  --image UbuntuLTS \
  --admin-username azureuser \
  --admin-password Azure123456! \
  --instance-count 2 \
  --vnet-name myVNet \
  --subnet myBackendSubnet \
  --vm-sku Standard_DS2 \
  --upgrade-policy-mode Automatic \
  --app-gateway myAppGateway \
  --backend-pool-name appGatewayBackendPool
```

Install NGINX

```
az vmss extension set \
  --publisher Microsoft.Azure.Extensions \
  --version 2.0 \
  --name CustomScript \
  --resource-group myResourceGroupAG \
  --vmss-name myvmss \
  --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"],
  "commandToExecute": "./install_nginx.sh" }'
```

Test the application gateway

To get the public IP address of the application gateway, you can use az network public-ip show. Copy the public IP address, and then paste it into the address bar of your browser.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [ipAddress] \
  --output tsv
```

To accept the security warning if you used a self-signed certificate, select **Details** and then **Go on to the webpage**. Your secured NGINX site is then displayed as in the following example:

Next steps

- Create an application gateway with internal redirection using the Azure CLI

Create an application gateway with internal redirection using Azure PowerShell

8/22/2022 • 7 minutes to read • Edit Online

You can use Azure PowerShell to configure web traffic redirection when you create an application gateway. In this article, you define a backend pool using a virtual machines scale set. You then configure listeners and rules based on domains that you own to make sure web traffic arrives at the appropriate pool. This article assumes that you own multiple domains and uses examples of *www.contoso.com* and *www.contoso.org*.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Add listeners and redirection rule
- Create a virtual machine scale set with the backend pool
- Create a CNAME record in your domain

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.

2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this article requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az` . If you need to upgrade, see Install Azure PowerShell module. If you are running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Create the subnet configurations for *myBackendSubnet* and *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24
$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24
$vnet = New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig
$pip = New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Dynamic
```

Create an application gateway

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig. And then you can create the HTTP port using New-AzApplicationGatewayFrontendPort.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet
$subnet=$vnet.Subnets[0]
$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet
$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip
$frontendPort = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the backend pool and settings

Create a backend pool named *contosoPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend pool using New-AzApplicationGatewayBackendHttpSettings.

```
$contosoPool = New-AzApplicationGatewayBackendAddressPool `
  -Name contosoPool
$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the first listener and rule

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this article, you create two listeners for your two domains. In this example, listeners are created for the domains of *www.contoso.com* and *www.contoso.org*.

Create the first listener named *contosoComListener* using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *contosoComRule* using New-AzApplicationGatewayRequestRoutingRule.

```
$contosoComlistener = New-AzApplicationGatewayHttpListener `
  -Name contosoComListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendPort `
  -HostName "www.contoso.com"
$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name contosoComRule `
  -RuleType Basic `
  -HttpListener $contosoComListener `
  -BackendAddressPool $contosoPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_Medium `
  -Tier Standard `
  -Capacity 2
$appgw = New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $contosoPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendPort `
  -HttpListeners $contosoComListener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku
```

Add the second listener

Add the listener named *contosoOrgListener* that's needed to redirect traffic using Add-AzApplicationGatewayHttpListener.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway
$frontendPort = Get-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -ApplicationGateway $appgw
$ipconfig = Get-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name contosoOrgListener `
  -Protocol Http `
  -FrontendIPConfiguration $ipconfig `
  -FrontendPort $frontendPort `
  -HostName "www.contoso.org"
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add the redirection configuration

You can configure redirection for the listener using Add-AzApplicationGatewayRedirectConfiguration.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway
$contosoComlistener = Get-AzApplicationGatewayHttpListener `
  -Name contosoComListener `
  -ApplicationGateway $appgw
$contosoOrglistener = Get-AzApplicationGatewayHttpListener `
  -Name contosoOrgListener `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayRedirectConfiguration `
  -ApplicationGateway $appgw `
  -Name redirectOrgtoCom `
  -RedirectType Found `
  -TargetListener $contosoComListener `
  -IncludePath $true `
  -IncludeQueryString $true
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add the second routing rule

You can then associate the redirection configuration to a new rule named *contosoOrgRule* using Add-AzApplicationGatewayRequestRoutingRule.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway
$contosoOrglistener = Get-AzApplicationGatewayHttpListener `
  -Name contosoOrgListener `
  -ApplicationGateway $appgw
$redirectConfig = Get-AzApplicationGatewayRedirectConfiguration `
  -Name redirectOrgtoCom `
  -ApplicationGateway $appgw
Add-AzApplicationGatewayRequestRoutingRule `
  -ApplicationGateway $appgw `
  -Name contosoOrgRule `
  -RuleType Basic `
  -HttpListener $contosoOrgListener `
  -RedirectConfiguration $redirectConfig
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Create a virtual machine scale set

In this example, you create a virtual machine scale set that supports the backend pool that you created. The scale set that you create is named *myvmss* and contains two virtual machine instances on which you install IIS. You assign the scale set to the backend pool when you configure the IP settings.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway
$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name contosoPool `
  -ApplicationGateway $appgw
$ipConfig = New-AzVmssIpConfig `
  -Name myVmssIPConfig `
  -SubnetId $vnet.Subnets[1].Id `
  -ApplicationGatewayBackendAddressPoolsId $backendPool.Id
$vmssConfig = New-AzVmssConfig `
  -Location eastus `
  -SkuCapacity 2 `
  -SkuName Standard_DS2 `
  -UpgradePolicyMode Automatic
Set-AzVmssStorageProfile $vmssConfig `
  -ImageReferencePublisher MicrosoftWindowsServer `
  -ImageReferenceOffer WindowsServer `
  -ImageReferenceSku 2016-Datacenter `
  -ImageReferenceVersion latest `
  -OsDiskCreateOption FromImage
Set-AzVmssOsProfile $vmssConfig `
  -AdminUsername azureuser `
  -AdminPassword "Azure123456!" `
  -ComputerNamePrefix myvmss
Add-AzVmssNetworkInterfaceConfiguration `
  -VirtualMachineScaleSet $vmssConfig `
  -Name myVmssNetConfig `
  -Primary $true `
  -IPConfiguration $ipConfig
New-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmssConfig
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }
$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss
Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
  -Name "customScript" `
  -Publisher "Microsoft.Compute" `
  -Type "CustomScriptExtension" `
  -TypeHandlerVersion 1.8 `
  -Setting $publicSettings
Update-AzVmss `
  -ResourceGroupName myResourceGroupAG `
  -Name myvmss `
  -VirtualMachineScaleSet $vmss
```

Create CNAME record in your domain

After the application gateway is created with its public IP address, you can get the DNS address and use it to create a CNAME record in your domain. You can use Get-AzPublicIPAddress to get the DNS address of the application gateway. Copy the *fqdn* value of the DNSSettings and use it as the value of the CNAME record that you create. The use of A-records is not recommended because the VIP may change when the application gateway is restarted.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Test the application gateway

Enter your domain name into the address bar of your browser. Such as, `https://www.contoso.com`.

Change the address to your other domain, for example `https://www.contoso.org` and you should see that the traffic has been redirected back to the listener for www.contoso.com.

Next steps

- Application Gateway redirect overview

Create an application gateway with internal redirection using the Azure CLI

8/22/2022 • 6 minutes to read • Edit Online

You can use the Azure CLI to configure web traffic redirection when you create an application gateway. In this tutorial, you define a backend pool using a virtual machines scale set. You then configure listeners and rules based on domains that you own to make sure web traffic arrives at the appropriate pool. This tutorial assumes that you own multiple domains and uses examples of *www.contoso.com* and *www.contoso.org*.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Add listeners and redirection rule
- Create a virtual machine scale set with the backend pool
- Create a CNAME record in your domain

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This tutorial requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create a resource group using az group create.

The following example creates a resource group named *myResourceGroupAG* in the *eastus* location.

```
az group create --name myResourceGroupAG --location eastus
```

Create network resources

Create the virtual network named *myVNet* and the subnet named *myAGSubnet* using az network vnet create. You can then add the subnet named *myBackendSubnet* that's needed by the backend pool of servers using az network vnet subnet create. Create the public IP address named *myAGPublicIPAddress* using az network public-ip create.

```
az network vnet create \
  --name myVNet \
  --resource-group myResourceGroupAG \
  --location eastus \
  --address-prefix 10.0.0.0/16 \
  --subnet-name myAGSubnet \
  --subnet-prefix 10.0.1.0/24
az network vnet subnet create \
  --name myBackendSubnet \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --address-prefix 10.0.2.0/24
az network public-ip create \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress
```

Create an application gateway

You can use az network application-gateway create to create the application gateway named *myAppGateway*. When you create an application gateway using the Azure CLI, you specify configuration information, such as capacity, sku, and HTTP settings. The application gateway is assigned to *myAGSubnet* and *myAGPublicIPAddress* that you previously created.

```
az network application-gateway create \
  --name myAppGateway \
  --location eastus \
  --resource-group myResourceGroupAG \
  --vnet-name myVNet \
  --subnet myAGsubnet \
  --capacity 2 \
  --sku Standard_Medium \
  --http-settings-cookie-based-affinity Disabled \
  --frontend-port 80 \
  --http-settings-port 80 \
  --http-settings-protocol Http \
  --public-ip-address myAGPublicIPAddress
```

It may take several minutes for the application gateway to be created. After the application gateway is created, you can see these new features of it:

- *appGatewayBackendPool* - An application gateway must have at least one backend address pool.
- *appGatewayBackendHttpSettings* - Specifies that port 80 and an HTTP protocol is used for communication.
- *appGatewayHttpListener* - The default listener associated with *appGatewayBackendPool*.
- *appGatewayFrontendIP* - Assigns *myAGPublicIPAddress* to *appGatewayHttpListener*.
- *rule1* - The default routing rule that is associated with *appGatewayHttpListener*.

Add listeners and rules

A listener is required to enable the application gateway to route traffic appropriately to the backend pool. In this tutorial, you create two listeners for your two domains. In this example, listeners are created for the domains of *www.contoso.com* and *www.contoso.org*.

Add the backend listeners that are needed to route traffic using az network application-gateway http-listener create.

```
az network application-gateway http-listener create \
  --name contosoComListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port appGatewayFrontendPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway \
  --host-name www.contoso.com
az network application-gateway http-listener create \
  --name contosoOrgListener \
  --frontend-ip appGatewayFrontendIP \
  --frontend-port appGatewayFrontendPort \
  --resource-group myResourceGroupAG \
  --gateway-name myAppGateway \
  --host-name www.contoso.org
```

Add the redirection configuration

Add the redirection configuration that sends traffic from *www.consoto.org* to the listener for *www.contoso.com* in the application gateway using az network application-gateway redirect-config create.

```
az network application-gateway redirect-config create \
  --name orgToCom \
  --gateway-name myAppGateway \
  --resource-group myResourceGroupAG \
  --type Permanent \
  --target-listener contosoListener \
  --include-path true \
  --include-query-string true
```

Add routing rules

Rules are processed in the order in which they are created, and traffic is directed using the first rule that matches the URL sent to the application gateway. For example, if you have a rule using a basic listener and a rule using a multi-site listener both on the same port, the rule with the multi-site listener must be listed before the rule with the basic listener in order for the multi-site rule to function as expected.

In this example, you create two new rules and delete the default rule that was created. You can add the rule using az network application-gateway rule create.

```
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name contosoComRule \
  --resource-group myResourceGroupAG \
  --http-listener contosoComListener \
  --rule-type Basic \
  --address-pool appGatewayBackendPool
az network application-gateway rule create \
  --gateway-name myAppGateway \
  --name contosoOrgRule \
  --resource-group myResourceGroupAG \
  --http-listener contosoOrgListener \
  --rule-type Basic \
  --redirect-config orgToCom
az network application-gateway rule delete \
  --gateway-name myAppGateway \
  --name rule1 \
  --resource-group myResourceGroupAG
```

Create virtual machine scale sets

In this example, you create a virtual machine scale set that supports the backend pool that you created. The scale set that you create is named *myvmss* and contains two virtual machine instances on which you install NGINX.

```
az vmss create \
  --name myvmss \
  --resource-group myResourceGroupAG \
  --image UbuntuLTS \
  --admin-username azureuser \
  --admin-password Azure123456! \
  --instance-count 2 \
  --vnet-name myVNet \
  --subnet myBackendSubnet \
  --vm-sku Standard_DS2 \
  --upgrade-policy-mode Automatic \
  --app-gateway myAppGateway \
  --backend-pool-name appGatewayBackendPool
```

Install NGINX

Run this command in the shell window:

```
az vmss extension set \
  --publisher Microsoft.Azure.Extensions \
  --version 2.0 \
  --name CustomScript \
  --resource-group myResourceGroupAG \
  --vmss-name myvmss \
  --settings '{ "fileUris": ["https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/install_nginx.sh"],
  "commandToExecute": "./install_nginx.sh" }'
```

Create CNAME record in your domain

After the application gateway is created with its public IP address, you can get the DNS address and use it to create a CNAME record in your domain. You can use az network public-ip show to get the DNS address of the application gateway. Copy the *fqdn* value of the DNSSettings and use it as the value of the CNAME record that you create. The use of A-records is not recommended because the VIP may change when the application gateway is restarted.

```
az network public-ip show \
  --resource-group myResourceGroupAG \
  --name myAGPublicIPAddress \
  --query [dnsSettings.fqdn] \
  --output tsv
```

Test the application gateway

Enter your domain name into the address bar of your browser. Such as, http://www.contoso.com.

Change the address to your other domain, for example http://www.contoso.org and you should see that the traffic has been redirected back to the listener for www.contoso.com.

Next steps

In this tutorial, you learned how to:

- Set up the network
- Create an application gateway
- Add listeners and redirection rule
- Create a virtual machine scale set with the backend pool
- Create a CNAME record in your domain

Create an application gateway with URL path-based redirection using Azure PowerShell

8/22/2022 • 9 minutes to read • Edit Online

You can use Azure PowerShell to configure URL-based routing rules when you create an application gateway. In this article, you create backend pools using virtual machine scale sets. You then create URL routing rules that make sure web traffic is redirected to the appropriate backend pool.

In this article, you learn how to:

- Set up the network
- Create an application gateway
- Add listeners and routing rules
- Create virtual machine scale sets for backend pools

The following example shows site traffic coming from both ports 8080 and 8081 and being directed to the same backend pools:

If you prefer, you can complete this procedure using Azure CLI.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this procedure requires the Azure PowerShell module version 1.0.0 or later. To find the version, run `Get-Module -ListAvailable Az` . If you need to upgrade, see Install Azure PowerShell module. If you are running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Create an Azure resource group using New-AzResourceGroup.

```
New-AzResourceGroup -Name myResourceGroupAG -Location eastus
```

Create network resources

Create the subnet configurations for *myBackendSubnet* and *myAGSubnet* using New-AzVirtualNetworkSubnetConfig. Create the virtual network named *myVNet* using New-AzVirtualNetwork with the subnet configurations. And finally, create the public IP address named *myAGPublicIPAddress* using New-AzPublicIpAddress. These resources are used to provide network connectivity to the application gateway and its associated resources.

```
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myBackendSubnet `
  -AddressPrefix 10.0.1.0/24

$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `
  -Name myAGSubnet `
  -AddressPrefix 10.0.2.0/24

New-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myVNet `
  -AddressPrefix 10.0.0.0/16 `
  -Subnet $backendSubnetConfig, $agSubnetConfig

New-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -Name myAGPublicIPAddress `
  -AllocationMethod Dynamic
```

Create an application gateway

In this section, you create resources that support the application gateway, and then finally create it. The resources that you create include:

- *IP configurations and frontend port* - Associates the subnet that you previously created to the application gateway and assigns a port to use to access it.
- *Default pool* - All application gateways must have at least one backend pool of servers.
- *Default listener and rule* - The default listener listens for traffic on the port that was assigned and the default rule sends traffic to the default pool.

Create the IP configurations and frontend port

Associate *myAGSubnet* that you previously created to the application gateway using New-AzApplicationGatewayIPConfiguration. Assign *myAGPublicIPAddress* to the application gateway using New-AzApplicationGatewayFrontendIPConfig. And then you can create the HTTP port using New-AzApplicationGatewayFrontendPort.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$subnet=$vnet.Subnets[0]

$pip = Get-AzPublicIpAddress `
  -ResourceGroupName myResourceGroupAG `
  -Name myAGPublicIPAddress

$gipconfig = New-AzApplicationGatewayIPConfiguration `
  -Name myAGIPConfig `
  -Subnet $subnet

$fipconfig = New-AzApplicationGatewayFrontendIPConfig `
  -Name myAGFrontendIPConfig `
  -PublicIPAddress $pip

$frontendport = New-AzApplicationGatewayFrontendPort `
  -Name myFrontendPort `
  -Port 80
```

Create the default pool and settings

Create the default backend pool named *appGatewayBackendPool* for the application gateway using New-AzApplicationGatewayBackendAddressPool. Configure the settings for the backend pool using New-AzApplicationGatewayBackendHttpSettings.

```
$defaultPool = New-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool

$poolSettings = New-AzApplicationGatewayBackendHttpSettings `
  -Name myPoolSettings `
  -Port 80 `
  -Protocol Http `
  -CookieBasedAffinity Enabled `
  -RequestTimeout 120
```

Create the default listener and rule

A listener is required to enable the application gateway to route traffic appropriately to a backend pool. In this article, you create multiple listeners. The first basic listener expects traffic at the root URL. The other listeners expect traffic at specific URLs, such as `http://52.168.55.24:8080/images/` or `http://52.168.55.24:8081/video/`.

Create a listener named *defaultListener* using New-AzApplicationGatewayHttpListener with the frontend configuration and frontend port that you previously created. A rule is required for the listener to know which backend pool to use for incoming traffic. Create a basic rule named *rule1* using New-AzApplicationGatewayRequestRoutingRule.

```
$defaultlistener = New-AzApplicationGatewayHttpListener `
  -Name defaultListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $frontendport

$frontendRule = New-AzApplicationGatewayRequestRoutingRule `
  -Name rule1 `
  -RuleType Basic `
  -HttpListener $defaultlistener `
  -BackendAddressPool $defaultPool `
  -BackendHttpSettings $poolSettings
```

Create the application gateway

Now that you created the necessary supporting resources, specify parameters for the application gateway named *myAppGateway* using New-AzApplicationGatewaySku, and then create it using New-AzApplicationGateway.

```
$sku = New-AzApplicationGatewaySku `
  -Name Standard_Medium `
  -Tier Standard `
  -Capacity 2

New-AzApplicationGateway `
  -Name myAppGateway `
  -ResourceGroupName myResourceGroupAG `
  -Location eastus `
  -BackendAddressPools $defaultPool `
  -BackendHttpSettingsCollection $poolSettings `
  -FrontendIpConfigurations $fipconfig `
  -GatewayIpConfigurations $gipconfig `
  -FrontendPorts $frontendport `
  -HttpListeners $defaultlistener `
  -RequestRoutingRules $frontendRule `
  -Sku $sku
```

Add backend pools and ports

You can add backend pools to your application gateway by using Add-AzApplicationGatewayBackendAddressPool. In this example, *imagesBackendPool* and *videoBackendPool* are created. You add the frontend port for the pools using Add-AzApplicationGatewayFrontendPort. You then submit the changes to the application gateway using Set-AzApplicationGateway.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

Add-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name imagesBackendPool

Add-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name videoBackendPool

Add-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name bport `
  -Port 8080

Add-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name rport `
  -Port 8081

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add listeners and rules

Add listeners

Add the listeners named *backendListener* and *redirectedListener* that are needed to route traffic using Add-AzApplicationGatewayHttpListener.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPort = Get-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name bport

$redirectPort = Get-AzApplicationGatewayFrontendPort `
  -ApplicationGateway $appgw `
  -Name rport

$fipconfig = Get-AzApplicationGatewayFrontendIPConfig `
  -ApplicationGateway $appgw

Add-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name backendListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $backendPort

Add-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name redirectedListener `
  -Protocol Http `
  -FrontendIPConfiguration $fipconfig `
  -FrontendPort $redirectPort

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add the default URL path map

URL path maps make sure that specific URLs are routed to specific backend pools. You can create the URL path maps named *imagePathRule* and *videoPathRule* using New-AzApplicationGatewayPathRuleConfig and Add-AzApplicationGatewayUrlPathMapConfig.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$poolSettings = Get-AzApplicationGatewayBackendHttpSettings `
  -ApplicationGateway $appgw `
  -Name myPoolSettings

$imagePool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name imagesBackendPool

$videoPool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name videoBackendPool

$defaultPool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name appGatewayBackendPool

$imagePathRule = New-AzApplicationGatewayPathRuleConfig `
  -Name imagePathRule `
  -Paths "/images/*" `
  -BackendAddressPool $imagePool `
  -BackendHttpSettings $poolSettings

$videoPathRule = New-AzApplicationGatewayPathRuleConfig `
  -Name videoPathRule `
  -Paths "/video/*" `
  -BackendAddressPool $videoPool `
  -BackendHttpSettings $poolSettings

Add-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name urlpathmap `
  -PathRules $imagePathRule, $videoPathRule `
  -DefaultBackendAddressPool $defaultPool `
  -DefaultBackendHttpSettings $poolSettings

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add redirection configuration

You can configure redirection for the listener using Add-AzApplicationGatewayRedirectConfiguration.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendListener = Get-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name backendListener

$redirectConfig = Add-AzApplicationGatewayRedirectConfiguration `
  -ApplicationGateway $appgw `
  -Name redirectConfig `
  -RedirectType Found `
  -TargetListener $backendListener `
  -IncludePath $true `
  -IncludeQueryString $true

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add the redirection URL path map

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$poolSettings = Get-AzApplicationGatewayBackendHttpSettings `
  -ApplicationGateway $appgw `
  -Name myPoolSettings

$defaultPool = Get-AzApplicationGatewayBackendAddressPool `
  -ApplicationGateway $appgw `
  -Name appGatewayBackendPool

$redirectConfig = Get-AzApplicationGatewayRedirectConfiguration `
  -ApplicationGateway $appgw `
  -Name redirectConfig

$redirectPathRule = New-AzApplicationGatewayPathRuleConfig `
  -Name redirectPathRule `
  -Paths "/images/*" `
  -RedirectConfiguration $redirectConfig

Add-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name redirectpathmap `
  -PathRules $redirectPathRule `
  -DefaultBackendAddressPool $defaultPool `
  -DefaultBackendHttpSettings $poolSettings

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Add routing rules

The routing rules associate the URL maps with the listeners that you created. You can add the rules named *defaultRule* and *redirectedRule* using Add-AzApplicationGatewayRequestRoutingRule.

```
$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendlistener = Get-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name backendListener

$redirectlistener = Get-AzApplicationGatewayHttpListener `
  -ApplicationGateway $appgw `
  -Name redirectedListener

$urlPathMap = Get-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name urlpathmap

$redirectPathMap = Get-AzApplicationGatewayUrlPathMapConfig `
  -ApplicationGateway $appgw `
  -Name redirectpathmap

Add-AzApplicationGatewayRequestRoutingRule `
  -ApplicationGateway $appgw `
  -Name defaultRule `
  -RuleType PathBasedRouting `
  -HttpListener $backendlistener `
  -UrlPathMap $urlPathMap

Add-AzApplicationGatewayRequestRoutingRule `
  -ApplicationGateway $appgw `
  -Name redirectedRule `
  -RuleType PathBasedRouting `
  -HttpListener $redirectlistener `
  -UrlPathMap $redirectPathMap

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Create virtual machine scale sets

In this example, you create three virtual machine scale sets that support the three backend pools that you created. The scale sets that you create are named *myvmss1*, *myvmss2*, and *myvmss3*. Each scale set contains two virtual machine instances on which you install IIS. You assign the scale set to the backend pool when you configure the IP settings.

Replace <username> and <password> with your own values before you run the script.

```
$vnet = Get-AzVirtualNetwork `
  -ResourceGroupName myResourceGroupAG `
  -Name myVNet

$appgw = Get-AzApplicationGateway `
  -ResourceGroupName myResourceGroupAG `
  -Name myAppGateway

$backendPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name appGatewayBackendPool `
  -ApplicationGateway $appgw

$imagesPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name imagesBackendPool `
  -ApplicationGateway $appgw

$videoPool = Get-AzApplicationGatewayBackendAddressPool `
  -Name videoBackendPool `
  -ApplicationGateway $appgw
```

```
for ($i=1; $i -le 3; $i++)
{
  if ($i -eq 1)
  {
     $poolId = $backendPool.Id
  }
  if ($i -eq 2)
  {
    $poolId = $imagesPool.Id
  }
  if ($i -eq 3)
  {
    $poolId = $videoPool.Id
  }

  $ipConfig = New-AzVmssIpConfig `
    -Name myVmssIPConfig$i `
    -SubnetId $vnet.Subnets[1].Id `
    -ApplicationGatewayBackendAddressPoolsId $poolId

  $vmssConfig = New-AzVmssConfig `
    -Location eastus `
    -SkuCapacity 2 `
    -SkuName Standard_DS2 `
    -UpgradePolicyMode Automatic

  Set-AzVmssStorageProfile $vmssConfig `
    -ImageReferencePublisher MicrosoftWindowsServer `
    -ImageReferenceOffer WindowsServer `
    -ImageReferenceSku 2016-Datacenter `
    -ImageReferenceVersion latest `
    -OsDiskCreateOption FromImage

  Set-AzVmssOsProfile $vmssConfig `
    -AdminUsername <username> `
    -AdminPassword "<password>" `
    -ComputerNamePrefix myvmss$i

  Add-AzVmssNetworkInterfaceConfiguration `
    -VirtualMachineScaleSet $vmssConfig `
    -Name myVmssNetConfig$i `
    -Primary $true `
    -IPConfiguration $ipConfig

  New-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmssConfig
}
```

Install IIS

```
$publicSettings = @{ "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-
samples/master/application-gateway/iis/appgatewayurl.ps1");
  "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }

for ($i=1; $i -le 3; $i++)
{
  $vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG -VMScaleSetName myvmss$i

  Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
    -Name "customScript" `
    -Publisher "Microsoft.Compute" `
    -Type "CustomScriptExtension" `
    -TypeHandlerVersion 1.8 `
    -Setting $publicSettings

  Update-AzVmss `
    -ResourceGroupName myResourceGroupAG `
    -Name myvmss$i `
    -VirtualMachineScaleSet $vmss
}
```

Test the application gateway

You can use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://52.168.55.24`, `http://52.168.55.24:8080/images/test.htm`, `http://52.168.55.24:8080/video/test.htm`, or `http://52.168.55.24:8081/images/test.htm`.

```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

Change the URL to http://<ip-address>:8080/images/test.htm, substituting your IP address for <ip-address>, and you should see something like the following example:

Change the URL to http://<ip-address>:8080/video/test.htm, substituting your IP address for <ip-address>, and you should see something like the following example:

Now, change the URL to http://<ip-address>:8081/images/test.htm, substituting your IP address for <ip-address>, and you should see traffic redirected back to the images backend pool at http://<ip-address>:8080/images.

Clean up resources

When no longer needed, remove the resource group, application gateway, and all related resources using Remove-AzResourceGroup.

```
Remove-AzResourceGroup -Name myResourceGroupAG
```

Next steps

Learn more about what you can do with application gateway

Rewrite HTTP request and response headers with Azure Application Gateway - Azure portal

8/22/2022 • 3 minutes to read • Edit Online

This article describes how to use the Azure portal to configure an Application Gateway v2 SKU instance to rewrite the HTTP headers in requests and responses.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

You need to have an Application Gateway v2 SKU instance to complete the steps in this article. Rewriting headers isn't supported in the v1 SKU. If you don't have the v2 SKU, create an Application Gateway v2 SKU instance before you begin.

Create required objects

To configure HTTP header rewrite, you need to complete these steps.

1. Create the objects that are required for HTTP header rewrite:

   - **Rewrite action**: Used to specify the request and request header fields that you intend to rewrite and the new value for the headers. You can associate one or more rewrite conditions with a rewrite action.
   - **Rewrite condition**: An optional configuration. Rewrite conditions evaluate the content of HTTP(S) requests and responses. The rewrite action will occur if the HTTP(S) request or response matches the rewrite condition.

     If you associate more than one condition with an action, the action occurs only when all the conditions are met. In other words, the operation is a logical AND operation.
   - **Rewrite rule**: Contains multiple rewrite action / rewrite condition combinations.
   - **Rule sequence**: Helps determine the order in which the rewrite rules execute. This configuration is helpful when you have multiple rewrite rules in a rewrite set. A rewrite rule that has a lower rule sequence value runs first. If you assign the same rule sequence value to two rewrite rules, the order of execution is non-deterministic.
   - **Rewrite set**: Contains multiple rewrite rules that will be associated with a request routing rule.

2. Attach the rewrite set to a routing rule. The rewrite configuration is attached to the source listener via the routing rule. When you use a basic routing rule, the header rewrite configuration is associated with a source listener and is a global header rewrite. When you use a path-based routing rule, the header rewrite configuration is defined on the URL path map. In that case, it applies only to the specific path area of a site.

You can create multiple HTTP header rewrite sets and apply each rewrite set to multiple listeners. But you can apply only one rewrite set to a specific listener.

Sign in to Azure

Sign in to the Azure portal with your Azure account.

Configure header rewrite

In this example, we'll modify a redirection URL by rewriting the location header in the HTTP response sent by a back-end application.

1. Select **All resources**, and then select your application gateway.
2. Select **Rewrites** in the left pane.
3. Select **Rewrite set**:

4. Provide a name for the rewrite set and associate it with a routing rule:
   - Enter the name for the rewrite set in the **Name** box.
   - Select one or more of the rules listed in the **Associated routing rules** list. You can select only rules that haven't been associated with other rewrite sets. The rules that have already been associated with other rewrite sets are dimmed.
   - Select **Next**.

5. Create a rewrite rule:

- Select **Add rewrite rule**.

- Enter a name for the rewrite rule in the **Rewrite rule name** box. Enter a number in the **Rule sequence** box.

6. In this example, we'll rewrite the location header only when it contains a reference to azurewebsites.net. To do this, add a condition to evaluate whether the location header in the response contains azurewebsites.net:

   - Select **Add condition** and then select the box containing the **If** instructions to expand it.

   - In the **Type of variable to check** list, select **HTTP header**.
   - In the **Header type** list, select **Response**.
   - Because in this example we're evaluating the location header, which is a common header, select **Common header** under **Header name**.
   - In the **Common header** list, select **Location**.
   - Under **Case-sensitive**, select **No**.
   - In the **Operator** list, select **equal (=)**.
   - Enter a regular expression pattern. In this example, we'll use the pattern `(https?):\/\/.*azurewebsites.net(.*)$`.
   - Select **OK**.

7. Add an action to rewrite the location header:

   - In the **Action type** list, select **Set**.
   - In the **Header type** list, select **Response**.
   - Under **Header name**, select **Common header**.
   - In the **Common header** list, select **Location**.
   - Enter the header value. In this example, we'll use `{http_resp_Location_1}://contoso.com{http_resp_Location_2}` as the header value. This value will replace *azurewebsites.net* with *contoso.com* in the location header.
   - Select **OK**.

8. Select **Create** to create the rewrite set:

9. The Rewrite set view will open. Verify that the rewrite set you created is in the list of rewrite sets:

Next steps

To learn more about how to set up some common use cases, see common header rewrite scenarios.

Rewrite HTTP request and response headers with Azure Application Gateway - Azure PowerShell

8/22/2022 • 3 minutes to read • Edit Online

This article describes how to use Azure PowerShell to configure an Application Gateway v2 SKU instance to rewrite the HTTP headers in requests and responses.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

- You need to run Azure PowerShell locally to complete the steps in this article. You also need to have Az module version 1.0.0 or later installed. Run `Import-Module Az` and then `Get-Module Az` to determine the version that you have installed. If you need to upgrade, see Install Azure PowerShell module. After you verify the PowerShell version, run `Login-AzAccount` to create a connection with Azure.
- You need to have an Application Gateway v2 SKU instance. Rewriting headers isn't supported in the v1 SKU. If you don't have the v2 SKU, create an Application Gateway v2 SKU instance before you begin.

Create required objects

To configure HTTP header rewrite, you need to complete these steps.

1. Create the objects that are required for HTTP header rewrite:

   - **RequestHeaderConfiguration**: Used to specify the request header fields that you intend to rewrite and the new value for the headers.

   - **ResponseHeaderConfiguration**: Used to specify the response header fields that you intend to rewrite and the new value for the headers.

   - **ActionSet**: Contains the configurations of the request and response headers specified previously.

   - **Condition**: An optional configuration. Rewrite conditions evaluate the content of HTTP(S) requests and responses. The rewrite action will occur if the HTTP(S) request or response matches the rewrite condition.

     If you associate more than one condition with an action, the action occurs only when all the conditions are met. In other words, the operation is a logical AND operation.

   - **RewriteRule**: Contains multiple rewrite action / rewrite condition combinations.

   - **RuleSequence**: An optional configuration that helps determine the order in which rewrite rules execute. This configuration is helpful when you have multiple rewrite rules in a rewrite set. A rewrite rule that has a lower rule sequence value runs first. If you assign the same rule sequence value to two rewrite rules, the order of execution is non-deterministic.

     If you don't specify the RuleSequence explicitly, a default value of 100 is set.

   - **RewriteRuleSet**: Contains multiple rewrite rules that will be associated to a request routing rule.

2. Attach the RewriteRuleSet to a routing rule. The rewrite configuration is attached to the source listener via the routing rule. When you use a basic routing rule, the header rewrite configuration is associated with a source listener and is a global header rewrite. When you use a path-based routing rule, the header

rewrite configuration is defined on the URL path map. In that case, it applies only to the specific path area of a site.

You can create multiple HTTP header rewrite sets and apply each rewrite set to multiple listeners. But you can apply only one rewrite set to a specific listener.

Sign in to Azure

```
Connect-AzAccount
Select-AzSubscription -Subscription "<sub name>"
```

Specify the HTTP header rewrite rule configuration

In this example, we'll modify a redirection URL by rewriting the location header in the HTTP response whenever the location header contains a reference to azurewebsites.net. To do this, we'll add a condition to evaluate whether the location header in the response contains azurewebsites.net. We'll use the pattern `(https?)://.*azurewebsites.net(.*)$`. And we'll use `{http_resp_Location_1}://contoso.com{http_resp_Location_2}` as the header value. This value will replace *azurewebsites.net* with *contoso.com* in the location header.

```
$responseHeaderConfiguration = New-AzApplicationGatewayRewriteRuleHeaderConfiguration -HeaderName "Location"
-HeaderValue "{http_resp_Location_1}://contoso.com{http_resp_Location_2}"
$actionSet = New-AzApplicationGatewayRewriteRuleActionSet -ResponseHeaderConfiguration
$responseHeaderConfiguration
$condition = New-AzApplicationGatewayRewriteRuleCondition -Variable "http_resp_Location" -Pattern "
(https?):\/\/.*azurewebsites\.net(.*)$" -IgnoreCase
$rewriteRule = New-AzApplicationGatewayRewriteRule -Name LocationHeader -ActionSet $actionSet -Condition
$condition
$rewriteRuleSet = New-AzApplicationGatewayRewriteRuleSet -Name LocationHeaderRewrite -RewriteRule
$rewriteRule
```

Retrieve the configuration of your application gateway

```
$appgw = Get-AzApplicationGateway -Name "AutoscalingAppGw" -ResourceGroupName "<rg name>"
```

Retrieve the configuration of your request routing rule

```
$reqRoutingRule = Get-AzApplicationGatewayRequestRoutingRule -Name rule1 -ApplicationGateway $appgw
```

Update the application gateway with the configuration for rewriting HTTP headers

In this example, the rewrite set would be associated instantly against a basic routing rule. In case of a path based routing rule, the association would not be enabled by default. The rewrite set can be enabled either via checking the paths on which it needs to be applied via portal or by providing a URL path map config specifying the RewriteRuleSet against each path option.

```
Add-AzApplicationGatewayRewriteRuleSet -ApplicationGateway $appgw -Name $rewriteRuleSet.Name  -RewriteRule
$rewriteRuleSet.RewriteRules
Set-AzApplicationGatewayRequestRoutingRule -ApplicationGateway $appgw -Name $reqRoutingRule.Name -RuleType
$reqRoutingRule.RuleType -BackendHttpSettingsId $reqRoutingRule.BackendHttpSettings.Id -HttpListenerId
$reqRoutingRule.HttpListener.Id -BackendAddressPoolId $reqRoutingRule.BackendAddressPool.Id -
RewriteRuleSetId $rewriteRuleSet.Id
Set-AzApplicationGateway -ApplicationGateway $appgw
```

Delete a rewrite rule

```
$appgw = Get-AzApplicationGateway -Name "AutoscalingAppGw" -ResourceGroupName "<rg name>"
Remove-AzApplicationGatewayRewriteRuleSet -Name "LocationHeaderRewrite" -ApplicationGateway $appgw
$requestroutingrule= Get-AzApplicationGatewayRequestRoutingRule -Name "rule1" -ApplicationGateway $appgw
$requestroutingrule.RewriteRuleSet= $null
set-AzApplicationGateway -ApplicationGateway $appgw
```

Next steps

To learn more about how to set up some common use cases, see common header rewrite scenarios.

Create an application gateway and rewrite HTTP headers

8/22/2022 • 4 minutes to read • Edit Online

You can use Azure PowerShell to configure rules to rewrite HTTP request and response headers when you create the new autoscaling and zone-redundant application gateway SKU

In this article, you learn how to:

- Create an autoscale virtual network
- Create a reserved public IP
- Set up your application gateway infrastructure
- Specify your http header rewrite rule configuration
- Specify autoscale
- Create the application gateway
- Test the application gateway

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

This article requires that you run Azure PowerShell locally. You must have Az module version 1.0.0 or later installed. Run `Import-Module Az` and then `Get-Module Az` to find the version. If you need to upgrade, see Install Azure PowerShell module. After you verify the PowerShell version, run `Login-AzAccount` to create a connection with Azure.

Sign in to Azure

```
Connect-AzAccount
Select-AzSubscription -Subscription "<sub name>"
```

Create a resource group

Create a resource group in one of the available locations.

```
$location = "East US 2"
$rg = "<rg name>"

#Create a new Resource Group
New-AzResourceGroup -Name $rg -Location $location
```

Create a virtual network

Create a virtual network with one dedicated subnet for an autoscaling application gateway. Currently only one autoscaling application gateway can be deployed in each dedicated subnet.

```
#Create VNet with two subnets
$sub1 = New-AzVirtualNetworkSubnetConfig -Name "AppGwSubnet" -AddressPrefix "10.0.0.0/24"
$sub2 = New-AzVirtualNetworkSubnetConfig -Name "BackendSubnet" -AddressPrefix "10.0.1.0/24"
$vnet = New-AzvirtualNetwork -Name "AutoscaleVNet" -ResourceGroupName $rg `
       -Location $location -AddressPrefix "10.0.0.0/16" -Subnet $sub1, $sub2
```

Create a reserved public IP

Specify the allocation method of PublicIPAddress as **Static**. An autoscaling application gateway VIP can only be static. Dynamic IPs are not supported. Only the standard PublicIpAddress SKU is supported.

```
#Create static public IP
$pip = New-AzPublicIpAddress -ResourceGroupName $rg -name "AppGwVIP" `
       -location $location -AllocationMethod Static -Sku Standard
```

Retrieve details

Retrieve details of the resource group, subnet, and IP in a local object to create the IP configuration details for the application gateway.

```
$resourceGroup = Get-AzResourceGroup -Name $rg
$publicip = Get-AzPublicIpAddress -ResourceGroupName $rg -name "AppGwVIP"
$vnet = Get-AzvirtualNetwork -Name "AutoscaleVNet" -ResourceGroupName $rg
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name "AppGwSubnet" -VirtualNetwork $vnet
```

Configure the infrastructure

Configure the IP config, front-end IP config, back-end pool, HTTP settings, certificate, port, and listener in an identical format to the existing Standard application gateway. The new SKU follows the same object model as the Standard SKU.

```
$ipconfig = New-AzApplicationGatewayIPConfiguration -Name "IPConfig" -Subnet $gwSubnet
$fip = New-AzApplicationGatewayFrontendIPConfig -Name "FrontendIPCOnfig" -PublicIPAddress $publicip
$pool = New-AzApplicationGatewayBackendAddressPool -Name "Pool1" `
       -BackendIPAddresses testbackend1.westus.cloudapp.azure.com, testbackend2.westus.cloudapp.azure.com
$fp01 = New-AzApplicationGatewayFrontendPort -Name "HTTPPort" -Port 80

$listener01 = New-AzApplicationGatewayHttpListener -Name "HTTPListener" `
             -Protocol Http -FrontendIPConfiguration $fip -FrontendPort $fp01

$setting = New-AzApplicationGatewayBackendHttpSettings -Name "BackendHttpSetting1" `
          -Port 80 -Protocol Http -CookieBasedAffinity Disabled
```

Specify your HTTP header rewrite rule configuration

Configure the new objects required to rewrite the http headers:

- **RequestHeaderConfiguration**: this object is used to specify the request header fields that you intend to rewrite and the new value that the original headers need to be rewritten to.
- **ResponseHeaderConfiguration**: this object is used to specify the response header fields that you intend to rewrite and the new value that the original headers need to be rewritten to.
- **ActionSet**: this object contains the configurations of the request and response headers specified above.

- **RewriteRule**: this object contains all the *actionSets* specified above.
- **RewriteRuleSet**- this object contains all the *rewriteRules* and will need to be attached to a request routing rule - basic or path-based.

```
$requestHeaderConfiguration = New-AzApplicationGatewayRewriteRuleHeaderConfiguration -HeaderName "X-isThroughProxy" -HeaderValue "True"
$responseHeaderConfiguration = New-AzApplicationGatewayRewriteRuleHeaderConfiguration -HeaderName "Strict-Transport-Security" -HeaderValue "max-age=31536000"
$actionSet = New-AzApplicationGatewayRewriteRuleActionSet -RequestHeaderConfiguration $requestHeaderConfiguration -ResponseHeaderConfiguration $responseHeaderConfiguration
$rewriteRule = New-AzApplicationGatewayRewriteRule -Name rewriteRule1 -ActionSet $actionSet
$rewriteRuleSet = New-AzApplicationGatewayRewriteRuleSet -Name rewriteRuleSet1 -RewriteRule $rewriteRule
```

Specify the routing rule

Create a request routing rule. Once created, this rewrite configuration is attached to the source listener via the routing rule. When using a basic routing rule, the header rewrite configuration is associated with a source listener and is a global header rewrite. When a path-based routing rule is used, the header rewrite configuration is defined on the URL path map. So, it only applies to the specific path area of a site. Below, a basic routing rule is created and the rewrite rule set is attached.

```
$rule01 = New-AzApplicationGatewayRequestRoutingRule -Name "Rule1" -RuleType basic `
         -BackendHttpSettings $setting -HttpListener $listener01 -BackendAddressPool $pool -RewriteRuleSet $rewriteRuleSet
```

Specify autoscale

Now you can specify the autoscale configuration for the application gateway. Two autoscaling configuration types are supported:

- **Fixed capacity mode**. In this mode, the application gateway does not autoscale and operates at a fixed Scale Unit capacity.

```
$sku = New-AzApplicationGatewaySku -Name Standard_v2 -Tier Standard_v2 -Capacity 2
```

- **Autoscaling mode**. In this mode, the application gateway autoscales based on the application traffic pattern.

```
$autoscaleConfig = New-AzApplicationGatewayAutoscaleConfiguration -MinCapacity 2
$sku = New-AzApplicationGatewaySku -Name Standard_v2 -Tier Standard_v2
```

Create the application gateway

Create the application gateway and include redundancy zones and the autoscale configuration.

```
$appgw = New-AzApplicationGateway -Name "AutoscalingAppGw" -Zone 1,2,3 -ResourceGroupName $rg -Location $location -BackendAddressPools $pool -BackendHttpSettingsCollection $setting -GatewayIpConfigurations $ipconfig -FrontendIpConfigurations $fip -FrontendPorts $fp01 -HttpListeners $listener01 -RequestRoutingRules $rule01 -Sku $sku -AutoscaleConfiguration $autoscaleConfig -RewriteRuleSet $rewriteRuleSet
```

Test the application gateway

Use Get-AzPublicIPAddress to get the public IP address of the application gateway. Copy the public IP address or DNS name, and then paste it into the address bar of your browser.

```
Get-AzPublicIPAddress -ResourceGroupName $rg -Name AppGwVIP
```

Clean up resources

First explore the resources that were created with the application gateway. Then, when they're no longer needed, you can use the `Remove-AzResourceGroup` command to remove the resource group, application gateway, and all related resources.

`Remove-AzResourceGroup -Name $rg`

Next steps

- Create an application gateway with URL path-based routing rules

Rewrite URL with Azure Application Gateway - Azure portal

8/22/2022 • 3 minutes to read • Edit Online

This article describes how to use the Azure portal to configure an Application Gateway v2 SKU instance to rewrite URL.

> **NOTE**
>
> URL rewrite feature is available only for Standard_v2 and WAF_v2 SKU of Application Gateway. When URL rewrite is configured on a WAF enabled gateway, WAF evaluation will take place on the rewritten request headers and URL. Learn more.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

You need to have an Application Gateway v2 SKU instance to complete the steps in this article. Rewriting URL isn't supported in the v1 SKU. If you don't have the v2 SKU, create an Application Gateway v2 SKU instance before you begin.

Sign in to Azure

Sign in to the Azure portal with your Azure account.

Configure URL rewrite

In the below example whenever the request URL contains */article*, the URL path and URL query string are rewritten

`contoso.com/article/123/fabrikam` -> `contoso.com/article.aspx?id=123&title=fabrikam`

1. Select **All resources**, and then select your application gateway.
2. Select **Rewrites** in the left pane.
3. Select **Rewrite set**:

4. Provide a name for the rewrite set and associate it with a routing rule:

   a. Enter the name for the rewrite set in the **Name** box.

   b. Select one or more of the rules listed in the **Associated routing rules** list. This is used to associate the rewrite configuration to the source listener via the routing rule. You can select only those routing rules that haven't been associated with other rewrite sets. The rules that have already been associated with other rewrite sets are greyed out.

   c. Select **Next**.

5. Create a rewrite rule:

   a. Select **Add rewrite rule**.

   b. Enter a name for the rewrite rule in the **Rewrite rule name** box. Enter a number in the **Rule sequence** box.

6. In this example, we'll rewrite URL path and URL query string only when path contains */article*. To do this, add a condition to evaluate whether the URL path contains */article*

   a. Select **Add condition** and then select the box containing the **If** instructions to expand it.

   b. Since in this example we want to check the pattern */article* in the URL path, in the **Type of variable to check** list, select **Server variable**.

   c. In the **Server variable** list, select uri_path

   d. Under **Case-sensitive**, select **No**.

   e. In the **Operator** list, select **equal (=)**.

   f. Enter a regular expression pattern. In this example, we'll use the pattern `.*article/(.*)/(.*)`

   ( ) is used to capture the substring for later use in composing the expression for rewriting the URL path.

For more information, see here.

g. Select **OK**.

7. Add an action to rewrite the URL and URL path

   a. In the **Rewrite type** list, select **URL**.

   b. In the **Action type** list, select **Set**.

   c. Under **Components**, select **Both URL path and URL query string**

   d. In the **URL path value**, enter the new value of the path. In this example, we will use **/article.aspx**

   e. In the **URL query string value**, enter the new value of the URL query string. In this example, we will use **id={var_uri_path_1}&title={var_uri_path_2}**

   `{var_uri_path_1}` and `{var_uri_path_2}` are used to fetch the substrings captured while evaluating the condition in this expression `.*article/(.*)/(.*)`

   f. Select **OK**.

8. Click **Create** to create the rewrite set.

9. Verify that the new rewrite set appears in the list of rewrite sets

Verify URL rewrite through access logs

Observe the below fields in access logs to verify if the URL rewrite happened as per your expectation.

- **originalRequestUriWithArgs**: This field contains the original request URL
- **requestUri**: This field contains the URL after the rewrite operation on Application Gateway

For more information on all the fields in the access logs, see here.

Next steps

To learn more about how to set up rewrites for some common use cases, see common rewrite scenarios.

Configure App Service with Application Gateway

8/22/2022 • 15 minutes to read • Edit Online

Application gateway allows you to have an App Service app or other multi-tenant service as a back-end pool member. In this article, you learn to configure an App Service app with Application Gateway. The configuration for Application Gateway will differ depending on how App Service will be accessed:

- The first option makes use of a **custom domain** on both Application Gateway and the App Service in the backend.
- The second option is to have Application Gateway access App Service using its **default domain**, suffixed as ".azurewebsites.net".

- Custom domain (recommended)
- Default domain

This configuration is recommended for production-grade scenarios and meets the practice of not changing the host name in the request flow. You are required to have a custom domain (and associated certificate) available to avoid having to rely on the default ".azurewebsites" domain.

By associating the same domain name to both Application Gateway and App Service in the backend pool, the request flow does not need to override the host name. The backend web application will see the original host as was used by the client.

In this article you'll learn how to:

- Configure DNS
- Add App Service as backend pool to the Application Gateway
- Configure HTTP Settings for the connection to App Service
- Configure an HTTP Listener
- Configure a Request Routing Rule

Prerequisites

- Custom domain (recommended)
- Default domain

- Application Gateway: Create an application gateway without a backend pool target. For more information, see Quickstart: Direct web traffic with Azure Application Gateway - Azure portal

- App Service: If you don't have an existing App Service, see App Service documentation.
- A custom domain name and associated certificate (signed by a well known authority), stored in Key Vault. For more information on how to store certificates in Key Vault, see Tutorial: Import a certificate in Azure Key Vault

Configuring DNS

In the context of this scenario, DNS is relevant in two places:

- The DNS name, which the user or client is using towards Application Gateway and what is shown in a browser
- The DNS name, which Application Gateway is internally using to access the App Service in the backend

- Custom domain (recommended)
- Default domain

Route the user or client to Application Gateway using the custom domain. Set up DNS using a CNAME alias pointed to the DNS for Application Gateway. The Application Gateway DNS address is shown on the overview page of the associated Public IP address. Alternatively create an A record pointing to the IP address directly. (For Application Gateway V1 the VIP can change if you stop and start the service, which makes this option undesired.)

App Service should be configured so it accepts traffic from Application Gateway using the custom domain name as the incoming host. For more information on how to map a custom domain to the App Service, see Tutorial: Map an existing custom DNS name to Azure App Service To verify the domain, App Service only requires adding a TXT record. No change is required on CNAME or A-records. The DNS configuration for the custom domain will remain directed towards Application Gateway.

To accept connections to App Service over HTTPS, configure its TLS binding. For more information, see Secure a custom DNS name with a TLS/SSL binding in Azure App Service Configure App Service to pull the certificate for the custom domain from Azure Key Vault.

Add App service as backend pool

- Azure portal
- PowerShell

1. In the Azure portal, select your Application Gateway.
2. Under **Backend pools**, select the backend pool.
3. Under **Target type**, select **App Services**.
4. Under **Target** select your App Service.

> **NOTE**
>
> The dropdown only populates those app services which are in the same subscription as your Application Gateway. If you want to use an app service which is in a different subscription than the one in which the Application Gateway is, then instead of choosing **App Services** in the **Targets** dropdown, choose **IP address or hostname** option and enter the hostname (example.azurewebsites.net) of the app service.

5. Select **Save**.

Edit HTTP settings for App Service

- Azure portal
- PowerShell

An HTTP Setting is required that instructs Application Gateway to access the App Service backend using the **custom domain name**. The HTTP Setting will by default use the default health probe which relies on the hostname as is configured in the Backend Pool (suffixed "azurewebsites.net"). For this reason, it is good to first configure a custom health probe that is configured with the correct custom domain name as its host name.

We will connect to the backend using HTTPS.

1. Under **HTTP Settings**, select an existing HTTP setting or add a new one.
2. When creating a new HTTP Setting, give it a name
3. Select HTTPS as the desired backend protocol using port 443
4. If the certificate is signed by a well known authority, select "Yes" for "User well known CA certificate". Alternatively Add authentication/trusted root certificates of back-end servers
5. Make sure to set "Override with new host name" to "No"
6. Select the custom HTTPS health probe in the dropdown for "Custom probe".

> **NOTE**
>
> It will work with the default probe but for correctness we recommend using a custom probe with the correct domain name.)

Add HTTP setting

HTTP settings name

http-setting-to-app-service

Backend protocol

○ HTTP ◉ HTTPS

Backend port *

443

Trusted root certificate

For end-to-end SSL encryption, the backends must be in the allowlist of the application gateway. Upload the public certificate of the backend servers to this HTTP setting.

Use well known CA certificate

◉ Yes ○ No

Additional settings

Cookie-based affinity ⓘ

○ Enable ◉ Disable

Connection draining ⓘ

○ Enable ◉ Disable

Request time-out (seconds) * ⓘ

20

Override backend path ⓘ

Host name

By default, Application Gateway does not change the incoming HTTP host header from the client and sends the header unaltered to the backend. Multi-tenant services like App service or API management rely on a specific host header or SNI extension to resolve to the correct endpoint. Change these settings to overwrite the incoming HTTP host header.

Override with new host name

Yes **No**

Host name override

○ Pick host name from backend target

◉ Override with specific domain name

e.g. contoso.com

Use custom probe ⓘ

◉ Yes ○ No

Custom probe *

probe-for-appvs-backend-https

Configure an HTTP listener

To accept traffic we need to configure a Listener. For more info on this see Application Gateway listener

configuration.

- Azure portal
- PowerShell

1. Open the "Listeners" section and choose "Add listener" or click an existing one to edit
2. For a new listener: give it a name
3. Under "Frontend IP", select the IP address to listen on
4. Under "Port", select 443
5. Under "Protocol", select "HTTPS"
6. Under "Choose a certificate", select "Choose a certificate from Key Vault". For more information, see Using Key Vault where you find more information on how to assign a managed identity and provide it with rights to your Key Vault.
   a. Give the certificate a name
   b. Select the Managed Identity
   c. Select the Key Vault from where to get the certificate
   d. Select the certificate
7. Under "Listener Type", select "Basic"
8. Click "Add" to add the listener

Configure request routing rule

Provided with the earlier configured Backend Pool and the HTTP Settings, the request routing rule can be set up to take traffic from a listener and route it to the Backend Pool using the HTTP Settings. For this, make sure you have an HTTP or HTTPS listener available that is not already bound to an existing routing rule.

- Azure portal
- PowerShell

1. Under "Rules", click to add a new "Request routing rule"
2. Provide the rule with a name

3. Select an HTTP or HTTPS listener that is not bound yet to an existing routing rule
4. Under "Backend targets", choose the Backend Pool in which App Service has been configured
5. Configure the HTTP settings with which Application Gateway should connect to the App Service backend
6. Select "Add" to save this configuration

Testing

Before we do so, make sure that the backend health shows as healthy:

- Azure portal
- PowerShell

Open the "Backend health" section and ensure the "Status" column indicates the combination for HTTP Setting and Backend Pool shows as "Healthy".

Now browse to the web application using the custom domain which you associated with both Application Gateway and the App Service in the backend.

Restrict access

The web apps deployed in these examples use public IP addresses that can be accessed directly from the Internet. This helps with troubleshooting when you are learning about a new feature and trying new things. But if you intend to deploy a feature into production, you'll want to add more restrictions. Consider the following options:

- Configure Access restriction rules based on service endpoints. This allows you to lock down inbound access to the app making sure the source address is from Application Gateway.
- Use Azure App Service static IP restrictions. For example, you can restrict the web app so that it only receives traffic from the application gateway. Use the app service IP restriction feature to list the application gateway VIP as the only address with access.

Create a custom probe for Application Gateway by using the portal

8/22/2022 • 8 minutes to read • Edit Online

In this article, you add a custom health probe to an existing application gateway through the Azure portal. Azure Application Gateway uses these health probes to monitor the health of the resources in the back-end pool.

Before you begin

If you don't already have an application gateway, visit Create an Application Gateway to create an application gateway to work with.

Create probe for Application Gateway v2 SKU

Probes are configured in a two-step process through the portal. The first step is to enter the values required for the probe configuration. In the second step, you test the backend health using this probe configuration and save the probe.

Enter probe properties

1. Sign in to the Azure portal. If you don't already have an account, you can sign up for a free one-month trial

2. In the Azure portal Favorites pane, click All resources. Click the application gateway in the All resources blade. If the subscription you selected already has several resources in it, you can enter partners.contoso.net in the Filter by name... box to easily access the application gateway.

3. Select **Health probes** and then select **Add** to add a new health probe.

4. On the **Add health probe** page, fill out the required information for the probe, and when complete select **OK**.

SETTING	VALUE	DETAILS
Name	customProbe	This value is a friendly name given to the probe that is accessible in the portal.
Protocol	HTTP or HTTPS	The protocol that the health probe uses.
Host	i.e contoso.com	This value is the name of the virtual host (different from the VM host name) running on the application server. The probe is sent to <protocol>://<host name>:<port>/<urlPath> This can also be the private IP address of the server, or the public IP address, or the DNS entry of the public IP address. The probe will attempt to access the server when used with a file based path entry, and validate a specific file exists on the server as a health check.
Pick host name from backend HTTP settings	Yes or No	Sets the *host* header in the probe to the host name from the HTTP settings to which this probe is associated. Specially required for multi-tenant backends such as Azure app service. Learn more
Pick port from backend HTTP settings	Yes or No	Sets the *port* of the health probe to the port from HTTP settings to which this probe is associated. If you choose no, you can enter a custom destination port to use
Port	1-65535	Custom port to be used for the health probes
Path	/ or any valid path	The remainder of the full url for the custom probe. A valid path starts with '/'. For the default path of http://contoso.com, just use '/'. You can also input a server path to a file for a static health check instead of web based. File paths should be used while using public / private ip, or public ip dns entry as the hostname entry.
Interval (secs)	30	How often the probe is run to check for health. It isn't recommended to set the lower than 30 seconds.

SETTING	VALUE	DETAILS
Timeout (secs)	30	The amount of time the probe waits before timing out. If a valid response isn't received within this time-out period, the probe is marked as failed. The timeout interval needs to be high enough that an http call can be made to ensure the backend health page is available. The time-out value shouldn't be more than the 'Interval' value used in this probe setting or the 'Request timeout' value in the HTTP setting, which will be associated with this probe.
Unhealthy threshold	3	Number of consecutive failed attempts to be considered unhealthy. The threshold can be set to 1 or more.
Use probe matching conditions	Yes or No	By default, an HTTP(S) response with status code between 200 and 399 is considered healthy. You can change the acceptable range of backend response code or backend response body. Learn more
HTTP Settings	selection from dropdown	Probe will get associated with the HTTP settings selected here and therefore, will monitor the health of that backend pool, which is associated with the selected HTTP setting. It will use the same port for the probe request as the one being used in the selected HTTP setting. You can only choose those HTTP settings, which aren't associated with any other custom probe. The only HTTP settings that are available for association are those that have the same protocol as the protocol chosen in this probe configuration, and have the same state for the *Pick Host Name From Backend HTTP setting* switch.

> **IMPORTANT**
> The probe will monitor health of the backend only when it's associated with one or more HTTP settings. It will monitor back-end resources of those back-end pools which are associated to the HTTP settings to which this probe is associated with. The probe request will be sent as <protocol>://<hostName>:<port>/<urlPath>.

Test backend health with the probe

After entering the probe properties, you can test the health of the back-end resources to verify that the probe configuration is correct and that the back-end resources are working as expected.

1. Select **Test** and note the result of the probe. The Application gateway tests the health of all the backend

resources in the backend pools associated with the HTTP settings used for this probe.

2. If there are any unhealthy backend resources, then check the **Details** column to understand the reason for unhealthy state of the resource. If the resource has been marked unhealthy due to an incorrect probe configuration, then select the **Go back to probe** link and edit the probe configuration. Otherwise, if the resource has been marked unhealthy due to an issue with the backend, then resolve the issues with the backend resource and then test the backend again by selecting the **Go back to probe** link and select **Test**.

> **NOTE**
>
> You can choose to save the probe even with unhealthy backend resources, but it isn't recommended. This is because the Application Gateway will not forward requests to the backend servers from the backend pool, which are determined to be unhealthy by the probe. In case there are no healthy resources in a backend pool, you will not be able to access your application and will get a HTTP 502 error.

3. Select **Add** to save the probe.

Create probe for Application Gateway v1 SKU

Probes are configured in a two-step process through the portal. The first step is to create the probe. In the second step, you add the probe to the backend http settings of the application gateway.

Create the probe

1. Sign in to the Azure portal. If you don't already have an account, you can sign up for a free one-month trial

2. In the Azure portal Favorites pane, select **All resources**. Select the application gateway in the **All resources** page. If the subscription you selected already has several resources in it, you can enter partners.contoso.net in the Filter by name... box to easily access the application gateway.

3. Select **Probes** and then select **Add** to add a probe.

4. On the **Add health probe** blade, fill out the required information for the probe, and when complete select **OK**.

SETTING	VALUE	DETAILS
Name	customProbe	This value is a friendly name given to the probe that is accessible in the portal.
Protocol	HTTP or HTTPS	The protocol that the health probe uses.

SETTING	VALUE	DETAILS
Host	i.e contoso.com	This value is the name of the virtual host (different from the VM host name) running on the application server. The probe is sent to (protocol)://(host name):(port from httpsetting)/urlPath. This is applicable when multi-site is configured on Application Gateway. If the Application Gateway is configured for a single site, then enter '127.0.0.1'. You can also input a server path to a file for a static health check instead of web based. File paths should be used while using public / private ip, or public ip dns entry as the hostname entry.
Pick host name from backend HTTP settings	Yes or No	Sets the *host* header in the probe to the host name of the back-end resource in the back-end pool associated with the HTTP Setting to which this probe is associated. Specially required for multi-tenant backends such as Azure app service. Learn more
Path	/ or any valid path	The remainder of the full url for the custom probe. A valid path starts with '/'. For the default path of http://contoso.com, just use '/' You can also input a server path to a file for a static health check instead of web based. File paths should be used while using public / private ip, or public ip dns entry as the hostname entry.
Interval (secs)	30	How often the probe is run to check for health. It isn't recommended to set the lower than 30 seconds.
Timeout (secs)	30	The amount of time the probe waits before timing out. If a valid response isn't received within this time-out period, the probe is marked as failed. The timeout interval needs to be high enough that an http call can be made to ensure the backend health page is available. The time-out value shouldn't be more than the 'Interval' value used in this probe setting or the 'Request timeout' value in the HTTP setting, which will be associated with this probe.

SETTING	VALUE	DETAILS
Unhealthy threshold	3	Number of consecutive failed attempts to be considered unhealthy. The threshold can be set to 1 or more.
Use probe matching conditions	Yes or No	By default, an HTTP(S) response with status code between 200 and 399 is considered healthy. You can change the acceptable range of backend response code or backend response body. Learn more

> **IMPORTANT**
>
> The host name isn't the same as server name. This value is the name of the virtual host running on the application server. The probe is sent to <protocol>://<hostName>:<port from http settings>/<urlPath>

Add probe to the gateway

Now that the probe has been created, it's time to add it to the gateway. Probe settings are set on the backend http settings of the application gateway.

1. Click **HTTP settings** on the application gateway, to bring up the configuration blade click the current backend http settings listed in the window.

2. On the **appGatewayBackEndHttpSettings** settings page, check the **Use custom probe** checkbox and choose the probe created in the Create the probe section on the **Custom probe** drop-down. When complete, click **Save** and the settings are applied.

Next steps

View the health of the backend resources as determined by the probe using the backend health view.

Create a custom probe for Azure Application Gateway (classic) by using PowerShell

8/22/2022 • 4 minutes to read • Edit Online

In this article, you add a custom probe to an existing application gateway with PowerShell. Custom probes are useful for applications that have a specific health check page or for applications that do not provide a successful response on the default web application.

> **IMPORTANT**
>
> Azure has two different deployment models for creating and working with resources: Resource Manager and Classic. This article covers using the Classic deployment model. Microsoft recommends that most new deployments use the Resource Manager model. Learn how to perform these steps using the Resource Manager model.

Prerequisite: Install the Azure PowerShell module

To perform the steps in this article, you need to install and configure the Azure PowerShell module. Be sure to complete all of the instructions. After the installation is finished, sign in to Azure and select your subscription.

> **NOTE**
>
> You need an Azure account to complete these steps. If you don't have an Azure account, you can sign up for a free trial.

Create an application gateway

To create an application gateway:

1. Create an application gateway resource.
2. Create a configuration XML file or a configuration object.
3. Commit the configuration to the newly created application gateway resource.

Create an application gateway resource with a custom probe

To create the gateway, use the `New-AzureApplicationGateway` cmdlet, replacing the values with your own. Billing for the gateway does not start at this point. Billing begins in a later step, when the gateway is successfully started.

The following example creates an application gateway by using a virtual network called "testvnet1" and a subnet called "subnet-1".

```
New-AzureApplicationGateway -Name AppGwTest -VnetName testvnet1 -Subnets @("Subnet-1")
```

To validate that the gateway was created, you can use the `Get-AzureApplicationGateway` cmdlet.

```
Get-AzureApplicationGateway AppGwTest
```

> **NOTE**
>
> The default value for *InstanceCount* is 2, with a maximum value of 10. The default value for *GatewaySize* is Medium. You can choose between Small, Medium, and Large.

VirtualIPs and *DnsName* are shown as blank because the gateway has not started yet. These values are created once the gateway is in the running state.

Configure an application gateway by using XML

In the following example, you use an XML file to configure all application gateway settings and commit them to the application gateway resource.

Copy the following text to Notepad.

```
<ApplicationGatewayConfiguration xmlns:i="https://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/windowsazure">
<FrontendIPConfigurations>
    <FrontendIPConfiguration>
        <Name>fip1</Name>
        <Type>Private</Type>
    </FrontendIPConfiguration>
</FrontendIPConfigurations>
<FrontendPorts>
    <FrontendPort>
        <Name>port1</Name>
        <Port>80</Port>
    </FrontendPort>
</FrontendPorts>
<Probes>
    <Probe>
        <Name>Probe01</Name>
        <Protocol>Http</Protocol>
        <Host>contoso.com</Host>
        <Path>/path/custompath.htm</Path>
        <Interval>15</Interval>
        <Timeout>15</Timeout>
        <UnhealthyThreshold>5</UnhealthyThreshold>
    </Probe>
    </Probes>
    <BackendAddressPools>
    <BackendAddressPool>
        <Name>pool1</Name>
        <IPAddresses>
            <IPAddress>1.1.1.1</IPAddress>
            <IPAddress>2.2.2.2</IPAddress>
        </IPAddresses>
    </BackendAddressPool>
</BackendAddressPools>
<BackendHttpSettingsList>
    <BackendHttpSettings>
        <Name>setting1</Name>
        <Port>80</Port>
        <Protocol>Http</Protocol>
        <CookieBasedAffinity>Enabled</CookieBasedAffinity>
        <RequestTimeout>120</RequestTimeout>
        <Probe>Probe01</Probe>
    </BackendHttpSettings>
</BackendHttpSettingsList>
<HttpListeners>
    <HttpListener>
        <Name>listener1</Name>
        <FrontendIP>fip1</FrontendIP>
    <FrontendPort>port1</FrontendPort>
        <Protocol>Http</Protocol>
    </HttpListener>
</HttpListeners>
<HttpLoadBalancingRules>
    <HttpLoadBalancingRule>
        <Name>lbrule1</Name>
        <Type>basic</Type>
        <BackendHttpSettings>setting1</BackendHttpSettings>
        <Listener>listener1</Listener>
        <BackendAddressPool>pool1</BackendAddressPool>
    </HttpLoadBalancingRule>
</HttpLoadBalancingRules>
</ApplicationGatewayConfiguration>
```

Edit the values between the parentheses for the configuration items. Save the file with extension .xml.

The following example shows how to use a configuration file to set up the application gateway to load balance

HTTP traffic on public port 80 and send network traffic to back-end port 80 between two IP addresses by using a custom probe.

> **IMPORTANT**
>
> The protocol item Http or Https is case-sensitive.

A new configuration item <Probe> is added to configure custom probes.

The configuration parameters are:

PARAMETER	DESCRIPTION
Name	Reference name for custom probe.
Protocol	Protocol used (possible values are HTTP or HTTPS).
Host and **Path**	Complete URL path that is invoked by the application gateway to determine the health of the instance. For example, if you have a website http://contoso.com/, then the custom probe can be configured for "http://contoso.com/path/custompath.htm" for probe checks to have a successful HTTP response.
Interval	Configures the probe interval checks in seconds.
Timeout	Defines the probe time-out for an HTTP response check.
UnhealthyThreshold	The number of failed HTTP responses needed to flag the back-end instance as *unhealthy*.

The probe name is referenced in the <BackendHttpSettings> configuration to assign which back-end pool uses custom probe settings.

Add a custom probe to an existing application gateway

Changing the current configuration of an application gateway requires three steps: Get the current XML configuration file, modify to have a custom probe, and configure the application gateway with the new XML settings.

1. Get the XML file by using `Get-AzureApplicationGatewayConfig`. This cmdlet exports the configuration XML to be modified to add a probe setting.

   ```
   Get-AzureApplicationGatewayConfig -Name "<application gateway name>" -Exporttofile "<path to file>"
   ```

2. Open the XML file in a text editor. Add a `<probe>` section after `<frontendport>`.

```
<Probes>
 <Probe>
     <Name>Probe01</Name>
     <Protocol>Http</Protocol>
     <Host>contoso.com</Host>
     <Path>/path/custompath.htm</Path>
     <Interval>15</Interval>
     <Timeout>15</Timeout>
     <UnhealthyThreshold>5</UnhealthyThreshold>
 </Probe>
</Probes>
```

In the backendHttpSettings section of the XML, add the probe name as shown in the following example:

```
 <BackendHttpSettings>
     <Name>setting1</Name>
     <Port>80</Port>
     <Protocol>Http</Protocol>
     <CookieBasedAffinity>Enabled</CookieBasedAffinity>
     <RequestTimeout>120</RequestTimeout>
     <Probe>Probe01</Probe>
 </BackendHttpSettings>
```

Save the XML file.

3. Update the application gateway configuration with the new XML file by using `Set-AzureApplicationGatewayConfig`. This cmdlet updates your application gateway with the new configuration.

```
Set-AzureApplicationGatewayConfig -Name "<application gateway name>" -Configfile "<path to file>"
```

Next steps

If you want to configure Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL) offload, see Configure an application gateway for TLS offload.

If you want to configure an application gateway to use with an internal load balancer, see Create an application gateway with an internal load balancer (ILB).

Create a custom probe for Azure Application Gateway by using PowerShell for Azure Resource Manager

8/22/2022 • 6 minutes to read • Edit Online

In this article, you add a custom probe to an existing application gateway with PowerShell. Custom probes are useful for applications that have a specific health check page or for applications that do not provide a successful response on the default web application.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisite: Install the Azure PowerShell module

To perform the steps in this article, you need to install and configure the Azure PowerShell module. Be sure to complete all of the instructions. After the installation is finished, sign in to Azure and select your subscription.

> **NOTE**
> You need an Azure account to complete these steps. If you don't have an Azure account, you can sign up for a free trial.

Create an application gateway with a custom probe

Sign in and create resource group

1. Use `Connect-AzAccount` to authenticate.

```
Connect-AzAccount
```

2. Get the subscriptions for the account.

```
Get-AzSubscription
```

3. Choose which of your Azure subscriptions to use.

```
Select-AzSubscription -Subscriptionid '{subscriptionGuid}'
```

4. Create a resource group. You can skip this step if you have an existing resource group.

```
New-AzResourceGroup -Name appgw-rg -Location 'West US'
```

Azure Resource Manager requires that all resource groups specify a location. This location is used as the default

location for resources in that resource group. Make sure that all commands to create an application gateway use the same resource group.

In the preceding example, we created a resource group called **appgw-RG** in location **West US**.

Create a virtual network and a subnet

The following example creates a virtual network and a subnet for the application gateway. Application gateway requires its own subnet for use. For this reason, the subnet created for the application gateway should be smaller than the address space of the VNET to allow for other subnets to be created and used.

```
# Assign the address range 10.0.0.0/24 to a subnet variable to be used to create a virtual network.
$subnet = New-AzVirtualNetworkSubnetConfig -Name subnet01 -AddressPrefix 10.0.0.0/24

# Create a virtual network named appgwvnet in resource group appgw-rg for the West US region using the
prefix 10.0.0.0/16 with subnet 10.0.0.0/24.
$vnet = New-AzVirtualNetwork -Name appgwvnet -ResourceGroupName appgw-rg -Location 'West US' -AddressPrefix
10.0.0.0/16 -Subnet $subnet

# Assign a subnet variable for the next steps, which create an application gateway.
$subnet = $vnet.Subnets[0]
```

Create a public IP address for the front-end configuration

Create a public IP resource **publicIP01** in resource group **appgw-rg** for the West US region. This example uses a public IP address for the front-end IP address of the application gateway. Application gateway requires the public IP address to have a dynamically created DNS name therefore the `-DomainNameLabel` cannot be specified during the creation of the public IP address.

```
$publicip = New-AzPublicIpAddress -ResourceGroupName appgw-rg -Name publicIP01 -Location 'West US' -
AllocationMethod Dynamic
```

Create an application gateway

You set up all configuration items before creating the application gateway. The following example creates the configuration items that are needed for an application gateway resource.

COMPONENT	DESCRIPTION
Gateway IP configuration	An IP configuration for an application gateway.
Backend pool	A pool of IP addresses, FQDN's, or NICs that are to the application servers that host the web application
Health probe	A custom probe used to monitor the health of the backend pool members
HTTP settings	A collection of settings including, port, protocol, cookie-based affinity, probe, and timeout. These settings determine how traffic is routed to the backend pool members
Frontend port	The port that the application gateway listens for traffic on
Listener	A combination of a protocol, frontend IP configuration, and frontend port. This is what listens for incoming requests.
Rule	Routes the traffic to the appropriate backend based on HTTP settings.

```
# Creates an application gateway Frontend IP configuration named gatewayIP01
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name gatewayIP01 -Subnet $subnet

#Creates a back-end IP address pool named pool01 with IP addresses 134.170.185.46, 134.170.188.221,
134.170.185.50.
$pool = New-AzApplicationGatewayBackendAddressPool -Name pool01 -BackendIPAddresses 134.170.185.46,
134.170.188.221, 134.170.185.50

# Creates a probe that will check health at http://contoso.com/path/path.htm
$probe = New-AzApplicationGatewayProbeConfig -Name probe01 -Protocol Http -HostName 'contoso.com' -Path
'/path/path.htm' -Interval 30 -Timeout 120 -UnhealthyThreshold 8

# Creates the backend http settings to be used. This component references the $probe created in the previous
command.
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name poolsetting01 -Port 80 -Protocol Http -
CookieBasedAffinity Disabled -Probe $probe -RequestTimeout 80

# Creates a frontend port for the application gateway to listen on port 80 that will be used by the
listener.
$fp = New-AzApplicationGatewayFrontendPort -Name frontendport01 -Port 80

# Creates a frontend IP configuration. This associates the $publicip variable defined previously with the
front-end IP that will be used by the listener.
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name fipconfig01 -PublicIPAddress $publicip

# Creates the listener. The listener is a combination of protocol and the frontend IP configuration
$fipconfig and frontend port $fp created in previous steps.
$listener = New-AzApplicationGatewayHttpListener -Name listener01  -Protocol Http -FrontendIPConfiguration
$fipconfig -FrontendPort $fp

# Creates the rule that routes traffic to the backend pools.  In this example we create a basic rule that
uses the previous defined http settings and backend address pool.  It also associates the listener to the
rule
$rule = New-AzApplicationGatewayRequestRoutingRule -Name rule01 -RuleType Basic -BackendHttpSettings
$poolSetting -HttpListener $listener -BackendAddressPool $pool

# Sets the SKU of the application gateway, in this example we create a small standard application gateway
with 2 instances.
$sku = New-AzApplicationGatewaySku -Name Standard_Small -Tier Standard -Capacity 2

# The final step creates the application gateway with all the previously defined components.
$appgw = New-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg -Location 'West US' -
BackendAddressPools $pool -Probes $probe -BackendHttpSettingsCollection $poolSetting -
FrontendIpConfigurations $fipconfig  -GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners
$listener -RequestRoutingRules $rule -Sku $sku
```

Add a probe to an existing application gateway

The following code snippet adds a probe to an existing application gateway.

```
# Load the application gateway resource into a PowerShell variable by using Get-AzApplicationGateway.
$getgw =  Get-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg

# Create the probe object that will check health at http://contoso.com/path/path.htm
$probe = Add-AzApplicationGatewayProbeConfig -ApplicationGateway $getgw -Name probe01 -Protocol Http -
HostName 'contoso.com' -Path '/path/custompath.htm' -Interval 30 -Timeout 120 -UnhealthyThreshold 8

# Set the backend HTTP settings to use the new probe
$getgw = Set-AzApplicationGatewayBackendHttpSettings -ApplicationGateway $getgw -Name
$getgw.BackendHttpSettingsCollection.name -Port 80 -Protocol Http -CookieBasedAffinity Disabled -Probe
$probe -RequestTimeout 120

# Save the application gateway with the configuration changes
Set-AzApplicationGateway -ApplicationGateway $getgw
```

Remove a probe from an existing application gateway

The following code snippet removes a probe from an existing application gateway.

```
# Load the application gateway resource into a PowerShell variable by using Get-AzApplicationGateway.
$getgw =  Get-AzApplicationGateway -Name appgwtest -ResourceGroupName appgw-rg

# Remove the probe from the application gateway configuration object
$getgw = Remove-AzApplicationGatewayProbeConfig -ApplicationGateway $getgw -Name $getgw.Probes.name

# Set the backend HTTP settings to remove the reference to the probe. The backend http settings now use the
default probe
$getgw = Set-AzApplicationGatewayBackendHttpSettings -ApplicationGateway $getgw -Name
$getgw.BackendHttpSettingsCollection.name -Port 80 -Protocol http -CookieBasedAffinity Disabled

# Save the application gateway with the configuration changes
Set-AzApplicationGateway -ApplicationGateway $getgw
```

Get application gateway DNS name

Once the gateway is created, the next step is to configure the front end for communication. When using a public IP, application gateway requires a dynamically assigned DNS name, which is not friendly. To ensure end users can hit the application gateway a CNAME record can be used to point to the public endpoint of the application gateway. Configuring a custom domain name for in Azure. To do this, retrieve details of the application gateway and its associated IP/DNS name using the PublicIPAddress element attached to the application gateway. The application gateway's DNS name should be used to create a CNAME record, which points the two web applications to this DNS name. The use of A-records is not recommended since the VIP may change on restart of application gateway.

```
Get-AzPublicIpAddress -ResourceGroupName appgw-RG -Name publicIP01
```

```
Name                      : publicIP01
ResourceGroupName         : appgw-RG
Location                  : westus
Id                        : /subscriptions/<subscription_id>/resourceGroups/appgw-
RG/providers/Microsoft.Network/publicIPAddresses/publicIP01
Etag                      : W/"00000d5b-54ed-4907-bae8-99bd5766d0e5"
ResourceGuid              : 00000000-0000-0000-0000-000000000000
ProvisioningState         : Succeeded
Tags                      :
PublicIpAllocationMethod : Dynamic
IpAddress                 : xx.xx.xxx.xx
PublicIpAddressVersion    : IPv4
IdleTimeoutInMinutes      : 4
IpConfiguration           : {
                                "Id": "/subscriptions/<subscription_id>/resourceGroups/appgw-
RG/providers/Microsoft.Network/applicationGateways/appgwtest/frontendIP
                             Configurations/frontend1"
                             }
DnsSettings               : {
                                "Fqdn": "00000000-0000-xxxx-xxxx-xxxxxxxxxxxx.cloudapp.net"
                             }
```

Next steps

Learn to configure TLS offloading by visiting: Configure TLS Offload

Back-end server certificate isn't allow-listed for an application gateway using an Internal Load Balancer with an App Service Environment

8/22/2022 • 2 minutes to read • Edit Online

This article troubleshoots the following issue: A certificate isn't allow-listed when you create an application gateway by using an Internal Load Balancer (ILB) together with an App Service Environment (ASE) at the back end when using end-to-end TLS in Azure.

Symptoms

When you create an application gateway by using an ILB with an ASE at the back end, the back-end server may become unhealthy. This problem occurs if the authentication certificate of the application gateway doesn't match the configured certificate on the back-end server. See the following scenario as an example:

Application Gateway configuration:

- **Listener:** Multi-site
- **Port:** 443
- **Hostname:** test.appgwtestase.com
- **SSL Certificate:** CN=test.appgwtestase.com
- **Backend Pool:** IP address or FQDN
- **IP Address::** 10.1.5.11
- **HTTP Settings:** HTTPS
- **Port::** 443
- **Custom Probe:** Hostname – test.appgwtestase.com
- **Authentication Certificate:** .cer of test.appgwtestase.com
- **Backend Health:** Unhealthy – Backend server certificate isn't allow-listed with Application Gateway.

ASE configuration:

- **ILB IP:** 10.1.5.11
- **Domain name:** appgwtestase.com
- **App Service:** test.appgwtestase.com
- **SSL Binding:** SNI SSL – CN=test.appgwtestase.com

When you access the application gateway, you receive the following error message because the back-end server is unhealthy:

502 – Web server received an invalid response while acting as a gateway or proxy server.

Solution

When you don't use a host name to access an HTTPS website, the back-end server will return the configured certificate on the default website, in case SNI is disabled. For an ILB ASE, the default certificate comes from the ILB certificate. If there are no configured certificates for the ILB, the certificate comes from the ASE App certificate.

When you use a fully qualified domain name (FQDN) to access the ILB, the back-end server will return the

correct certificate that's uploaded in the HTTP settings. If that isn't the case , consider the following options:

- Use FQDN in the back-end pool of the application gateway to point to the IP address of the ILB. This option only works if you have a private DNS zone or a custom DNS configured. Otherwise, you have to create an "A" record for a public DNS.
- Use the uploaded certificate on the ILB or the default certificate (ILB certificate) in the HTTP settings. The application gateway gets the certificate when it accesses the ILB's IP for the probe.
- Use a wildcard certificate on the ILB and the back-end server, so that for all the websites, the certificate is common. However, this solution is possible only for subdomains and not if each of the websites require different hostnames.
- Clear the **Use for App service** option for the application gateway in case you're using the IP address of the ILB.

To reduce overhead, you can upload the ILB certificate in the HTTP settings to make the probe path work. (This step is just for allow listing. It won't be used for TLS communication.) You can retrieve the ILB certificate by accessing the ILB with its IP address from your browser on HTTPS then exporting the TLS/SSL certificate in a Base-64 encoded CER format and uploading the certificate on the respective HTTP settings.

Need help? Contact support

If you still need help, contact support to get your issue resolved quickly.

Troubleshoot App Service issues in Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

Learn how to diagnose and resolve issues you might encounter when Azure App Service is used as a back-end target with Azure Application Gateway.

Overview

In this article, you'll learn how to troubleshoot the following issues, as described in more detail in Architecture Center: Preserve the original HTTP host name between a reverse proxy and its backend web application

- Incorrect absolute URLs
- Incorrect redirect URLs
  - the app service URL is exposed in the browser when there's a redirection
  - an example of this: an OIDC authentication flow is broken because of a redirect with wrong hostname; this includes the use of App Service Authentication and Authorization
- Broken cookies
  - cookies are not propagated between the browser and the App Service
  - an example of this: the app service ARRAffinity cookie domain is set to the app service host name and is tied to "example.azurewebsites.net", instead of the original host. As a result, session affinity is broken.

The root-cause for the above symptoms is a setup that overrides the hostname as used by Application Gateway towards App Service into a different hostname as is seen by the browser. Often the hostname is overridden to the default App Service "azurewebsites.net" domain.

Sample configuration

In case your configuration matches one of below two situations, your setup is subject to the instructions in this article:

- **Pick Hostname from Backend Address** is enabled in HTTP Settings
- **Override with specific domain name** is set to a value different from what the browser request has

Cause

App Service is a multitenant service, so it uses the host header in the request to route the request to the correct endpoint. The default domain name of App Services, *.azurewebsites.net (say, contoso.azurewebsites.net), is different from the application gateway's domain name (say, contoso.com). The backend App Service is missing

the required context to generate redirect url's or cookies that align with the domain as seen by the browser.

Solution

The production-recommended solution is to configure Application Gateway and App Service to not override the hostname. Follow the instructions for **"Custom Domain (recommended)"** in Configure App Service with Application Gateway

Only consider applying another workaround (like a rewrite of the Location header as described below) after assessing the implications as described in the article: Preserve the original HTTP host name between a reverse proxy and its backend web application. These implications include the potential for domain-bound cookies and for absolute URL's outside of the location header, to remain broken.

Workaround: rewrite the Location header

> **WARNING**
>
> This configuration comes with limitations. We recommend to review the implications of using different host names between the client and Application Gateway and between Application and App Service in the backend. For more information, please review the article in Architecture Center: Preserve the original HTTP host name between a reverse proxy and its backend web application

Set the host name in the location header to the application gateway's domain name. To do this, create a rewrite rule with a condition that evaluates if the location header in the response contains azurewebsites.net. It must also perform an action to rewrite the location header to have the application gateway's host name. For more information, see instructions on how to rewrite the location header.

> **NOTE**
>
> The HTTP header rewrite support is only available for the Standard_v2 and WAF_v2 SKU of Application Gateway. We recommend migrating to v2 for Header Rewrite and other advanced capabilities that are available with v2 SKU.

Next steps

If the preceding steps didn't resolve the issue, open a support ticket.

Troubleshoot Azure Application Gateway session affinity issues

8/22/2022 • 6 minutes to read • Edit Online

Learn how to diagnose and resolve session affinity issues with Azure Application Gateway.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Overview

The cookie-based session affinity feature is useful to keep a user session on the same server. By using gateway-managed cookies, the Application Gateway can direct subsequent traffic from a user session to the same server for processing. This is important in cases where session state is saved locally on the server for a user session. Session affinity is also known as sticky sessions.

> **NOTE**
>
> Application Gateway v1 issues a cookie called ARRAffinity, which is used to direct traffic to the same backend pool member. In Application Gateway v2, this cookie has been renamed to ApplicationGatewayAffinity. For the purposes of this document, ApplicationGatewayAffinity will be used as an example, ARRAffinity can be substituted in where applicable for Application Gateway v1 instances.

Possible problem causes

The problem in maintaining cookie-based session affinity may happen due to the following main reasons:

- "Cookie-based Affinity" setting is not enabled
- Your application cannot handle cookie-based affinity
- Application is using cookie-based affinity but requests still bouncing between back-end servers

Check whether the "Cookie-based Affinity" setting is enabled

Sometimes the session affinity issues might occur when you forget to enable "Cookie based affinity" setting. To determine whether you have enabled the "Cookie based affinity" setting on the HTTP Settings tab in the Azure portal, follow the instructions:

1. Log on to the Azure portal.
2. In the **left navigation** pane, click **All resources**. Click the application gateway name in the All resources blade. If the subscription that you selected already has several resources in it, you can enter the application gateway name in the **Filter by name...** box to easily access the application gateway.
3. Select **HTTP settings** tab under **SETTINGS**.

4. Select the HTTP setting, and on the **Add HTTP setting** page, check if **Cookie based affinity** is enabled.

You can also check the value of the "**CookieBasedAffinity**" is set to *Enabled*under "**backendHttpSettingsCollection**" by using one of the following methods:

- Run Get-AzApplicationGatewayBackendHttpSetting in PowerShell
- Look through the JSON file by using the Azure Resource Manager template

```
"cookieBasedAffinity": "Enabled",
```

The application cannot handle cookie-based affinity

Cause

The application gateway can only perform session-based affinity by using a cookie.

Workaround

If the application cannot handle cookie-based affinity, you must use an external or internal azure load balancer or another third-party solution.

Application is using cookie-based affinity but requests still bouncing between back-end servers

Symptom

You have enabled the Cookie-based Affinity setting, when you access the Application Gateway by using a short name URL in Internet Explorer, for example: `http://website` , the request is still bouncing between back-end servers.

To identify this issue, follow the instructions:

1. Take a web debugger trace on the “Client” which is connecting to the application behind the Application Gateway(We are using Fiddler in this example). **Tip** If you don't know how to use the Fiddler, check the option "**I want to collect network traffic and analyze it using web debugger**" at the bottom.

2. Check and analyze the session logs, to determine whether the cookies provided by the client have the ApplicationGatewayAffinity details. If you don't find the ApplicationGatewayAffinity details, such as "**ApplicationGatewayAffinity=** *ApplicationGatewayAffinityValue*" within the cookie set, that means the client is not replying with the ApplicationGatewayAffinity cookie, which is provided by the Application Gateway. For example:

The application continues to try to set the cookie on each request until it gets reply.

Cause

This issue occurs because Internet Explorer and other browsers may not store or use the cookie with a short name URL.

Resolution

To fix this issue, you should access the Application Gateway by using a FQDN. For example, use http://website.com or http://appgw.website.com .

Additional logs to troubleshoot

You can collect additional logs and analyze them to troubleshoot the issues related cookie-based session affinity

Analyze Application Gateway logs

To collect the Application Gateway logs, follow the instructions:

Enable logging using the Azure portal.

1. In the Azure portal, find your resource and then select **Diagnostic setting**.

   For Application Gateway, three logs are available: Access log, Performance log, and Firewall log.

2. To start to collect data, select **Add diagnostic setting**.

3. The **Diagnostic setting** page provides the settings for the diagnostic logs. In this example, Log Analytics stores the logs. You can also use event hubs and a storage account to save the diagnostic logs.

4. Confirm the settings and then select **Save**.

Use web debugger to capture and analyze the HTTP or HTTPS traffics

Web debugging tools like Fiddler, can help you debug web applications by capturing network traffic between the Internet and test computers. These tools enable you to inspect incoming and outgoing data as the browser receives/sends them. Fiddler, in this example, has the HTTP replay option that can help you troubleshoot client-side issues with web applications, especially for authentication kind of issue.

Use the web debugger of your choice. In this sample we will use Fiddler to capture and analyze http or https traffics, follow the instructions:

1. Download Fiddler.

> **NOTE**
>
> Choose Fiddler4 if the capturing computer has .NET 4 installed. Otherwise, choose Fiddler2.

2. Right click the setup executable, and run as administrator to install.

3. When you open Fiddler, it should automatically start capturing traffic (notice the Capturing at lower-left-hand corner). Press F12 to start or stop traffic capture.

4. Most likely, you will be interested in decrypted HTTPS traffic, and you can enable HTTPS decryption by selecting **Tools** > **Fiddler Options**, and check the box " **Decrypt HTTPS traffic**".

5. You can remove previous unrelated sessions before reproducing the issue by clicking **X** (icon) > **Remove All** as follow screenshot:

6. Once you have reproduced the issue, save the file for review by selecting **File** > **Save** > **All Sessions...**

7. Check and analyze the session logs to determine what the issue is.

   For examples:

- **Example A:** You find a session log that the request is sent from the client, and it goes to the public IP address of the Application Gateway, click this log to view the details. On the right side, data in the bottom box is what the Application Gateway is returning to the client. Select the "RAW" tab and determine whether the client is receiving a "**Set-Cookie: ApplicationGatewayAffinity=** *ApplicationGatewayAffinityValue*." If there's no cookie, session affinity isn't set, or the Application Gateway isn't applying cookie back to the client.

> **NOTE**
>
> This ApplicationGatewayAffinity value is the cookie-id, that the Application Gateway sets for the client to be sent to a particular back-end server.

- **Example B:** The next session log followed by the previous one is the client responding back to the Application Gateway, which has set the ApplicationGatewayAffinity. If the ApplicationGatewayAffinity cookie-id matches, the packet should be sent to the same back-end server that was used previously. Check the next several lines of http communication to see whether the client's ApplicationGatewayAffinity cookie is changing.

> **NOTE**
> For the same communication session, the cookie should not to change. Check the top box on the right side, select "Cookies" tab to see whether the client is using the cookie and sending it back to the Application Gateway. If not, the client browser isn't keeping and using the cookie for conversations. Sometimes, the client might lie.

Next steps

If the preceding steps do not resolve the issue, open a support ticket.

Troubleshooting bad gateway errors in Application Gateway

8/22/2022 • 7 minutes to read • Edit Online

Learn how to troubleshoot bad gateway (502) errors received when using Azure Application Gateway.

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Overview

After configuring an application gateway, one of the errors that you may see is "Server Error: 502 - Web server received an invalid response while acting as a gateway or proxy server". This error may happen for the following main reasons:

- NSG, UDR, or Custom DNS is blocking access to backend pool members.
- Back-end VMs or instances of virtual machine scale set aren't responding to the default health probe.
- Invalid or improper configuration of custom health probes.
- Azure Application Gateway's back-end pool isn't configured or empty.
- None of the VMs or instances in virtual machine scale set are healthy.
- Request time-out or connectivity issues with user requests.

Network Security Group, User Defined Route, or Custom DNS issue

Cause

If access to the backend is blocked because of an NSG, UDR, or custom DNS, application gateway instances can't reach the backend pool. This causes probe failures, resulting in 502 errors.

The NSG/UDR could be present either in the application gateway subnet or the subnet where the application VMs are deployed.

Similarly, the presence of a custom DNS in the VNet could also cause issues. A FQDN used for backend pool members might not resolve correctly by the user configured DNS server for the VNet.

Solution

Validate NSG, UDR, and DNS configuration by going through the following steps:

- Check NSGs associated with the application gateway subnet. Ensure that communication to backend isn't blocked.
- Check UDR associated with the application gateway subnet. Ensure that the UDR isn't directing traffic away from the backend subnet. For example, check for routing to network virtual appliances or default routes being advertised to the application gateway subnet via ExpressRoute/VPN.

```
$vnet = Get-AzVirtualNetwork -Name vnetName -ResourceGroupName rgName
Get-AzVirtualNetworkSubnetConfig -Name appGwSubnet -VirtualNetwork $vnet
```

- Check effective NSG and route with the backend VM

```
Get-AzEffectiveNetworkSecurityGroup -NetworkInterfaceName nic1 -ResourceGroupName testrg
Get-AzEffectiveRouteTable -NetworkInterfaceName nic1 -ResourceGroupName testrg
```

- Check presence of custom DNS in the VNet. DNS can be checked by looking at details of the VNet properties in the output.

```
Get-AzVirtualNetwork -Name vnetName -ResourceGroupName rgName
DhcpOptions            : {
                           "DnsServers": [
                             "x.x.x.x"
                           ]
                         }
```

If present, ensure that the DNS server can resolve the backend pool member's FQDN correctly.

Problems with default health probe

Cause

502 errors can also be frequent indicators that the default health probe can't reach back-end VMs.

When an application gateway instance is provisioned, it automatically configures a default health probe to each BackendAddressPool using properties of the BackendHttpSetting. No user input is required to set this probe. Specifically, when a load-balancing rule is configured, an association is made between a BackendHttpSetting and a BackendAddressPool. A default probe is configured for each of these associations and the application gateway starts a periodic health check connection to each instance in the BackendAddressPool at the port specified in the BackendHttpSetting element.

The following table lists the values associated with the default health probe:

PROBE PROPERTY	VALUE	DESCRIPTION
Probe URL	`http://127.0.0.1/`	URL path
Interval	30	Probe interval in seconds
Time-out	30	Probe time-out in seconds
Unhealthy threshold	3	Probe retry count. The back-end server is marked down after the consecutive probe failure count reaches the unhealthy threshold.

Solution

- Host value of the request will be set to 127.0.0.1. Ensure that a default site is configured and is listening at 127.0.0.1.
- Protocol of the request is determined by the BackendHttpSetting protocol.
- URI Path will be set to ⁄.
- If BackendHttpSetting specifies a port other than 80, the default site should be configured to listen at that port.
- The call to `protocol://127.0.0.1:port` should return an HTTP result code of 200. This should be returned within the 30-second timeout period.

- Ensure the configured port is open and there are no firewall rules or Azure Network Security Groups blocking incoming or outgoing traffic on the port configured.
- If Azure classic VMs or Cloud Service is used with a FQDN or a public IP, ensure that the corresponding endpoint is opened.
- If the VM is configured via Azure Resource Manager and is outside the VNet where the application gateway is deployed, a Network Security Group must be configured to allow access on the desired port.

Problems with custom health probe

Cause

Custom health probes allow additional flexibility to the default probing behavior. When you use custom probes, you can configure the probe interval, the URL, the path to test, and how many failed responses to accept before marking the back-end pool instance as unhealthy.

The following additional properties are added:

PROBE PROPERTY	DESCRIPTION
Name	Name of the probe. This name is used to refer to the probe in back-end HTTP settings.
Protocol	Protocol used to send the probe. The probe uses the protocol defined in the back-end HTTP settings
Host	Host name to send the probe. Applicable only when multi-site is configured on the application gateway. This is different from VM host name.
Path	Relative path of the probe. The valid path starts from '/'. The probe is sent to <protocol>://<host>:<port><path>
Interval	Probe interval in seconds. This is the time interval between two consecutive probes.
Time-out	Probe time-out in seconds. If a valid response isn't received within this time-out period, the probe is marked as failed.
Unhealthy threshold	Probe retry count. The back-end server is marked down after the consecutive probe failure count reaches the unhealthy threshold.

Solution

Validate that the Custom Health Probe is configured correctly as the preceding table. In addition to the preceding troubleshooting steps, also ensure the following:

- Ensure that the probe is correctly specified as per the guide.
- If the application gateway is configured for a single site, by default the Host name should be specified as `127.0.0.1`, unless otherwise configured in custom probe.
- Ensure that a call to http://<host>:<port><path> returns an HTTP result code of 200.
- Ensure that Interval, Timeout, and UnhealtyThreshold are within the acceptable ranges.
- If using an HTTPS probe, make sure that the backend server doesn't require SNI by configuring a fallback certificate on the backend server itself.

Request time-out

Cause

When a user request is received, the application gateway applies the configured rules to the request and routes it to a back-end pool instance. It waits for a configurable interval of time for a response from the back-end instance. By default, this interval is `20` seconds. In Application Gateway v1, if the application gateway does not receive a response from back-end application in this interval, the user request gets a 502 error. In Application Gateway v2, if the application gateway does not receive a response from the back-end application in this interval, the request will be tried against a second back-end pool member. If the second request fails the user request gets a 502 error.

Solution

Application Gateway allows you to configure this setting via the BackendHttpSetting, which can be then applied to different pools. Different back-end pools can have different BackendHttpSetting, and a different request time-out configured.

```
    New-AzApplicationGatewayBackendHttpSettings -Name 'Setting01' -Port 80 -Protocol Http -
CookieBasedAffinity Enabled -RequestTimeout 60
```

Empty BackendAddressPool

Cause

If the application gateway has no VMs or virtual machine scale set configured in the back-end address pool, it can't route any customer request and sends a bad gateway error.

Solution

Ensure that the back-end address pool isn't empty. This can be done either via PowerShell, CLI, or portal.

```
Get-AzApplicationGateway -Name "SampleGateway" -ResourceGroupName "ExampleResourceGroup"
```

The output from the preceding cmdlet should contain non-empty back-end address pool. The following example shows two pools returned which are configured with a FQDN or an IP addresses for the backend VMs. The provisioning state of the BackendAddressPool must be 'Succeeded'.

BackendAddressPoolsText:

```
[{
    "BackendAddresses": [{
        "ipAddress": "10.0.0.10",
        "ipAddress": "10.0.0.11"
    }],
    "BackendIpConfigurations": [],
    "ProvisioningState": "Succeeded",
    "Name": "Pool01",
    "Etag": "W/\"00000000-0000-0000-0000-000000000000\"",
    "Id": "/subscriptions/<subscription id>/resourceGroups/<resource group
name>/providers/Microsoft.Network/applicationGateways/<application gateway name>/backendAddressPools/pool01"
}, {
    "BackendAddresses": [{
        "Fqdn": "xyx.cloudapp.net",
        "Fqdn": "abc.cloudapp.net"
    }],
    "BackendIpConfigurations": [],
    "ProvisioningState": "Succeeded",
    "Name": "Pool02",
    "Etag": "W/\"00000000-0000-0000-0000-000000000000\"",
    "Id": "/subscriptions/<subscription id>/resourceGroups/<resource group
name>/providers/Microsoft.Network/applicationGateways/<application gateway name>/backendAddressPools/pool02"
}]
```

Unhealthy instances in BackendAddressPool

Cause

If all the instances of BackendAddressPool are unhealthy, then the application gateway doesn't have any back-end to route user request to. This can also be the case when back-end instances are healthy but don't have the required application deployed.

Solution

Ensure that the instances are healthy and the application is properly configured. Check if the back-end instances can respond to a ping from another VM in the same VNet. If configured with a public end point, ensure a browser request to the web application is serviceable.

Next steps

If the preceding steps don't resolve the issue, open a support ticket.

HTTP response codes in Application Gateway

8/22/2022 • 4 minutes to read • Edit Online

This article lists some HTTP response codes that can be returned by Azure Application Gateway. Common causes and troubleshooting steps are provided to help you determine the root cause. HTTP response codes can be returned to a client request whether or not a connection was initiated to a backend target.

3XX response codes (redirection)

300-399 responses are presented when a client request matches an application gateway rule that has redirects configured. Redirects can be configured on a rule as-is or via a path map rule. For more information about redirects, see Application Gateway redirect overview.

301 Permanent Redirect

HTTP 301 responses are presented when a redirection rule is specified with the **Permanent** value.

302 Found

HTTP 302 responses are presented when a redirection rule is specified with the **Found** value.

303 See Other

HTTP 302 responses are presented when a redirection rule is specified with the **See Other** value.

307 Temporary Redirect

HTTP 307 responses are presented when a redirection rule is specified with the **Temporary** value.

4XX response codes (client error)

400-499 response codes indicate an issue that is initiated from the client. These issues can range from the client initiating requests to an unmatched hostname, request timeout, unauthenticated request, malicious request, and more.

400 – Bad Request

HTTP 400 response codes are commonly observed when:

- Non-HTTP / HTTPS traffic is initiated to an application gateway with an HTTP or HTTPS listener.
- HTTP traffic is initiated to a listener with HTTPS, with no redirection configured.
- Mutual authentication is configured and unable to properly negotiate.

For cases when mutual authentication is configured, several scenarios can lead to an HTTP 400 response being returned the client, such as:

- Client certificate isn't presented, but mutual authentication is enabled.
- DN validation is enabled and the DN of the client certificate doesn't match the DN of the specified certificate chain.
- Client certificate chain doesn't match certificate chain configured in the defined SSL Policy.
- Client certificate is expired.
- OCSP Client Revocation check is enabled and the certificate is revoked.
- OCSP Client Revocation check is enabled, but unable to be contacted.
- OCSP Client Revocation check is enabled, but OCSP responder isn't provided in the certificate.

For more information about troubleshooting mutual authentication, see Error code troubleshooting.

403 – Forbidden

HTTP 403 Forbidden is presented when customers are utilizing WAF skus and have WAF configured in Prevention mode. If enabled WAF rulesets or custom deny WAF rules match the characteristics of an inbound request, the client will be presented a 403 forbidden response.

404 – Page not found

An HTTP 404 response can be returned if a request is sent to an application gateway that is:

- Using a v2 sku.
- Without a hostname match defined in any multi-site listeners.
- Not configured with a basic listener.

408 – Request Timeout

An HTTP 408 response can be observed when client requests to the frontend listener of application gateway do not respond back within 60 seconds. This error can be observed due to traffic congestion between on-premises networks and Azure, when traffic is inspected by virtual appliances, or the client itself becomes overwhelmed.

499 – Client closed the connection

An HTTP 499 response is presented if a client request that is sent to application gateways using v2 sku is closed before the server finished responding. This error can be observed when a large response is returned to the client, but the client may have closed or refreshed their browser/application before the server had a chance to finish responding. In application gateways using v1 sku, an HTTP 0 response code may be raised for the client closing the connection before the server has finished responding as well.

5XX response codes (server error)

500-599 response codes indicate a problem has occurred with application gateway or the backend server while performing the request.

500 – Internal Server Error

Azure Application Gateway shouldn't exhibit 500 response codes. Please open a support request if you see this code, because this issue is an internal error to the service. For information on how to open a support case, see Create an Azure support request.

502 – Bad Gateway

HTTP 502 errors can have several root causes, for example:

- NSG, UDR, or custom DNS is blocking access to backend pool members.
- Back-end VMs or instances of virtual machine scale sets aren't responding to the default health probe.
- Invalid or improper configuration of custom health probes.
- Azure Application Gateway's back-end pool isn't configured or empty.
- None of the VMs or instances in virtual machine scale set are healthy.
- Request time-out or connectivity issues with user requests.

For information about scenarios where 502 errors occur, and how to troubleshoot them, see Troubleshoot Bad Gateway errors.

504 – Request timeout

HTTP 504 errors are presented if a request is sent to application gateways using v2 sku, and the backend response exceeds the time-out value associated to the listener's rule. This value is defined in the HTTP setting.

Next steps

If the information in this article doesn't help to resolve the issue, submit a support ticket.

Troubleshooting mutual authentication errors in Application Gateway

8/22/2022 • 5 minutes to read • Edit Online

Learn how to troubleshoot problems with mutual authentication when using Application Gateway.

Overview

After configuring mutual authentication on an Application Gateway, there can be a number of errors that appear when trying to use mutual authentication. Some common causes for errors include:

- Uploaded a certificate or certificate chain without a root CA certificate
- Uploaded a certificate chain with multiple root CA certificates
- Uploaded a certificate chain that only contained a leaf certificate without a CA certificate
- Validation errors due to issuer DN mismatch

We'll go through different scenarios that you might run into and how to troubleshoot those scenarios. We'll then address error codes and explain likely causes for certain error codes you might be seeing with mutual authentication. All client certificate authentication failures should result in an HTTP 400 error code.

Scenario troubleshooting - configuration problems

There are a few scenarios that you might be facing when trying to configure mutual authentication. We'll walk through how to troubleshoot some of the most common pitfalls.

Self-signed certificate

Problem

The client certificate you uploaded is a self-signed certificate and is resulting in the error code ApplicationGatewayTrustedClientCertificateDoesNotContainAnyCACertificate.

Solution

Double check that the self-signed certificate that you're using has the extension *BasicConstraintsOid* = "2.5.29.19" which indicates the subject can act as a CA. This will ensure that the certificate used is a CA certificate. For more information about how to generate self-signed client certificates, check out trusted client certificates.

Scenario troubleshooting - connectivity problems

You might have been able to configure mutual authentication without any problems but you're running into problems when sending requests to your Application Gateway. We address some common problems and solutions in the following section. You can find the *sslClientVerify* property in the access logs of your Application Gateway.

SslClientVerify is NONE

Problem

The property *sslClientVerify* is appearing as "NONE" in your access logs.

Solution

This is seen when the client doesn't send a client certificate when sending a request to the Application Gateway. This could happen if the client sending the request to the Application Gateway isn't configured correctly to use

client certificates. One way to verify that the client authentication setup on Application Gateway is working as expected is through the following OpenSSL command:

```
openssl s_client -connect <hostname:port> -cert <path-to-certificate> -key <client-private-key-file>
```

The `-cert` flag is the leaf certificate, the `-key` flag is the client private key file.

For more information on how to use the OpenSSL `s_client` command, check out their manual page.

SslClientVerify is FAILED

Problem

The property *sslClientVerify* is appearing as "FAILED" in your access logs.

Solution

There are a number of potential causes for failures in the access logs. Below is a list of common causes for failure:

- **Unable to get issuer certificate:** The issuer certificate of the client certificate couldn't be found. This normally means the trusted client CA certificate chain is not complete on the Application Gateway. Validate that the trusted client CA certificate chain uploaded on the Application Gateway is complete.
- **Unable to get local issuer certificate:** Similar to unable to get issuer certificate, the issuer certificate of the client certificate couldn't be found. This normally means the trusted client CA certificate chain is not complete on the Application Gateway. Validate that the trusted client CA certificate chain uploaded on the Application Gateway is complete.
- **Unable to verify the first certificate:** Unable to verify the client certificate. This error occurs specifically when the client presents only the leaf certificate, whose issuer is not trusted. Validate that the trusted client CA certificate chain uploaded on the Application Gateway is complete.
- **Unable to verify the client certificate issuer:** This error occurs when the configuration *VerifyClientCertIssuerDN* is set to true. This typically happens when the Issuer DN of the client certificate doesn't match the *ClientCertificateIssuerDN* extracted from the trusted client CA certificate chain uploaded by the customer. For more information about how Application Gateway extracts the *ClientCertificateIssuerDN*, check out Application Gateway extracting issuer DN. As best practice, make sure you're uploading one certificate chain per file to Application Gateway.
- **Unsupported certificate purpose:** Ensure the client certificate designates Extended Key Usage for Client Authentication (1.3.6.1.5.5.7.3.2). More details on definition of extended key usage and object identifier for client authentication can be found in RFC 3280 and RFC 5280.

For more information on how to extract the entire trusted client CA certificate chain to upload to Application Gateway, see how to extract trusted client CA certificate chains.

Error code troubleshooting

If you're seeing any of the following error codes, we have a few recommended solutions to help resolve the problem you might be facing.

Error code: ApplicationGatewayTrustedClientCertificateMustSpecifyData

Cause

There is certificate data that is missing. The certificate uploaded could have been an empty file without any certificate data.

Solution

Validate that the certificate file uploaded does not have any missing data.

Error code: ApplicationGatewayTrustedClientCertificateMustNotHavePrivateKey

Cause

There is a private key in the certificate chain. There shouldn't be a private key in the certificate chain.

Solution

Double check the certificate chain that was uploaded and remove the private key that was part of the chain. Reupload the chain without the private key.

Error code: ApplicationGatewayTrustedClientCertificateInvalidData

Cause

There are two potential causes behind this error code.

1. The parsing failed due to the chain not being presented in the right format. Application Gateway expects a certificate chain to be in PEM format and also expects individual certificate data to be delimited.
2. The parser didn't find anything to parse. The file uploaded could potentially only have had the delimiters but no certificate data.

Solution

Depending on the cause of this error, there are two potential solutions.

- Validate that the certificate chain uploaded was in the right format (PEM) and that the certificate data was properly delimited.
- Check that the certificate file uploaded contained the certificate data in addition to the delimiters.

Error code: ApplicationGatewayTrustedClientCertificateDoesNotContainAnyCACertificate

Cause

The certificate uploaded only contained a leaf certificate without a CA certificate. Uploading a certificate chain with CA certificates and a leaf certificate is acceptable as the leaf certificate would just be ignored, but a certificate must have a CA.

Solution

Double check the certificate chain that was uploaded contained more than just the leaf certificate. The *BasicConstraintsOid* = "2.5.29.19" extension should be present and indicate the subject can act as a CA.

Error code: ApplicationGatewayOnlyOneRootCAAllowedInTrustedClientCertificate

Cause

The certificate chain contained multiple root CA certificates *or* contained zero root CA certificates.

Solution

Certificates uploaded must contain exactly one root CA certificate (and however many intermediate CA certificates as needed).

Troubleshoot common questions or issues with Ingress Controller

8/22/2022 • 6 minutes to read • Edit Online

Azure Cloud Shell is the most convenient way to troubleshoot any problems with your AKS and AGIC installation. Launch your shell from shell.azure.com or by clicking the link:

Test with a simple Kubernetes app

The steps below assume:

- You have an AKS cluster, with Advanced Networking enabled
- AGIC has been installed on the AKS cluster
- You already have an Application Gateway on a VNET shared with your AKS cluster

To verify that the Application Gateway + AKS + AGIC installation is set up correctly, deploy the simplest possible app:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-agic-app-pod
  labels:
    app: test-agic-app
spec:
  containers:
  - image: "mcr.microsoft.com/dotnet/core/samples:aspnetapp"
    name: aspnetapp-image
    ports:
    - containerPort: 80
      protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: test-agic-app-service
spec:
  selector:
    app: test-agic-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-agic-app-ingress
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
    - host: test.agic.contoso.com
      http:
        paths:
          - path: /
            backend:
              serviceName: test-agic-app-service
              servicePort: 80
EOF
```

Copy and paste all lines at once from the script above into a Azure Cloud Shell. Verify that the entire command is copied - starting with `cat` and including the last `EOF`.

```
https://shell.azure.com
Azure Cloud Shell
Bash
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

delyan@Azure:~$ cat <<EOF | kubectl apply -f -
> apiVersion: v1
> kind: Pod
> metadata:
>   name: test-agic-app-pod
>   labels:
>     app: test-agic-app
> spec:
>   containers:
>   - image: "mcr.microsoft.com/dotnet/core/samples:aspnetapp"
>     name: aspnetapp-image
>     ports:
>     - containerPort: 80
>       protocol: TCP
> ---
> apiVersion: v1
> kind: Service
> metadata:
>   name: test-agic-app-service
> spec:
>   selector:
>     app: test-agic-app
>   ports:
>   - protocol: TCP
>     port: 80
>     targetPort: 80
> ---
> apiVersion: extensions/v1beta1
> kind: Ingress
> metadata:
>   name: test-agic-app-ingress
>   annotations:
>     kubernetes.io/ingress.class: azure/application-gateway
> spec:
>   rules:
>     - host: test.agic.contoso.com
>       http:
>         paths:
>           - path: /
>             backend:
>               serviceName: test-agic-app-service
>               servicePort: 80
> EOF
pod/test-agic-app-pod created
service/test-agic-app-service created
ingress.extensions/test-agic-app-ingress created
delyan@Azure:~$
```

After a successful deployment of the app above your AKS cluster will have a new Pod, Service and an Ingress.

Get the list of pods with Cloud Shell: `kubectl get pods -o wide`. We expect for a pod named 'test-agic-app-pod' to have been created. It will have an IP address. This address must be within the VNET of the Application Gateway, which is used with AKS.

```
delyan@Azure:~$ kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP          NODE                       NOMINATED NODE   READINESS GATES
ingress-azure-66747895bb-ttpds    1/1     Running   0          18m   10.0.0.31   aks-agentpool-35064155-1   <none>           <none>
mic-67cc97d599-4qnfl              1/1     Running   1          3d    10.0.0.18   aks-agentpool-35064155-1   <none>           <none>
mic-67cc97d599-ptp8l              1/1     Running   0          3d    10.0.0.13   aks-agentpool-35064155-1   <none>           <none>
nmi-d9jqz                         1/1     Running   0          3d    10.0.0.4    aks-agentpool-35064155-1   <none>           <none>
test-agic-app-pod                 1/1     Running   0          34s   10.0.0.33   aks-agentpool-35064155-1   <none>           <none>
```

Get the list of services: `kubectl get services -o wide`. We expect to see a service named 'test-agic-app-service'.

```
delyan@Azure:~$ kubectl get services -o wide
NAME                    TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes              ClusterIP   10.2.0.1      <none>        443/TCP   38d   <none>
test-agic-app-service   ClusterIP   10.2.93.27    <none>        80/TCP    69s   app=test-agic-app
```

Get the list of the ingresses: `kubectl get ingress`. We expect an Ingress resource named 'test-agic-app-ingress' to have been created. The resource will have a host name 'test.agic.contoso.com'.

```
delyan@Azure:~$ kubectl get ingress
NAME                    HOSTS                   ADDRESS        PORTS   AGE
test-agic-app-ingress   test.agic.contoso.com   52.143.73.84   80      108s
```

One of the pods will be AGIC. `kubectl get pods` will show a list of pods, one of which will begin with 'ingress-azure'. Get all logs of that pod with `kubectl logs <name-of-ingress-controller-pod>` to verify that we've had a successful deployment. A successful deployment would have added the following lines to the log:

```
I0927 22:34:51.281437       1 process.go:156] Applied Application Gateway config in 20.461335266s
I0927 22:34:51.281585       1 process.go:165] cache: Updated with latest applied config.
I0927 22:34:51.282342       1 process.go:171] END AppGateway deployment
```

Alternatively, from Cloud Shell we can retrieve only the lines indicating successful Application Gateway configuration with `kubectl logs <ingress-azure-....> | grep 'Applied App Gateway config in'`, where `<ingress-azure....>` should be the exact name of the AGIC pod.

Application Gateway will have the following configuration applied:

- Listener:

- Routing Rule:

- Backend Pool:
  - There will be one IP address in the backend address pool and it will match the IP address of the Pod we observed earlier with `kubectl get pods -o wide`

Finally we can use the `cURL` command from within Cloud Shell to establish an HTTP connection to the newly deployed app:

1. Use `kubectl get ingress` to get the Public IP address of Application Gateway
2. Use `curl -I -H 'test.agic.contoso.com' <publitc-ip-address-from-previous-command>`

A result of `HTTP/1.1 200 OK` indicates that the Application Gateway + AKS + AGIC system is working as expected.

Inspect Kubernetes Installation

Pods, Services, Ingress

Application Gateway Ingress Controller (AGIC) continuously monitors the following Kubernetes resources: Deployment or Pod, Service, Ingress

The following conditions must be in place for AGIC to function as expected:

1. AKS must have one or more healthy **pods**. Verify this configuration from Cloud Shell with `kubectl get pods -o wide --show-labels` If you have a Pod with an `apsnetapp`, your output may look like this:

```
delyan@Azure:~$ kubectl get pods -o wide --show-labels

NAME                    READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES   LABELS
aspnetapp               1/1     Running   0          17h   10.0.0.6    aks-agentpool-35064155-1
<none>           <none>            app=aspnetapp
```

2. One or more **services**, referencing the pods above via matching `selector` labels. Verify this configuration from Cloud Shell with `kubectl get services -o wide`

```
delyan@Azure:~$ kubectl get services -o wide --show-labels

NAME                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR        LABELS
aspnetapp           ClusterIP   10.2.63.254    <none>        80/TCP    17h   app=aspnetapp   <none>
```

3. **Ingress**, annotated with `kubernetes.io/ingress.class: azure/application-gateway`, referencing the service above Verify this configuration from Cloud Shell with `kubectl get ingress -o wide --show-labels`

```
delyan@Azure:~$ kubectl get ingress -o wide --show-labels

NAME        HOSTS   ADDRESS   PORTS   AGE   LABELS
aspnetapp   *                 80      17h   <none>
```

4. View annotations of the ingress above: `kubectl get ingress aspnetapp -o yaml` (substitute `aspnetapp`

with the name of your ingress)

```
delyan@Azure:~$ kubectl get ingress aspnetapp -o yaml

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
  name: aspnetapp
spec:
  backend:
    serviceName: aspnetapp
    servicePort: 80
```

The ingress resource must be annotated with `kubernetes.io/ingress.class: azure/application-gateway`.

Verify Observed Namespace

- Get the existing namespaces in Kubernetes cluster. What namespace is your app running in? Is AGIC watching that namespace? Refer to the Multiple Namespace Support documentation on how to properly configure observed namespaces.

```
# What namespaces exist on your cluster
kubectl get namespaces

# What pods are currently running
kubectl get pods --all-namespaces -o wide
```

- The AGIC pod should be in the `default` namespace (see column `NAMESPACE`). A healthy pod would have `Running` in the `STATUS` column. There should be at least one AGIC pod.

```
# Get a list of the Application Gateway Ingress Controller pods
kubectl get pods --all-namespaces --selector app=ingress-azure
```

- If the AGIC pod isn't healthy (`STATUS` column from the command above isn't `Running`), then:
  - get logs to understand why: `kubectl logs <pod-name>`
  - get logs for the previous instance of the pod: `kubectl logs <pod-name> --previous`
  - describe the pod to get more context: `kubectl describe pod <pod-name>`
- Do you have a Kubernetes Service and Ingress resources?

```
# Get all services across all namespaces
kubectl get service --all-namespaces -o wide

# Get all ingress resources across all namespaces
kubectl get ingress --all-namespaces -o wide
```

- Is your Ingress annotated with: `kubernetes.io/ingress.class: azure/application-gateway`? AGIC will only watch for Kubernetes Ingress resources that have this annotation.

```
# Get the YAML definition of a particular ingress resource
kubectl get ingress --namespace  <which-namespace?>  <which-ingress?>  -o yaml
```

- AGIC emits Kubernetes events for certain critical errors. You can view these events:

- in your terminal via `kubectl get events --sort-by=.metadata.creationTimestamp`
- in your browser using the Kubernetes Web UI (Dashboard)

Logging Levels

AGIC has three logging levels. Level 1 is the default one and it shows minimal number of log lines. Level 5, on the other hand, would display all logs, including sanitized contents of config applied to ARM.

The Kubernetes community has established nine levels of logging for the kubectl tool. In this repository, we're utilizing three of these levels, with similar semantics:

VERBOSITY	DESCRIPTION
1	Default log level; shows startup details, warnings and errors
3	Extended information about events and changes; lists of created objects
5	Logs marshaled objects; shows sanitized JSON config applied to ARM

The verbosity levels are adjustable via the `verbosityLevel` variable in the helm-config.yaml file. Increase verbosity level to `5` to get the JSON config dispatched to ARM:

- add `verbosityLevel: 5` on a line by itself in helm-config.yaml and reinstall
- get logs with `kubectl logs <pod-name>`

Sample Helm config file

```
# This file contains the essential configs for the ingress controller helm chart

# Verbosity level of the App Gateway Ingress Controller
verbosityLevel: 3

################################################################################
# Specify which application gateway the ingress controller will manage
#
appgw:
    subscriptionId: <subscriptionId>
    resourceGroup: <resourceGroupName>
    name: <applicationGatewayName>

    # Setting appgw.shared to "true" will create an AzureIngressProhibitedTarget CRD.
    # This prohibits AGIC from applying config for any host/path.
    # Use "kubectl get AzureIngressProhibitedTargets" to view and change this.
    shared: false

################################################################################
# Specify which kubernetes namespace the ingress controller will watch
# Default value is "default"
# Leaving this variable out or setting it to blank or empty string would
# result in Ingress Controller observing all acessible namespaces.
#
# kubernetes:
#   watchNamespace: <namespace>

################################################################################
# Specify the authentication with Azure Resource Manager
#
# Two authentication methods are available:
# - Option 1: AAD-Pod-Identity (https://github.com/Azure/aad-pod-identity)
armAuth:
    type: aadPodIdentity
    identityResourceID: <identityResourceId>
    identityClientID:  <identityClientId>

## Alternatively you can use Service Principal credentials
# armAuth:
#    type: servicePrincipal
#    secretJSON: <<Generate this value with: "az ad sp create-for-rbac --subscription <subscription-uuid> --
role Contributor --sdk-auth | base64 -w0" >>

################################################################################
# Specify if the cluster is Kubernetes RBAC enabled or not
rbac:
    enabled: false # true/false

# Specify aks cluster related information. THIS IS BEING DEPRECATED.
aksClusterConfiguration:
    apiServerAddress: <aks-api-server-address>
```

Azure Application Gateway Resource Health overview

8/22/2022 • 2 minutes to read • Edit Online

Azure Resource Health helps you diagnose and get support when an Azure service problem affects your resources. It informs you about the current and past health of your resources. And it provides technical support to help you mitigate problems.

For Application Gateway, Resource Health relies on signals emitted by the gateway to assess whether it's healthy or not. If the gateway is unhealthy, Resource Health analyzes additional information to determine the source of the problem. It also identifies actions that Microsoft is taking or what you can do to fix the problem.

For additional details on how health is assessed, review the full list of resource types and health checks in Azure Resource Health.

The health status for Application Gateway is displayed as one of the following statuses:

Available

An **Available** status means the service hasn't detected any events that affect the health of the resource. You'll see the **Recently resolved** notification in cases where the gateway has recovered from unplanned downtime during the last 24 hours.

Unavailable

An **Unavailable** status means the service has detected an ongoing platform or non-platform event that affects the health of the gateway.

Platform events

Platform events are triggered by multiple components of the Azure infrastructure. They include both scheduled actions (for example, planned maintenance) and unexpected incidents (for example, an unplanned host reboot).

Resource Health provides additional details on the event and the recovery process. It also enables you to contact support even if you don't have an active Microsoft support agreement.

Unknown

The **Unknown** health status indicates Resource Health hasn't received information about the gateway for more than 10 minutes. This status isn't a definitive indication of the state of the gateway. But it's an important data point in the troubleshooting process.

If the gateway is running as expected, the status changes to **Available** after a few minutes.

If you're experiencing problems, the **Unknown** health status might suggest that an event in the platform is affecting the gateway.

Degraded

The **Degraded** health status indicates your gateway has detected a loss in performance, although it's still available for usage.

Next steps

To learn about troubleshooting Application Gateway Web Application Firewall (WAF), see Troubleshoot Web Application Firewall (WAF) for Azure Application Gateway.

Use Log Analytics to examine Application Gateway Web Application Firewall (WAF) Logs

8/22/2022 • 2 minutes to read • Edit Online

Once your Application Gateway WAF is operational, you can enable logs to inspect what is happening with each request. Firewall logs give insight to what the WAF is evaluating, matching, and blocking. With Log Analytics, you can examine the data inside the firewall logs to give even more insights. For more information about creating a Log Analytics workspace, see Create a Log Analytics workspace in the Azure portal. For more information about log queries, see Overview of log queries in Azure Monitor.

Import WAF logs

To import your firewall logs into Log Analytics, see Back-end health, diagnostic logs, and metrics for Application Gateway. When you have the firewall logs in your Log Analytics workspace, you can view data, write queries, create visualizations, and add them to your portal dashboard.

Explore data with examples

To view the raw data in the firewall log, you can run the following query:

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
```

This will look similar to the following query:

You can drill down into the data, and plot graphs or create visualizations from here. See the following queries as a starting point:

Matched/Blocked requests by IP

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
| summarize count() by clientIp_s, bin(TimeGenerated, 1m)
| render timechart
```

Matched/Blocked requests by URI

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
| summarize count() by requestUri_s, bin(TimeGenerated, 1m)
| render timechart
```

Top matched rules

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
| summarize count() by ruleId_s, bin(TimeGenerated, 1m)
| where count_ > 10
| render timechart
```

Top five matched rule groups

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
| summarize Count=count() by details_file_s, action_s
| top 5 by Count desc
| render piechart
```

Add to your dashboard

Once you create a query, you can add it to your dashboard. Select the **Pin to dashboard** in the top right of the log analytics workspace. With the previous four queries pinned to an example dashboard, this is the data you can see at a glance:

Next steps

Back-end health, diagnostic logs, and metrics for Application Gateway

Troubleshoot backend health issues in Application Gateway

8/22/2022 • 21 minutes to read • Edit Online

Overview

By default, Azure Application Gateway probes backend servers to check their health status and to check whether they're ready to serve requests. Users can also create custom probes to mention the host name, the path to be probed, and the status codes to be accepted as Healthy. In each case, if the backend server doesn't respond successfully, Application Gateway marks the server as Unhealthy and stops forwarding requests to the server. After the server starts responding successfully, Application Gateway resumes forwarding the requests.

How to check backend health

To check the health of your backend pool, you can use the **Backend Health** page on the Azure portal. Or, you can use Azure PowerShell, CLI, or REST API.

The status retrieved by any of these methods can be any one of the following states:

- Healthy
- Unhealthy
- Unknown

If the backend health status for a server is healthy, it means that Application Gateway will forward the requests to that server. But if the backend health for all the servers in a backend pool is unhealthy or unknown, you might encounter problems when you try to access applications. This article describes the symptoms, cause, and resolution for each of the errors shown.

Backend health status: Unhealthy

If the backend health status is **Unhealthy**, the portal view will resemble the following screenshot:

SERVER (BACKEND POOL)	PORT (HTTP SETTING)	STATUS	DETAILS
10.0.0.5 (appGatewayBackendPool)	80 (appGatewayBackendHttpSettings)	Healthy	Success
10.0.0.6 (appGatewayBackendPool)	80 (appGatewayBackendHttpSettings)	Unhealthy	Cannot connect to server. Check whether any NSG/UDR/F...

Or if you're using an Azure PowerShell, CLI, or Azure REST API query, you'll get a response that resembles the following example:

```
PS C:\Users\testuser\> Get-AzApplicationGatewayBackendHealth -Name "appgw1" -ResourceGroupName "rgOne"
BackendAddressPools :
{Microsoft.Azure.Commands.Network.Models.PSApplicationGatewayBackendHealthPool}
BackendAddressPoolsText : [
{
                              "BackendAddressPool": {
                                "Id": "/subscriptions/536d30b8-665b-40fc-bd7e-
68c65f816365/resourceGroups/rgOne/providers/Microsoft.Network/applicationGateways/appgw1/b
                          ackendAddressPools/appGatewayBackendPool"
                              },
                              "BackendHttpSettingsCollection": [
                                {
                                  "BackendHttpSettings": {
                                    "TrustedRootCertificates": [],
                                    "Id": "/subscriptions/536d30b8-665b-40fc-bd7e-
68c65f816365/resourceGroups/rgOne/providers/Microsoft.Network/applicationGateways/appg
                          w1/backendHttpSettingsCollection/appGatewayBackendHttpSettings"
                                  },
                                  "Servers": [
                                    {
                                      "Address": "10.0.0.5",
                                      "Health": "Healthy"
                                    },
                                    {
                                      "Address": "10.0.0.6",
                                      "Health": "Unhealthy"
                                    }
                                  ]
                                }
                              ]
                            }
                        ]
```

After you receive an unhealthy backend server status for all the servers in a backend pool, requests aren't forwarded to the servers, and Application Gateway returns a "502 Bad Gateway" error to the requesting client. To troubleshoot this issue, check the **Details** column on the **Backend Health** tab.

The message displayed in the **Details** column provides more detailed insights about the issue, and based on those details, you can start troubleshooting the issue.

> **NOTE**
>
> The default probe request is sent in the format of <protocol>://127.0.0.1:<port>. For example, `http://127.0.0.1:80` for an HTTP probe on port 80. Only HTTP status codes of 200 through 399 are considered healthy. The protocol and destination port are inherited from the HTTP settings. If you want Application Gateway to probe on a different protocol, host name, or path and to recognize a different status code as Healthy, configure a custom probe and associate it with the HTTP settings.

Error messages

Backend server timeout

Message: Time taken by the backend to respond to application gateway's health probe is more than the timeout threshold in the probe setting.

Cause: After Application Gateway sends an HTTP(S) probe request to the backend server, it waits for a response from the backend server for a configured period. If the backend server doesn't respond within the configured period (the timeout value), it's marked as Unhealthy until it starts responding within the configured timeout period again.

Resolution: Check why the backend server or application isn't responding within the configured timeout

period, and also check the application dependencies. For example, check whether the database has any issues that might trigger a delay in response. If you're aware of the application's behavior and it should respond only after the timeout value, increase the timeout value from the custom probe settings. You must have a custom probe to change the timeout value. For information about how to configure a custom probe, see the documentation page.

To increase the timeout value, follow these steps:

1. Access the backend server directly and check the time taken for the server to respond on that page. You can use any tool to access the backend server, including a browser using developer tools.
2. After you've figured out the time taken for the application to respond, select the **Health Probes** tab, then select the probe that's associated with your HTTP settings.
3. Enter any timeout value that's greater than the application response time, in seconds.
4. Save the custom probe settings and check whether the backend health shows as Healthy now.

DNS resolution error

Message: Application Gateway could not create a probe for this backend. This usually happens when the FQDN of the backend has not been entered correctly.

Cause: If the backend pool is of type IP Address, FQDN or App Service, Application Gateway resolves to the IP address of the FQDN entered through DNS (custom or Azure default). The application gateway then tries to connect to the server on the TCP port mentioned in the HTTP settings. But if this message is displayed, it suggests that Application Gateway couldn't successfully resolve the IP address of the FQDN entered.

Resolution:

1. Verify that the FQDN entered in the backend pool is correct and that it's a public domain, then try to resolve it from your local machine.
2. If you can resolve the IP address, there might be something wrong with the DNS configuration in the virtual network.
3. Check whether the virtual network is configured with a custom DNS server. If it is, check the DNS server about why it can't resolve to the IP address of the specified FQDN.
4. If you're using Azure default DNS, check with your domain name registrar about whether proper A record or CNAME record mapping has been completed.
5. If the domain is private or internal, try to resolve it from a VM in the same virtual network. If you can resolve it, restart Application Gateway and check again. To restart Application Gateway, you need to stop and start by using the PowerShell commands described in these linked resources.

Updates to the DNS entries of the backend pool

Message: The backend health status could not be retrieved. This happens when an NSG/UDR/Firewall on the application gateway subnet is blocking traffic on ports 65503-65534 in case of v1 SKU, and ports 65200-65535 in case of the v2 SKU or if the FQDN configured in the backend pool could not be resolved to an IP address. To learn more visit - https://aka.ms/UnknownBackendHealth.

Cause: Application Gateway resolves the DNS entries for the backend pool at time of startup and doesn't update them dynamically while running.

Resolution:

Application Gateway must be restarted after any modification to the backend server DNS entries to begin to use the new IP addresses. This operation can be completed via Azure PowerShell or Azure CLI.

Azure PowerShell

```
# Get Azure Application Gateway
$appgw=Get-AzApplicationGateway -Name <appgw_name> -ResourceGroupName <rg_name>

# Stop the Azure Application Gateway
Stop-AzApplicationGateway -ApplicationGateway $appgw

# Start the Azure Application Gateway
Start-AzApplicationGateway -ApplicationGateway $appgw
```

Azure CLI

```
# Stop the Azure Application Gateway
az network application-gateway stop -n <appgw_name> -g <rg_name>

# Start the Azure Application Gateway
az network application-gateway start -n <appgw_name> -g <rg_name>
```

TCP connect error

Message: Application Gateway could not connect to the backend. Check that the backend responds on the port used for the probe. Also check whether any NSG/UDR/Firewall is blocking access to the Ip and port of this backend.

Cause: After the DNS resolution phase, Application Gateway tries to connect to the backend server on the TCP port that's configured in the HTTP settings. If Application Gateway can't establish a TCP session on the port specified, the probe is marked as Unhealthy with this message.

Solution: If you receive this error, follow these steps:

1. Check whether you can connect to the backend server on the port mentioned in the HTTP settings by using a browser or PowerShell. For example, run the following command: `Test-NetConnection -ComputerName www.bing.com -Port 443`.

2. If the port mentioned is not the desired port, enter the correct port number for Application Gateway to connect to the backend server.

3. If you can't connect on the port from your local machine as well, then:

   a. Check the network security group (NSG) settings of the backend server's network adapter and subnet and whether inbound connections to the configured port are allowed. If they aren't, create a new rule to allow the connections. To learn how to create NSG rules, see the documentation page.

   b. Check whether the NSG settings of the Application Gateway subnet allow outbound public and private traffic, so that a connection can be made. Check the document page that's provided in step 3a to learn more about how to create NSG rules.

   ```
   $vnet = Get-AzVirtualNetwork -Name "vnetName" -ResourceGroupName "rgName"
   Get-AzVirtualNetworkSubnetConfig -Name appGwSubnet -VirtualNetwork $vnet
   ```

   c. Check the user-defined routes (UDR) settings of Application Gateway and the backend server's subnet for any routing anomalies. Make sure the UDR isn't directing the traffic away from the backend subnet. For example, check for routes to network virtual appliances or default routes being advertised to the Application Gateway subnet via Azure ExpressRoute and/or VPN.

   d. To check the effective routes and rules for a network adapter, you can use the following PowerShell commands:

```
Get-AzEffectiveNetworkSecurityGroup -NetworkInterfaceName "nic1" -ResourceGroupName "testrg"
Get-AzEffectiveRouteTable -NetworkInterfaceName "nic1" -ResourceGroupName "testrg"
```

4. If you don't find any issues with NSG or UDR, check your backend server for application-related issues that are preventing clients from establishing a TCP session on the ports configured. A few things to check:

   a. Open a command prompt (Win+R -> cmd), enter **netstat**, and select Enter.

   b. Check whether the server is listening on the port that's configured. For example:

```
Proto Local Address Foreign Address State PID
TCP 0.0.0.0:80 0.0.0.0:0 LISTENING 4
```

   c. If it's not listening on the configured port, check your web server settings. For example: site bindings in IIS, server block in NGINX and virtual host in Apache.

   d. Check your OS firewall settings to make sure that incoming traffic to the port is allowed.

HTTP status code mismatch

Message: Status code of the backend's HTTP response did not match the probe setting. Expected: {HTTPStatusCode0} Received:{HTTPStatusCode1}.

Cause: After the TCP connection has been established and a TLS handshake is done (if TLS is enabled), Application Gateway will send the probe as an HTTP GET request to the backend server. As described earlier, the default probe will be to `<protocol>://127.0.0.1:<port>/`, and it considers response status codes in the rage 200 through 399 as Healthy. If the server returns any other status code, it will be marked as Unhealthy with this message.

Solution: Depending on the backend server's response code, you can take the following steps. A few of the common status codes are listed here:

ERROR	ACTIONS
Probe status code mismatch: Received 401	Check whether the backend server requires authentication. Application Gateway probes can't pass credentials for authentication. Either allow "HTTP 401" in a probe status code match or probe to a path where the server doesn't require authentication.
Probe status code mismatch: Received 403	Access forbidden. Check whether access to the path is allowed on the backend server.
Probe status code mismatch: Received 404	Page not found. Check whether the host name path is accessible on the backend server. Change the host name or path parameter to an accessible value.
Probe status code mismatch: Received 405	The probe requests for Application Gateway use the HTTP GET method. Check whether your server allows this method.
Probe status code mismatch: Received 500	Internal server error. Check the backend server's health and whether the services are running.
Probe status code mismatch: Received 503	Service unavailable. Check the backend server's health and whether the services are running.

Or, if you think the response is legitimate and you want Application Gateway to accept other status codes as

Healthy, you can create a custom probe. This approach is useful in situations where the backend website needs authentication. Because the probe requests don't carry any user credentials, they will fail, and an HTTP 401 status code will be returned by the backend server.

To create a custom probe, follow these steps.

HTTP response body mismatch

Message: Body of the backend's HTTP response did not match the probe setting. Received response body does not contain {string}.

Cause: When you create a custom probe, you can mark a backend server as Healthy by matching a string from the response body. For example, you can configure Application Gateway to accept "unauthorized" as a string to match. If the backend server response for the probe request contains the string **unauthorized**, it will be marked as Healthy. Otherwise, it will be marked as Unhealthy with this message.

Solution: To resolve this issue, follow these steps:

1. Access the backend server locally or from a client machine on the probe path, and check the response body.
2. Verify that the response body in the Application Gateway custom probe configuration matches what's configured.
3. If they don't match, change the probe configuration so that it has the correct string value to accept.

Learn more about Application Gateway probe matching.

> **NOTE**
> For all TLS related error messages, to learn more about SNI behavior and differences between the v1 and v2 SKU, check the TLS overview page.

Backend server certificate invalid CA

Message: The server certificate used by the backend is not signed by a well-known Certificate Authority (CA). Allow the backend on the Application Gateway by uploading the root certificate of the server certificate used by the backend.

Cause: End-to-end SSL with Application Gateway v2 requires the backend server's certificate to be verified in order to deem the server Healthy. For a TLS/SSL certificate to be trusted, that certificate of the backend server must be issued by a CA that's included in the trusted store of Application Gateway. If the certificate wasn't issued by a trusted CA (for example, if a self-signed certificate was used), users should upload the issuer's certificate to Application Gateway.

Solution: Follow these steps to export and upload the trusted root certificate to Application Gateway. (These steps are for Windows clients.)

1. Sign in to the machine where your application is hosted.
2. Select Win+R or right-click the **Start** button, and then select **Run**.
3. Enter `certmgr.msc` and select Enter. You can also search for Certificate Manager on the **Start** menu.
4. Locate the certificate, typically in `\Certificates - Current User\\Personal\\Certificates\`, and open it.
5. Select the root certificate and then select **View Certificate**.
6. In the Certificate properties, select the **Details** tab.
7. On the **Details** tab, select the **Copy to File** option and save the file in the Base-64 encoded X.509 (.CER) format.
8. Open the Application Gateway HTTP **Settings** page in the Azure portal.
9. Open the HTTP settings, select **Add Certificate**, and locate the certificate file that you saved.
10. Select **Save** to save the HTTP settings.

Alternatively, you can export the root certificate from a client machine by directly accessing the server (bypassing Application Gateway) through browser and exporting the root certificate from the browser.

For more information about how to extract and upload Trusted Root Certificates in Application Gateway, see Export trusted root certificate (for v2 SKU).

Trusted root certificate mismatch

Message: The root certificate of the server certificate used by the backend does not match the trusted root certificate added to the application gateway. Ensure that you add the correct root certificate to whitelist the backend.

Cause: End-to-end SSL with Application Gateway v2 requires the backend server's certificate to be verified in order to deem the server Healthy. For a TLS/SSL certificate to be trusted, the backend server certificate must be issued by a CA that's included in the trusted store of Application Gateway. If the certificate wasn't issued by a trusted CA (for example, a self-signed certificate was used), users should upload the issuer's certificate to Application Gateway.

The certificate that has been uploaded to Application Gateway HTTP settings must match the root certificate of the backend server certificate.

Solution: If you receive this error message, there's a mismatch between the certificate that has been uploaded to Application Gateway and the one that was uploaded to the backend server.

Follow steps 1-10 in the preceding section to upload the correct trusted root certificate to Application Gateway.

For more information about how to extract and upload Trusted Root Certificates in Application Gateway, see Export trusted root certificate (for v2 SKU).

> **NOTE**
> This error can also occur if the backend server doesn't exchange the complete chain of the cert, including the Root Intermediate (if applicable) Leaf during the TLS handshake. To verify, you can use OpenSSL commands from any client and connect to the backend server by using the configured settings in the Application Gateway probe.

For example:

```
OpenSSL> s_client -connect 10.0.0.4:443 -servername www.example.com -showcerts
```

If the output doesn't show the complete chain of the certificate being returned, export the certificate again with the complete chain, including the root certificate. Configure that certificate on your backend server.

```
  CONNECTED(00000188)\
  depth=0 OU = Domain Control Validated, CN = \*.example.com\
  verify error:num=20:unable to get local issuer certificate\
  verify return:1\
  depth=0 OU = Domain Control Validated, CN = \*.example.com\
  verify error:num=21:unable to verify the first certificate\
  verify return:1\
  \-\-\-\
  Certificate chain\
   0 s:/OU=Domain Control Validated/CN=*.example.com\
     i:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./OU=http://certs.godaddy.com/repository//CN=Go Daddy
Secure Certificate Authority - G2\
  \-----BEGIN CERTIFICATE-----\
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\
  \-----END CERTIFICATE-----
```

Backend certificate invalid common name (CN)

Message: The Common Name (CN) of the backend certificate does not match the host header of the probe.

Cause: Application Gateway checks whether the host name specified in the backend HTTP settings matches that of the CN presented by the backend server's TLS/SSL certificate. This verification is Standard_v2 and WAF_v2 SKU (V2) behavior. The Standard and WAF SKU (v1) Server Name Indication (SNI) is set as the FQDN in the backend pool address. For more information on SNI behavior and differences between v1 and v2 SKU, see Overview of TLS termination and end to end TLS with Application Gateway.

In the v2 SKU, if there's a default probe (no custom probe has been configured and associated), SNI will be set from the host name mentioned in the HTTP settings. Or, if "Pick host name from backend address" is mentioned in the HTTP settings, where the backend address pool contains a valid FQDN, this setting will be applied.

If there's a custom probe associated with the HTTP settings, SNI will be set from the host name mentioned in the custom probe configuration. Or, if **Pick hostname from backend HTTP settings** is selected in the custom probe, SNI will be set from the host name mentioned in the HTTP settings.

If **Pick hostname from backend address** is set in the HTTP settings, the backend address pool must contain a valid FQDN.

If you receive this error message, the CN of the backend certificate doesn't match the host name configured in the custom probe, or the HTTP settings if **Pick hostname from backend HTTP settings** is selected. If you're using a default probe, the host name will be set as **127.0.0.1**. If that's not a desired value, you should create a custom probe and associate it with the HTTP settings.

Solution:

To resolve the issue, follow these steps.

For Windows:

1. Sign in to the machine where your application is hosted.
2. Select Win+R or right-click the **Start** button and select **Run**.
3. Enter **certmgr.msc** and select Enter. You can also search for Certificate Manager on the **Start** menu.
4. Locate the certificate (typically in `\Certificates - Current User\\Personal\\Certificates`), and open the certificate.
5. On the **Details** tab, check the certificate **Subject**.
6. Verify the CN of the certificate from the details and enter the same in the host name field of the custom probe or in the HTTP settings (if **Pick hostname from backend HTTP settings** is selected). If that's not the desired host name for your website, you must get a certificate for that domain or enter the correct host name in the custom probe or HTTP setting configuration.

For Linux using OpenSSL:

1. Run this command in OpenSSL:

   ```
   openssl x509 -in certificate.crt -text -noout
   ```

2. From the properties displayed, find the CN of the certificate and enter the same in the host name field of the http settings. If that's not the desired host name for your website, you must get a certificate for that domain or enter the correct host name in the custom probe or HTTP setting configuration.

Backend certificate is invalid

Message: Backend certificate is invalid. Current date is not within the "Valid from" and "Valid to" date range on the certificate.

Cause: Every certificate comes with a validity range, and the HTTPS connection won't be secure unless the server's TLS/SSL certificate is valid. The current data must be within the **valid from** and **valid to** range. If it's not, the certificate is considered invalid, and that will create a security issue in which Application Gateway marks the backend server as Unhealthy.

Solution: If your TLS/SSL certificate has expired, renew the certificate with your vendor and update the server settings with the new certificate. If it's a self-signed certificate, you must generate a valid certificate and upload the root certificate to the Application Gateway HTTP settings. To do that, follow these steps:

1. Open your Application Gateway HTTP settings in the portal.
2. Select the setting that has the expired certificate, select **Add Certificate**, and open the new certificate file.
3. Remove the old certificate by using the **Delete** icon next to the certificate, and then select **Save**.

Certificate verification failed

Message: The validity of the backend certificate could not be verified. To find out the reason, check OpenSSL diagnostics for the message associated with error code {errorCode}

Cause: This error occurs when Application Gateway can't verify the validity of the certificate.

Solution: To resolve this issue, verify that the certificate on your server was created properly. For example, you can use OpenSSL to verify the certificate and its properties and then try reuploading the certificate to the Application Gateway HTTP settings.

Backend health status: unknown

If the backend health is shown as Unknown, the portal view will resemble the following screenshot:

SERVER (BACKEND POOL)	PORT (HTTP SETTING)	STATUS	DETAILS
10.0.0.5 (appGatewayBackendPool)	443 (testhttpsettings)	Unknown	Unable to retrieve health status data. Check presence of ...
webhook.site (webhookSite)	80 (test)	Unknown	Unable to retrieve health status data. Check presence of ...
52.226.18.28 (testbepool)	80 (test)	Unknown	Unable to retrieve health status data. Check presence of ...

This behavior can occur for one or more of the following reasons:

1. The NSG on the Application Gateway subnet is blocking inbound access to ports 65503-65534 (v1 SKU) or 65200-65535 (v2 SKU) from “Internet."
2. The UDR on the Application Gateway subnet is set to the default route (0.0.0.0/0) and the next hop is not specified as "Internet."
3. The default route is advertised by an ExpressRoute/VPN connection to a virtual network over BGP.
4. The custom DNS server is configured on a virtual network that can't resolve public domain names.
5. Application Gateway is in an Unhealthy state.

Solution:

1. Check whether your NSG is blocking access to the ports 65503-65534 (v1 SKU) or 65200-65535 (v2 SKU) from **Internet**:

   a. On the Application Gateway **Overview** tab, select the **Virtual Network/Subnet** link. b. On the **Subnets** tab of your virtual network, select the subnet where Application Gateway has been deployed. c. Check whether any NSG is configured. d. If an NSG is configured, search for that NSG resource on the **Search** tab or under **All resources**. e. In the **Inbound Rules** section, add an inbound rule to allow destination port range 65503-65534 for v1 SKU or 65200-65535 v2 SKU with the **Source** set as **GatewayManager** service tag. f. Select **Save** and verify that you can view the backend as Healthy. Alternatively, you can do that through PowerShell/CLI.

2. Check whether your UDR has a default route (0.0.0.0/0) with the next hop not set as **Internet**:

a. Follow steps 1a and 1b to determine your subnet. b. Check whether there's any UDR configured. If there is, search for the resource on the search bar or under **All resources**. c. Check whether there are any default routes (0.0.0.0/0) with the next hop not set as **Internet**. If the setting is either **Virtual Appliance** or **Virtual Network Gateway**, you must make sure that your virtual appliance, or the on-premises device, can properly route the packet back to the internet destination without modifying the packet. d. Otherwise, change the next hop to **Internet**, select **Save**, and verify the backend health.

3. Default route advertised by the ExpressRoute/VPN connection to the virtual network over BGP:

   a. If you have an ExpressRoute/VPN connection to the virtual network over BGP, and if you are advertising a default route, you must make sure that the packet is routed back to the internet destination without modifying it. You can verify by using the **Connection Troubleshoot** option in the Application Gateway portal. b. Choose the destination manually as any internet-routable IP address like 1.1.1.1. Set the destination port as anything, and verify the connectivity. c. If the next hop is virtual network gateway, there might be a default route advertised over ExpressRoute or VPN.

4. If there's a custom DNS server configured on the virtual network, verify that the servers can resolve public domains. Public domain name resolution might be required in scenarios where Application Gateway must reach out to external domains like OCSP servers or to check the certificate's revocation status.

5. To verify that Application Gateway is healthy and running, go to the **Resource Health** option in the portal, and verify that the state is **Healthy**. If you see an **Unhealthy** or **Degraded** state, contact support.

6. If Internet and private traffic are going through an Azure Firewall hosted in a secured Virtual hub (using Azure Virtual WAN Hub):

   a. To ensure the application gateway can send traffic directly to the Internet, configure the following user defined route:

   Address prefix: 0.0.0.0/0
   Next hop: Internet

   b. To ensure the application gateway can send traffic to the backend pool via an Azure Firewall in the Virtual WAN hub, configure the following user defined route:

   Address Prefix: Backend pool subnet
   Next hop: Azure Firewall private IP address

Next steps

Learn more about Application Gateway diagnostics and logging.

Common key vault errors in Azure Application Gateway

8/22/2022 • 5 minutes to read • Edit Online

Application Gateway enables customers to securely store TLS certificates in Azure Key Vault. When using a key vault resource, it's important that the gateway always has access to the linked key vault. If your Application Gateway is unable to fetch the certificate, the associated HTTPS listeners will be placed in a disabled state. Learn more.

This article helps you understand the details of the error codes and the steps to resolve such key vault misconfigurations.

> **TIP**
> Use a secret identifier that doesn't specify a version. This way, Azure Application Gateway will automatically rotate the certificate, if a newer version is available in Azure Key Vault. An example of a secret URI without a version is: `https://myvault.vault.azure.net/secrets/mysecret/`.

Azure Advisor error codes

The following sections describe the various errors you might encounter. You can verify if your gateway has any such problem by visiting **Azure Advisor** for your account, and use this troubleshooting article to fix the problem. We recommend configuring Azure Advisor alerts to stay informed when a key vault problem is detected for your gateway.

> **NOTE**
> Azure Application Gateway generates logs for key vault diagnostics every four hours. If the diagnostic continues to show the error after you have fixed the configuration, you might have to wait for the logs to be refreshed.

Error code: UserAssignedIdentityDoesNotHaveGetPermissionOnKeyVault

Description: The associated user-assigned managed identity doesn't have the required permission.

Resolution: Configure the access policies of your key vault to grant the user-assigned managed identity permission on secrets. You may do so in any of the following ways:

Vault access policy

1. Go to the linked key vault in the Azure portal.
2. Open the **Access policies** blade.
3. For **Permission model**, select **Vault access policy**.
4. Under **Secret Management Operations**, select the **Get** permission.
5. Select **Save**.

For more information, see Assign a Key Vault access policy by using the Azure portal.

Azure role-based access control

1. Go to the linked key vault in the Azure portal.
2. Open the **Access policies** blade.
3. For **Permission model**, select **Azure role-based access control**.
4. Navigate to **Access Control (IAM)** blade to configure permissions.
5. **Add role assignment** for your managed identity by choosing the following
   a. **Role**: Key Vault Secrets User
   b. **Assign access to**: Managed identity
   c. **Members**: select the user-assigned managed identity that you've associated with your application gateway.
6. Select **Review + assign**.

For more information, see Azure role-based access control in Key Vault.

> **NOTE**
> Portal support for adding a new key vault-based certificate is currently not available when using **Azure role-based access control**. You can accomplish it by using ARM template, CLI, or PowerShell. Visit this page for guidance.

Error code: SecretDisabled

Description: The associated certificate has been disabled in Key Vault.

Resolution: Re-enable the certificate version that is currently in use for Application Gateway.

1. Go to the linked key vault in the Azure portal.
2. Open the **Certificates** pane.
3. Select the required certificate name, and then select the disabled version.
4. On the management page, use the toggle to enable that certificate version.

Error code: SecretDeletedFromKeyVault

Description: The associated certificate has been deleted from Key Vault.

Resolution: To recover a deleted certificate:

1. Go to the linked key vault in the Azure portal.
2. Open the **Certificates** pane.
3. Use the **Managed deleted certificates** tab to recover a deleted certificate.

On the other hand, if a certificate object is permanently deleted, you'll need to create a new certificate and update Application Gateway with the new certificate details. When you're configuring through the Azure CLI or Azure PowerShell, use a secret identifier URI without a version. This choice allows instances to retrieve a renewed version of the certificate, if it exists.

Error code: UserAssignedManagedIdentityNotFound

Description: The associated user-assigned managed identity has been deleted.

Resolution: Create a new managed identity and use it with the key vault.

1. Re-create a managed identity with the same name that was previously used, and under the same resource group. (**TIP**: Refer to resource Activity Logs for naming details).
2. Go to the desired key vault resource, and set its access policies to grant this new managed identity the required permission. You can follow the same steps as mentioned under UserAssignedIdentityDoesNotHaveGetPermissionOnKeyVault.

Error code: KeyVaultHasRestrictedAccess

Description: There's a restricted network setting for Key Vault.

Resolution: You'll encounter this error when you enable the Key Vault firewall for restricted access. You can still configure Application Gateway in a restricted network of Key Vault, by following these steps:

1. In Key Vault, open the **Networking** pane.
2. Select the **Firewalls and virtual networks** tab, and select **Private endpoint and selected networks**.
3. Then, using Virtual Networks, add your Application Gateway's virtual network and subnet. During the process, also configure 'Microsoft.KeyVault' service endpoint by selecting its checkbox.
4. Finally, select **Yes** to allow Trusted Services to bypass Key Vault's firewall.

Error code: KeyVaultSoftDeleted

Description: The associated key vault is in soft-delete state.

Resolution: In the Azure portal, search for *key vault*. Under **Services**, select **Key vaults**.

Select **Managed deleted vaults**. From here, you can find the deleted Key Vault resource and recover it.

Error code: CustomerKeyVaultSubscriptionDisabled

Description: The subscription for Key Vault is disabled.

Resolution: Your Azure subscription can get disabled for various reasons. To take the necessary action to resolve, see Reactivating a disabled Azure subscription.

Application Gateway Error Codes

Error code: ApplicationGatewayCertificateDataOrKeyVaultSecretIdMustBeSpecified / ApplicationGatewaySslCertificateDataMustBeSpecified

Description: You may encounter this error when trying to update a listener certificate. When this error occurs,

the change to update the certificate will be discarded, and the listener will continue to handle traffic with the previously defined configuration.

Resolution: To resolve this issue, please try uploading the certificate again. For example, the following PowerShell commands may be used to update certificates uploaded to Application Gateway or referenced via Azure Key Vault.

Update certificate uploaded directly to Application Gateway:

```
$appgw = Get-AzApplicationGateway -ResourceGroupName "<ResourceGroup>" -Name "<AppGatewayName>"

$password = ConvertTo-SecureString -String "<password>" -Force -AsPlainText

Set-AzApplicationGatewaySSLCertificate -Name "<oldcertname>" -ApplicationGateway $appgw-CertificateFile "
<newcertPath>" -Password $password

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Update certificate referenced from Azure Key Vault:

```
$appgw = Get-AzApplicationGateway -ResourceGroupName "<ResourceGroup>" -Name "<AppGatewayName>"

$secret = Get-AzKeyVaultSecret -VaultName "<KeyVaultName>" -Name "<CertificateName>"
$secretId = $secret.Id.Replace($secret.Version, "")
$cert = Set-AzApplicationGatewaySslCertificate -ApplicationGateway $AppGW -Name "<CertificateName>"-
KeyVaultSecretId $secretId

Set-AzApplicationGateway -ApplicationGateway $appgw
```

Next steps

These troubleshooting articles might be helpful as you continue to use Application Gateway:

- Understanding and fixing disabled listeners
- Azure Application Gateway Resource Health overview

Understanding disabled listeners

8/22/2022 • 2 minutes to read • Edit Online

The SSL/TLS certificates for Azure Application Gateway's listeners can be referenced from a customer's Key Vault resource. Your application gateway must always have access to such linked key vault resource and its certificate object to ensure smooth operations of the TLS termination feature and the overall health of the gateway resource.

It is important to consider any impact on your Application Gateway resource when making changes or revoking access to your Key Vault resource. In case your application gateway is unable to access the associated key vault or locate its certificate object, it will automatically put that listener in a disabled state. The action is triggered only in the case of configuration errors. Transient connectivity problems do not have any impact on the listeners.

A disabled listener doesn't affect the traffic for other operational listeners on your Application Gateway. For example, the HTTP listeners or HTTPS listeners for which PFX certificate file is directly uploaded on Application Gateway resource will never go in a disabled state.

Periodic check and its impact on listeners

Understanding the behavior of the Application Gateway's periodic check and its potential impact on the state of a key vault-based listener could help you to preempt such occurrences or resolve them much faster.

How does the periodic check work?

1. Application Gateway instances periodically poll the key vault resource to obtain a new certificate version.
2. During this activity, if the instances instead detect a broken access to the key vault resource or a missing certificate object, the listener(s) associated with that key vault will go in a disabled state. The instances are updated with this disabled status of the listener(s) within 60 secs to provide a consistent data plane behavior.
3. After the issue is resolved by the customer, the same four-hour periodic poll verifies the access to key vault certificate object and automatically re-enables listeners on all instances of that gateway.

Ways to identify a disabled listener

1. The clients will observe the error "ERR_SSL_UNRECOGNIZED_NAME_ALERT" if any request is made to a disabled listener of your Application Gateway.

2. You can verify if the error is a result of a disabled listener on your gateway by checking your Application Gateway's Resource Health page. You will see an event as shown below.

Resolving Key Vault configuration errors

You can narrow down to the exact cause and find steps to resolve the problem by visiting the Azure Advisor recommendation in your account.

1. Sign-in to your Azure portal
2. Select Advisor
3. Select Operational Excellence category from the left menu.
4. You will find a recommendation titled **Resolve Azure Key Vault issue for your Application Gateway**, if your gateway is experiencing this issue. Ensure the correct Subscription is selcted from the drop-down options above.
5. Select it to view the error details and the associated key vault resource along with the troubleshooting guide to fix your exact issue.

> **NOTE**
> The disabled listener(s) are automatically enabled if Application Gateway resource detects the underlying problem is resolved. This check occurs every four-hour interval. You can expedite it by performing any minor change to Application Gateway (for HTTP Setting, Resource Tags, etc.) that will force a check against the Key Vault.

Next steps

Troubleshooting key vault errors in Azure Application Gateway

Configure Request and Response Proxy Buffers

8/22/2022 • 2 minutes to read • Edit Online

Azure Application Gateway Standard v2 SKU supports buffering Requests (from clients) or Responses (from the backend servers). Based on the processing capabilities of the clients that interact with your Application Gateway, you can use these buffers to configure the speed of packet delivery.

Response Buffer

Application Gateway's Response buffer can collect all or parts of the response packets sent by the backend server, before delivering them to the clients. By default, the Response buffering is enabled on Application Gateway which is useful to accommodate slow clients. This setting allows you to conserve the backend TCP connections as they can be closed once Application Gateway receives complete response and work according to the client's processing speed. This way, your Application Gateway will continue to deliver the response as per client's pace.

Request Buffer

In a similar way, Application Gateway's Request buffer can temporarily store the entire or parts of the request body, and then forward a larger upload request at once to the backend server. By default, Request buffering setting is enabled on Application Gateway and is useful to offload the processing function of re-assembling the smaller packets of data on the backend server.

> **NOTE**
>
> By default, both Request and Response buffers are enabled on your Application Gateway resource but you can choose to configure them separately. Further, the settings are applied at a resource level and cannot be managed separately for each listener.

You can keep either the Request or Response buffer, enabled or disable, based on your requirements and/or the observed performance of the client systems that communicate with your Application Gateway.

> **WARNING**
>
> We strongly recommend that you test and evaluate the performance before rolling this out on the production gateways.

How to change the buffer setting?

You can change this setting by using GlobalConfiguration in the ARM template as shown below.

```
{
   "$schema":"https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
   "contentVersion":"1.0.0.0",
   "parameters":{
   },
   "variables":{
   },
   "resources":[
      {
         "type":"Microsoft.Network/applicationGateways",
         "apiVersion":"xxx-xx-xx",
         "name":"[parameters('applicationGateways_xxxx_x_xx_name')]",
         "location":"eastus",
         "tags":{
         },
         "identity":{
         },
         "properties":{
            "globalConfiguration":{
               "enableRequestBuffering":false,
               "enableResponseBuffering":false
            }
         }
      }
   ]
}
```

For reference, visit Azure SDK for .NET

Limitations

- API version 2020-01-01 or later should be used to configure buffers.
- Currently, these changes are supported only through ARM templates.
- Request and Response Buffers can only be disabled for the WAF v2 SKU if request body checking is disabled. Otherwise, Request and Response Buffers cannot be disabled for the WAF v2 SKU.

Create Application Gateway custom error pages

8/22/2022 • 3 minutes to read • Edit Online

Application Gateway allows you to create custom error pages instead of displaying default error pages. You can use your own branding and layout using a custom error page.

For example, you can define your own maintenance page if your web application isn't reachable. Or, you can create an unauthorized access page if a malicious request is sent to a web application.

Custom error pages are supported for the following two scenarios:

- **Maintenance page** - This custom error page is sent instead of a 502 bad gateway page. It's shown when Application Gateway has no backend to route traffic to. For example, when there's scheduled maintenance or when an unforeseen issue effects backend pool access.
- **Unauthorized access page** - This custom error page is sent instead of a 403 unauthorized access page. It's shown when the Application Gateway WAF detects malicious traffic and blocks it.

If an error originates from the backend servers, then it's passed along unmodified back to the caller. A custom error page isn't displayed. Application gateway can display a custom error page when a request can't reach the backend.

Custom error pages can be defined at the global level and the listener level:

- **Global level** - the error page applies to traffic for all the web applications deployed on that application gateway.
- **Listener level** - the error page is applied to traffic received on that listener.
- **Both** - the custom error page defined at the listener level overrides the one set at global level.

To create a custom error page, you must have:

- an HTTP response status code.
- corresponding location for the error page.
- error page should be internet accessible and return 200 response.
- error page should be in *.htm or *.html extension type.
- error page size must be less than 1 MB.

You may reference either internal or external images/CSS for this HTML file. For externally referenced resources, use absolute URLs that are publicly accessible. Be aware of the HTML file size when using internal images (Base64-encoded inline image) or CSS. Relative links with files in the same location are currently not supported.

After you specify an error page, the application gateway downloads it from the defined location and saves it to the local application gateway cache. Then, that HTML page is served by the application gateway, whereas the externally referenced resources are fetched directly by the client. To modify an existing custom error page, you must point to a different blob location in the application gateway configuration. The application gateway doesn't periodically check the blob location to fetch new versions.

Portal configuration

1. Navigate to Application Gateway in the portal and choose an application gateway.
2. Select **Listeners** and navigate to a particular listener where you want to specify an error page.
3. Configure a custom error page for a 403 WAF error or a 502 maintenance page at the listener level.

> **NOTE**
> Creating global level custom error pages from the Azure portal is currently not supported.

4. Under **Error page url**, select **Yes**, and then configure a publicly accessible blob URL for a given error status code. Select **Save**. The Application Gateway is now configured with the custom error page.

Azure PowerShell configuration

You can use Azure PowerShell to configure a custom error page. For example, a global custom error page:

```
$appgw   = Get-AzApplicationGateway -Name <app-gateway-name> -ResourceGroupName <resource-group-name>

$updatedgateway = Add-AzApplicationGatewayCustomError -ApplicationGateway $appgw -StatusCode HttpStatus502 -
CustomErrorPageUrl "http://<website-url>"
```

Or a listener level error page:

```
$appgw   = Get-AzApplicationGateway -Name <app-gateway-name> -ResourceGroupName <resource-group-name>

$listener01 = Get-AzApplicationGatewayHttpListener -Name <listener-name> -ApplicationGateway $appgw

$updatedlistener = Add-AzApplicationGatewayHttpListenerCustomError -HttpListener $listener01 -StatusCode
HttpStatus502 -CustomErrorPageUrl "http://<website-url>"
```

For more information, see Add-AzApplicationGatewayCustomError and Add-AzApplicationGatewayHttpListenerCustomError.

Next steps

For information about Application Gateway diagnostics, see Back-end health, diagnostic logs, and metrics for Application Gateway.

Migrate Azure Application Gateway and Web Application Firewall from v1 to v2

8/22/2022 • 10 minutes to read • Edit Online

Azure Application Gateway and Web Application Firewall (WAF) v2 is now available, offering additional features such as autoscaling and availability-zone redundancy. However, existing v1 gateways aren't automatically upgraded to v2. If you want to migrate from v1 to v2, follow the steps in this article.

There are two stages in a migration:

1. Migrate the configuration
2. Migrate the client traffic

This article covers configuration migration. Client traffic migration varies depending on your specific environment. However, some high-level, general recommendations are provided.

Migration overview

An Azure PowerShell script is available that does the following:

- Creates a new Standard_v2 or WAF_v2 gateway in a virtual network subnet that you specify.
- Seamlessly copies the configuration associated with the v1 Standard or WAF gateway to the newly created Standard_V2 or WAF_V2 gateway.

Caveats\Limitations

- The new v2 gateway has new public and private IP addresses. It isn't possible to move the IP addresses associated with the existing v1 gateway seamlessly to v2. However, you can allocate an existing (unallocated) public or private IP address to the new v2 gateway.
- You must provide an IP address space for another subnet within your virtual network where your v1 gateway is located. The script can't create the v2 gateway in any existing subnets that already have a v1 gateway. However, if the existing subnet already has a v2 gateway, that may still work provided there's enough IP address space.
- If you have a network security group or user defined routes associated to the v2 gateway subnet, make sure they adhere to the NSG requirements and UDR requirements for a successful migration
- Virtual network service endpoint policies are currently not supported in an Application Gateway subnet.
- To migrate a TLS/SSL configuration, you must specify all the TLS/SSL certs used in your v1 gateway.
- If you have FIPS mode enabled for your V1 gateway, it won't be migrated to your new v2 gateway. FIPS mode isn't supported in v2.
- v2 doesn't support IPv6, so IPv6 enabled v1 gateways aren't migrated. If you run the script, it may not complete.
- If the v1 gateway has only a private IP address, the script creates a public IP address and a private IP address for the new v2 gateway. v2 gateways currently don't support only private IP addresses.
- Headers with names containing anything other than letters, digits, and hyphens are not passed to your application. This only applies to header names, not header values. This is a breaking change from v1.
- NTLM and Kerberos authentication is not supported by Application Gateway v2. The script is unable to detect if the gateway is serving this type of traffic and may pose as a breaking change from v1 to v2 gateways if run.

Download the script

Download the migration script from the PowerShell Gallery.

Use the script

There are two options for you depending on your local PowerShell environment setup and preferences:

- If you don't have the Azure Az modules installed, or don't mind uninstalling the Azure Az modules, the best option is to use the `Install-Script` option to run the script.
- If you need to keep the Azure Az modules, your best bet is to download the script and run it directly.

To determine if you have the Azure Az modules installed, run `Get-InstalledModule -Name az`. If you don't see any installed Az modules, then you can use the `Install-Script` method.

Install using the Install-Script method

To use this option, you must not have the Azure Az modules installed on your computer. If they're installed, the following command displays an error. You can either uninstall the Azure Az modules, or use the other option to download the script manually and run it.

Run the script with the following command to get the latest version:

```
Install-Script -Name AzureAppGWMigration -Force
```

This command also installs the required Az modules.

Install using the script directly

If you do have some Azure Az modules installed and can't uninstall them (or don't want to uninstall them), you can manually download the script using the **Manual Download** tab in the script download link. The script is downloaded as a raw nupkg file. To install the script from this nupkg file, see Manual Package Download.

To run the script:

1. Use `Connect-AzAccount` to connect to Azure.
2. Use `Import-Module Az` to import the Az modules.
3. Run `Get-Help AzureAppGWMigration.ps1` to examine the required parameters:

   ```
   AzureAppGwMigration.ps1
    -resourceId <v1 application gateway Resource ID>
    -subnetAddressRange <subnet space you want to use>
    -appgwName <string to use to append>
    -AppGwResourceGroupName <resource group name you want to use>
    -sslCertificates <comma-separated SSLCert objects as above>
    -trustedRootCertificates <comma-separated Trusted Root Cert objects as above>
    -privateIpAddress <private IP string>
    -publicIpResourceId <public IP name string>
    -validateMigration -enableAutoScale
   ```

   Parameters for the script:

   - **resourceId: [String]: Required** - This is the Azure Resource ID for your existing Standard v1 or WAF v1 gateway. To find this string value, navigate to the Azure portal, select your application gateway or WAF resource, and click the **Properties** link for the gateway. The Resource ID is located on that page.

     You can also run the following Azure PowerShell commands to get the Resource ID:

```
$appgw = Get-AzApplicationGateway -Name <v1 gateway name> -ResourceGroupName <resource group
Name>
$appgw.Id
```

- **subnetAddressRange: [String]: Required** - This is the IP address space that you've allocated (or want to allocate) for a new subnet that contains your new v2 gateway. This must be specified in the CIDR notation. For example: 10.0.0.0/24. You don't need to create this subnet in advance but the CIDR needs to be part of the VNET address space. The script creates it for you if it doesn't exist and if it exists, it will use the existing one (make sure the subnet is either empty, contains only v2 Gateway if any, and has enough available IPs).

- **appgwName: [String]: Optional**. This is a string you specify to use as the name for the new Standard_v2 or WAF_v2 gateway. If this parameter isn't supplied, the name of your existing v1 gateway will be used with the suffix *_v2* appended.

- **AppGwResourceGroupName: [String]: Optional**. Name of resource group where you want v2 Application Gateway resources to be created (default value will be `<v1-app-gw-rgname>`)

- **sslCertificates: [PSApplicationGatewaySslCertificate]: Optional**. A comma-separated list of PSApplicationGatewaySslCertificate objects that you create to represent the TLS/SSL certs from your v1 gateway must be uploaded to the new v2 gateway. For each of your TLS/SSL certs configured for your Standard v1 or WAF v1 gateway, you can create a new PSApplicationGatewaySslCertificate object via the `New-AzApplicationGatewaySslCertificate` command shown here. You need the path to your TLS/SSL Cert file and the password.

  This parameter is only optional if you don't have HTTPS listeners configured for your v1 gateway or WAF. If you have at least one HTTPS listener setup, you must specify this parameter.

  ```
  $password = ConvertTo-SecureString <cert-password> -AsPlainText -Force
  $mySslCert1 = New-AzApplicationGatewaySslCertificate -Name "Cert01" `
     -CertificateFile <Cert-File-Path-1> `
     -Password $password
  $mySslCert2 = New-AzApplicationGatewaySslCertificate -Name "Cert02" `
     -CertificateFile <Cert-File-Path-2> `
     -Password $password
  ```

  You can pass in `$mySslCert1, $mySslCert2` (comma-separated) in the previous example as values for this parameter in the script.

- **trustedRootCertificates: [PSApplicationGatewayTrustedRootCertificate]: Optional**. A comma-separated list of PSApplicationGatewayTrustedRootCertificate objects that you create to represent the Trusted Root certificates for authentication of your backend instances from your v2 gateway.

  ```
  $certFilePath = ".\rootCA.cer"
  $trustedCert = New-AzApplicationGatewayTrustedRootCertificate -Name "trustedCert1" -
  CertificateFile $certFilePath
  ```

  To create a list of PSApplicationGatewayTrustedRootCertificate objects, see New-AzApplicationGatewayTrustedRootCertificate.

- **privateIpAddress: [String]: Optional**. A specific private IP address that you want to associate to your new v2 gateway. This must be from the same VNet that you allocate for your new v2 gateway. If this isn't specified, the script allocates a private IP address for your v2 gateway.

- **publicIpResourceId: [String]: Optional**. The resourceId of existing public IP address (standard

SKU) resource in your subscription that you want to allocate to the new v2 gateway. If this isn't specified, the script allocates a new public IP in the same resource group. The name is the v2 gateway's name with *-IP* appended.

- **validateMigration: [switch]: Optional**. Use this parameter if you want the script to do some basic configuration comparison validations after the v2 gateway creation and the configuration copy. By default, no validation is done.

- **enableAutoScale: [switch]: Optional**. Use this parameter if you want the script to enable AutoScaling on the new v2 gateway after it's created. By default, AutoScaling is disabled. You can always manually enable it later on the newly created v2 gateway.

4. Run the script using the appropriate parameters. It may take five to seven minutes to finish.

   Example

```
AzureAppGWMigration.ps1 `
   -resourceId /subscriptions/8b1d0fea-8d57-4975-adfb-
308f1f4d12aa/resourceGroups/MyResourceGroup/providers/Microsoft.Network/applicationGateways/myv1appga
teway `
   -subnetAddressRange 10.0.0.0/24 `
   -appgwname "MynewV2gw" `
   -AppGwResourceGroupName "MyResourceGroup" `
   -sslCertificates $mySslCert1,$mySslCert2 `
   -trustedRootCertificates $trustedCert `
   -privateIpAddress "10.0.0.1" `
   -publicIpResourceId "/subscriptions/8b1d0fea-8d57-4975-adfb-
308f1f4d12aa/resourceGroups/MyResourceGroup/providers/Microsoft.Network/publicIPAddresses/MyPublicIP"
`
   -validateMigration -enableAutoScale
```

Migrate client traffic

First, double check that the script successfully created a new v2 gateway with the exact configuration migrated over from your v1 gateway. You can verify this from the Azure portal.

Also, send a small amount of traffic through the v2 gateway as a manual test.

Here are a few scenarios where your current application gateway (Standard) may receive client traffic, and our recommendations for each one:

- **A custom DNS zone (for example, contoso.com) that points to the frontend IP address (using an A record) associated with your Standard v1 or WAF v1 gateway**.

  You can update your DNS record to point to the frontend IP or DNS label associated with your Standard_v2 application gateway. Depending on the TTL configured on your DNS record, it may take a while for all your client traffic to migrate to your new v2 gateway.

- **A custom DNS zone (for example, contoso.com) that points to the DNS label (for example: *myappgw.eastus.cloudapp.azure.com* using a CNAME record) associated with your v1 gateway**.

  You have two choices:

  - If you use public IP addresses on your application gateway, you can do a controlled, granular migration using a Traffic Manager profile to incrementally route traffic (weighted traffic routing method) to the new v2 gateway.

    You can do this by adding the DNS labels of both the v1 and v2 application gateways to the Traffic Manager profile, and CNAMEing your custom DNS record (for example, `www.contoso.com`) to the

Traffic Manager domain (for example, contoso.trafficmanager.net).

  - Or, you can update your custom domain DNS record to point to the DNS label of the new v2 application gateway. Depending on the TTL configured on your DNS record, it may take a while for all your client traffic to migrate to your new v2 gateway.

- **Your clients connect to the frontend IP address of your application gateway**.

  Update your clients to use the IP address(es) associated with the newly created v2 application gateway. We recommend that you don't use IP addresses directly. Consider using the DNS name label (for example, yourgateway.eastus.cloudapp.azure.com) associated with your application gateway that you can CNAME to your own custom DNS zone (for example, contoso.com).

Common questions

Are there any limitations with the Azure PowerShell script to migrate the configuration from v1 to v2?

Yes. See Caveats/Limitations.

Is this article and the Azure PowerShell script applicable for Application Gateway WAF product as well?

Yes.

Does the Azure PowerShell script also switch over the traffic from my v1 gateway to the newly created v2 gateway?

No. The Azure PowerShell script only migrates the configuration. Actual traffic migration is your responsibility and in your control.

Is the new v2 gateway created by the Azure PowerShell script sized appropriately to handle all of the traffic that is currently served by my v1 gateway?

The Azure PowerShell script creates a new v2 gateway with an appropriate size to handle the traffic on your existing v1 gateway. Autoscaling is disabled by default, but you can enable AutoScaling when you run the script.

I configured my v1 gateway to send logs to Azure storage. Does the script replicate this configuration for v2 as well?

No. The script doesn't replicate this configuration for v2. You must add the log configuration separately to the migrated v2 gateway.

Does this script support certificates uploaded to Azure KeyVault ?

No. Currently the script does not support certificates in KeyVault. However, this is being considered for a future version.

I ran into some issues with using this script. How can I get help?

You can contact Azure Support under the topic "Configuration and Setup/Migrate to V2 SKU". Learn more about Azure support here.

Next steps

Learn about Application Gateway v2

Configure Azure Monitor alerts for Application Gateway

8/22/2022 • 2 minutes to read • Edit Online

Azure Monitor alerts proactively notify you when important conditions are found in your monitoring data. They allow you to identify and address issues in your system before your customers notice them. For more information about Azure Monitor Alerts for Application Gateway, see Monitoring Azure Application Gateway.

Configure alerts using ARM templates

You can use ARM templates to quickly configure important alerts for Application Gateway. Before you begin, consider the following details:

- Azure Monitor alert rules are charged based on the type and number of signals it monitors. See Azure Monitor pricing before deploying for pricing information. Or you can see the estimated cost in the portal after deployment:

- You need to create an Azure Monitor action group in advance and then use the Resource ID for as many alerts as you need. Azure Monitor alerts use this action group to notify users that an alert has been triggered. For more information, see Create and manage action groups in the Azure portal.

TIP

You can manually form the ResourceID for your Action Group by following these steps.

1. Select Azure Monitor in your Azure portal.
2. Open the Alerts page and select Action Groups.
3. Select the action group to view its details.
4. Use the Resource Group Name, Action Group Name and Subscription Info here to form the ResourceID for the action group as shown here:

```
/subscriptions/<subscription-id-from-your-account>/resourcegroups/<resource-group-name>/providers/microsoft.insights/actiongroups/<action-group-name>
```

- The templates for alerts described here are defined generically for settings like Severity, Aggregation Granularity, Frequency of Evaluation, Condition Type, and so on. You can modify the settings after deployment to meet your needs. See Understand how metric alerts work in Azure Monitor for more information.
- The templates for metric-based alerts use the **Dynamic threshold** value with High sensitivity. You can choose to adjust these settings based on your needs.

ARM templates

The following ARM templates are available to configure Azure Monitor alerts for Application Gateway.

Alert for Backend Response Status as 5xx

Deploy to Azure

This notification is based on Metrics signal.

Alert for average Unhealthy Host Count

Deploy to Azure

This notification is based on Metrics signal.

Alert for Backend Last Byte Response Time

Deploy to Azure

This notification is based on Metrics signal.

Alert for Key Vault integration issues

Deploy to Azure

This notification is based on its Azure Advisor recommendation.

Next steps

- See Monitoring Application Gateway data reference for a reference of the metrics, logs, and other important values created by Application Gateway.
- See Monitoring Azure resources with Azure Monitor for details on monitoring Azure resources.

Application Gateway classic to Resource Manager migration

8/22/2022 • 2 minutes to read • Edit Online

Resource Manager enables deploying complex applications through templates, configures virtual machines by using VM extensions, and incorporates access management and tagging. Azure Resource Manager includes scalable, parallel deployment for virtual machines into availability sets. The new deployment model also provides lifecycle management of compute, network, and storage independently. You can read more about Azure Resource Manager features and benefits.

Application Gateway resources will **not** be migrated automatically as part of VNet migration from classic to Resource Manager. As part of VNet migration process as documented at IaaS resources migration page, if you have an Application Gateway resource present on the VNet that you're trying to migrate to Resource Manager deployment model, the automatic migration wouldn't be successful.

In order to migrate your Application Gateway resource to Resource Manager deployment model, you'll have to remove the Application Resource from the VNet before beginning migration and then recreate the Application Gateway resource once migration is complete.

Creating a new Application Gateway resource

For more information on how to set up an Application Gateway resource after VNet migration, you can refer:

- Deployment via portal
- Deployment via PowerShell
- Deployment via Azure CLI
- Deployment via ARM template

Common questions

What is Azure Service Manager and what does it mean by classic?

The word "classic" in classic networking service refers to networking resources managed by Azure Service Manager (ASM). Azure Service Manager (ASM) is the old control plane of Azure responsible for creating, managing, deleting VMs and performing other control plane operations.

What is Azure Resource Manager?

Azure Resource Manager is the latest control plane of Azure responsible for creating, managing, deleting VMs and performing other control plane operations.

Where can I find more information regarding classic to Azure Resource Manager migration?

Please refer to Frequently asked questions about classic to Azure Resource Manager migration

How do I report an issue?

Post your issues and questions about migration to our Microsoft Q&A page. We recommend posting all your questions on this forum. If you have a support contract, you're welcome to log a support ticket as well.

Next steps

To get started see: platform-supported migration of IaaS resources from classic to Resource Manager

For any concerns around migration, you can contact Azure Support. Learn more about Azure support here.

Configure Azure Application Gateway Private Link (preview)

8/22/2022 • 5 minutes to read • Edit Online

Application Gateway Private Link allows you to connect your workloads over a private connection spanning across VNets and subscriptions. For more information, see Application Gateway Private Link.

> **IMPORTANT**
>
> Azure Application Gateway Private Link is currently in public preview.

Configuration options

Application Gateway Private Link can be configured via multiple options, such as, but not limited to, the Azure portal, Azure PowerShell, and Azure CLI.

- Azure portal
- Azure PowerShell
- Azure CLI

Define a subnet for Private Link Configuration

To enable Private Link Configuration, a subnet, different from the Application Gateway subnet, is required for the private link IP configuration. Private Link must use a subnet that doesn't contain any Application Gateways. Subnet sizing can be determined by the number of connections required for your deployment. Each IP address allocated to this subnet ensures 64-K concurrent TCP connections that can be established via Private Link at single point in time. Allocate more IP addresses to allow more connections via Private Link. For example: `n * 64K`; where `n` is the number of IP addresses being provisioned.

> **NOTE**
>
> The maximum number of IP addresses per private link configuration is eight. Only dynamic allocation is supported.

The following steps can be completed to create a new subnet:

Add, change, or delete a virtual network subnet

Configure Private Link

The Private link configuration defines the infrastructure used by Application Gateway to enable connections from Private Endpoints. To create the Private link configuration, complete the following steps:

1. Go to the Azure portal
2. Search for and select **Application Gateways**.
3. Select the name of the application gateway you want to enable private link.
4. Select **Private link**
5. Configure the following items:
   - **Name**: The name of the private link configuration.
   - **Private link subnet**: The subnet IP addresses should be consumed from.
   - **Frontend IP Configuration**: The frontend IP address that private link should forward traffic to on Application Gateway.
   - **Private IP address settings**: specify at least one IP address
6. Select **Add**.
7. Within your **Application Gateways** properties blade, obtain and make a note of the **Resource ID**, you will require this if setting up a Private Endpoint within a diffrerent Azure AD tenant

Configure Private Endpoint

A private endpoint is a network interface that uses a private IP address from the virtual network containing clients wishing to connect to your gateway. Each of the clients will use the private IP address of the Private Endpoint to tunnel traffic to the Application Gateway. To create a private endpoint, complete the following steps:

1. Select the **Private endpoint connections** tab.
2. Select **Create**.
3. On the **Basics** tab, configure a resource group, name, and region for the Private Endpoint. Select **Next**.
4. On the **Resource** tab, select **Next**.
5. On the **Virtual Network** tab, configure a virtual network and subnet where the private endpoint network interface should be provisioned to. Configure whether the private endpoint should have a dynamic or static IP address. Select **Next**.
6. On the **Tags** tab, optionally configure resource tags. Select **Next**.
7. Select **Create**.

> **NOTE**
> If the public or private IP configuration resource is missing when trying to select a *Target sub-resource* on the *Resource* tab of private endpoint creation, please ensure a listener is actively utilizing the respected frontend IP configuration. Frontend IP configurations without an associated listener will not be shown as a *Target sub-resource*.

> **NOTE**
> If you are setting up the **Private Endpoint** from within another tenant, you will need to utilise the Azure Application Gateway Resource ID, along with sub-resource as either *appGwPublicFrontendIp* or *appGwPrivateFrontendIp*, depending upon your Azure Application Gateway Private Link Frontend IP Configuration.

Next steps

- Learn about Azure Private Link: What is Azure Private Link?

Monitoring Azure Application Gateway data reference

8/22/2022 • 10 minutes to read • Edit Online

See Monitoring Azure Application Gateway for details on collecting and analyzing monitoring data for Azure Application Gateway.

Metrics

Application Gateway v2 metrics

Resource Provider and Type: Microsoft.Network/applicationGateways

Timing metrics

Application Gateway provides several built-in timing metrics related to the request and response, which are all measured in milliseconds.

> **NOTE**
>
> If the Application Gateway has more than one listener, then always filter by the *Listener* dimension while comparing different latency metrics to get more meaningful inference.

METRIC	UNIT	DESCRIPTION
Backend connect time	Milliseconds	Time spent establishing a connection with the backend application. This includes the network latency and the time taken by the backend server's TCP stack to establish new connections. For TLS, it also includes the time spent on handshake.
Backend first byte response time	Milliseconds	Time interval between start of establishing a connection to backend server and receiving the first byte of the response header. This approximates the sum of Backend connect time, time taken by the request to reach the backend from Application Gateway, time taken by backend application to respond (the time the server took to generate content, potentially fetch database queries), and the time taken by first byte of the response to reach the Application Gateway from the backend.

METRIC	UNIT	DESCRIPTION
Backend last byte response time	Milliseconds	Time interval between start of establishing a connection to backend server and receiving the last byte of the response body. This approximates the sum of backend first byte response time and data transfer time. This number may vary greatly depending on the size of objects requested and the latency of the server network.
Application gateway total time	Milliseconds	Average time that it takes for a request to be received, processed and its response to be sent. This is the interval from the time when Application Gateway receives the first byte of the HTTP request to the time when the last response byte has been sent to the client. This includes the processing time taken by Application Gateway, the Backend last byte response time, the time taken by Application Gateway to send all the response, and the Client RTT.
Client RTT	Milliseconds	Average round-trip time between clients and Application Gateway.

These metrics can be used to determine whether the observed slowdown is due to the client network, Application Gateway performance, the backend network and backend server TCP stack saturation, backend application performance, or large file size.

For example, if there's a spike in *Backend first byte response time* trend but the *Backend connect time* trend is stable, then it can be inferred that the Application gateway to backend latency and the time taken to establish the connection is stable, and the spike is caused due to an increase in the response time of backend application. On the other hand, if the spike in *Backend first byte response time* is associated with a corresponding spike in *Backend connect time*, then it can be deduced that either the network between Application Gateway and backend server or the backend server TCP stack has saturated.

If you notice a spike in *Backend last byte response time* but the *Backend first byte response time* is stable, then it can be deduced that the spike is because of a larger file being requested.

Similarly, if the *Application gateway total time* has a spike but the *Backend last byte response time* is stable, then it can either be a sign of performance bottleneck at the Application Gateway or a bottleneck in the network between client and Application Gateway. Additionally, if the *client RTT* also has a corresponding spike, then it indicates that the degradation is because of the network between client and Application Gateway.

Application Gateway metrics

METRIC	UNIT	DESCRIPTION
Bytes received	Bytes	Count of bytes received by the Application Gateway from the clients.

METRIC	UNIT	DESCRIPTION
Bytes sent	Bytes	Count of bytes sent by the Application Gateway to the clients.
Client TLS protocol	Count	Count of TLS and non-TLS requests initiated by the client that established connection with the Application Gateway. To view TLS protocol distribution, filter by the TLS Protocol dimension.
Current capacity units	Count	Count of capacity units consumed to load balance the traffic. There are three determinants to capacity unit - compute unit, persistent connections, and throughput. Each capacity unit is composed of at most: one compute unit, or 2500 persistent connections, or 2.22-Mbps throughput.
Current compute units	Count	Count of processor capacity consumed. Factors affecting compute unit are TLS connections/sec, URL Rewrite computations, and WAF rule processing.
Current connections	Count	The total number of concurrent connections active from clients to the Application Gateway.
Estimated Billed Capacity units	Count	With the v2 SKU, the pricing model is driven by consumption. Capacity units measure consumption-based cost that is charged in addition to the fixed cost. *Estimated Billed Capacity units indicate the number of capacity units using which the billing is estimated. This is calculated as the greater value between *Current capacity units* (capacity units required to load balance the traffic) and *Fixed billable capacity units* (minimum capacity units kept provisioned).
Failed Requests	Count	Number of requests that Application Gateway has served with 5xx server error codes. This includes the 5xx codes that are generated from the Application Gateway and the 5xx codes that are generated from the backend. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

METRIC	UNIT	DESCRIPTION
Fixed Billable Capacity Units	Count	The minimum number of capacity units kept provisioned as per the *Minimum scale units* setting (one instance translates to 10 capacity units) in the Application Gateway configuration.
New connections per second	Count	The average number of new TCP connections per second established from clients to the Application Gateway and from the Application Gateway to the backend members.
Response Status	Status code	HTTP response status returned by Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.
Throughput	Bytes/sec	Number of bytes per second the Application Gateway has served.
Total Requests	Count	Count of successful requests that Application Gateway has served. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

Backend metrics

METRIC	UNIT	DESCRIPTION
Backend response status	Count	Count of HTTP response status codes returned by the backends. This doesn't include any response codes generated by the Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.
Healthy host count	Count	The number of backends that are determined healthy by the health probe. You can filter on a per backend pool basis to show the number of healthy hosts in a specific backend pool.
Unhealthy host count	Count	The number of backends that are determined unhealthy by the health probe. You can filter on a per backend pool basis to show the number of unhealthy hosts in a specific backend pool.

METRIC	UNIT	DESCRIPTION
Requests per minute per Healthy Host	Count	The average number of requests received by each healthy member in a backend pool in a minute. Specify the backend pool using the *BackendPool HttpSettings* dimension.

Application Gateway v1 metrics

Application Gateway metrics

METRIC	UNIT	DESCRIPTION
CPU Utilization	Percent	Displays the CPU usage allocated to the Application Gateway. Under normal conditions, CPU usage should not regularly exceed 90%, as this may cause latency in the websites hosted behind the Application Gateway and disrupt the client experience. You can indirectly control or improve CPU usage by modifying the configuration of the Application Gateway by increasing the instance count or by moving to a larger SKU size, or doing both.
Current connections	Count	Count of current connections established with Application Gateway.
Failed Requests	Count	Number of requests that failed because of connection issues. This count includes requests that failed due to exceeding the *Request time-out* HTTP setting and requests that failed due to connection issues between Application Gateway and the backend. This count doesn't include failures due to no healthy backend being available. 4xx and 5xx responses from the backend are also not considered as part of this metric.
Response Status	Status code	HTTP response status returned by Application Gateway. The response status code distribution can be further categorized to show responses in 2xx, 3xx, 4xx, and 5xx categories.
Throughput	Bytes/sec	Number of bytes per second the Application Gateway has served.
Total Requests	Count	Count of successful requests that Application Gateway has served. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

METRIC	UNIT	DESCRIPTION
Web Application Firewall Blocked Requests Count	Count	Number of requests blocked by WAF.
Web Application Firewall Blocked Requests Distribution	Count	Number of requests blocked by WAF filtered to show count per each/specific WAF rule group or WAF rule ID combination.
Web Application Firewall Total Rule Distribution	Count	Number of requests received per each specific WAF rule group or WAF rule ID combination.

For more information, see a list of all platform metrics supported in Azure Monitor.

Metric Dimensions

For more information on what metric dimensions are, see Multi-dimensional metrics.

Azure Application Gateway supports dimensions for some of the metrics in Azure Monitor. Each metric includes a description that explains the available dimensions specifically for that metric.

Resource logs

This section lists the types of resource logs you can collect for Azure Application Gateway.

For reference, see a list of all resource logs category types supported in Azure Monitor.

> **NOTE**
> The Performance log is available only for the v1 SKU. For the v2 SKU, use Metrics for performance data.

For more information, see Back-end health and diagnostic logs for Application Gateway

Application Gateway

Resource Provider and Type: Microsoft.Network/applicationGateways

CATEGORY	DISPLAY NAME	INFORMATION
Activitylog	Activity log	Activity log entries are collected by default. You can use Azure activity logs (formerly known as operational logs and audit logs) to view all operations that are submitted to your Azure subscription, and their status.

CATEGORY	DISPLAY NAME	INFORMATION
ApplicationGatewayAccessLog	Access log	You can use this log to view Application Gateway access patterns and analyze important information. This includes the caller's IP address, requested URL, response latency, return code, and bytes in and out. An access log is collected every 60 seconds. This log contains one record per instance of Application Gateway. The Application Gateway instance is identified by the instanceId property.
ApplicationGatewayPerformanceLog	Performance log	You can use this log to view how Application Gateway instances are performing. This log captures performance information for each instance, including total requests served, throughput in bytes, total requests served, failed request count, and healthy and unhealthy back-end instance count. A performance log is collected every 60 seconds. The Performance log is available only for the v1 SKU. For the v2 SKU, use Metrics for performance data.
ApplicationGatewayFirewallLog	Firewall log	You can use this log to view the requests that are logged through either detection or prevention mode of an application gateway that is configured with the web application firewall. Firewall logs are collected every 60 seconds.

Azure Monitor Logs tables

This section refers to all of the Azure Monitor Logs Kusto tables relevant to Azure Application Gateway and available for query by Log Analytics.

RESOURCE TYPE	NOTES
Application Gateway	Includes AzureActivity, AzureDiagnostics, and AzureMetrics

For a reference of all Azure Monitor Logs / Log Analytics tables, see the Azure Monitor Log Table Reference.

Diagnostics tables

Azure Application Gateway uses the Azure Diagnostics table to store resource log information. The following columns are relevant.

Azure Diagnostics

PROPERTY	DESCRIPTION
requestUri_s	The URI of the client request.
Message	Informational messages such as "SQL Injection Attack"

PROPERTY	DESCRIPTION
userAgent_s	User agent details of the client request
ruleName_s	Request routing rule that is used to serve this request
httpMethod_s	HTTP method of the client request
instanceId_s	The Appgw instance to which the client request is routed to for evaluation
httpVersion_s	HTTP version of the client request
clientIP_s	IP from which is request is made
host_s	Host header of the client request
requestQuery_s	Query string as part of the client request
sslEnabled_s	Does the client request have SSL enabled

See Also

- See Monitoring Azure Azure Application Gateway for a description of monitoring Azure Application Gateway.
- See Monitoring Azure resources with Azure Monitor for details on monitoring Azure resources.

Annotations for Application Gateway Ingress Controller

8/22/2022 • 4 minutes to read • Edit Online

Introductions

The Kubernetes Ingress resource can be annotated with arbitrary key/value pairs. AGIC relies on annotations to program Application Gateway features, which are not configurable via the Ingress YAML. Ingress annotations are applied to all HTTP setting, backend pools, and listeners derived from an ingress resource.

List of supported annotations

For an Ingress resource to be observed by AGIC, it **must be annotated** with `kubernetes.io/ingress.class: azure/application-gateway`. Only then AGIC will work with the Ingress resource in question.

ANNOTATION KEY	VALUE TYPE	DEFAULT VALUE	ALLOWED VALUES
appgw.ingress.kubernetes.io/backend-path-prefix	`string`	`nil`	
appgw.ingress.kubernetes.io/ssl-redirect	`bool`	`false`	
appgw.ingress.kubernetes.io/connection-draining	`bool`	`false`	
appgw.ingress.kubernetes.io/connection-draining-timeout	`int32` (seconds)	`30`	
appgw.ingress.kubernetes.io/cookie-based-affinity	`bool`	`false`	
appgw.ingress.kubernetes.io/request-timeout	`int32` (seconds)	`30`	
appgw.ingress.kubernetes.io/use-private-ip	`bool`	`false`	
appgw.ingress.kubernetes.io/backend-protocol	`string`	`http`	`http`, `https`
appgw.ingress.kubernetes.io/rewrite-rule-set	`string`	`nil`	

Backend Path Prefix

This annotation allows the backend path specified in an ingress resource to be rewritten with prefix specified in this annotation. This allows users to expose services whose endpoints are different than endpoint names used to

expose a service in an ingress resource.

Usage

```
appgw.ingress.kubernetes.io/backend-path-prefix: <path prefix>
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-bkprefix
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/backend-path-prefix: "/test/"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 80
```

In the example above, we have defined an ingress resource named `go-server-ingress-bkprefix` with an annotation `appgw.ingress.kubernetes.io/backend-path-prefix: "/test/"` . The annotation tells application gateway to create an HTTP setting, which will have a path prefix override for the path `/hello` to `/test/`.

> **NOTE**
> In the above example we have only one rule defined. However, the annotations are applicable to the entire ingress resource, so if a user had defined multiple rules, the backend path prefix would be set up for each of the paths specified. Thus, if a user wants different rules with different path prefixes (even for the same service) they would need to define different ingress resources.

TLS Redirect

Application Gateway can be configured to automatically redirect HTTP URLs to their HTTPS counterparts. When this annotation is present and TLS is properly configured, Kubernetes Ingress controller will create a routing rule with a redirection configuration and apply the changes to your Application Gateway. The redirect created will be HTTP `301 Moved Permanently`.

Usage

```
appgw.ingress.kubernetes.io/ssl-redirect: "true"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-redirect
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
   - hosts:
     - www.contoso.com
     secretName: testsecret-tls
  rules:
  - host: www.contoso.com
    http:
      paths:
      - backend:
          serviceName: websocket-repeater
          servicePort: 80
```

Connection Draining

`connection-draining`: This annotation allows users to specify whether to enable connection draining.
`connection-draining-timeout`: This annotation allows users to specify a timeout after which Application Gateway will terminate the requests to the draining backend endpoint.

Usage

```
appgw.ingress.kubernetes.io/connection-draining: "true"
appgw.ingress.kubernetes.io/connection-draining-timeout: "60"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-drain
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/connection-draining: "true"
    appgw.ingress.kubernetes.io/connection-draining-timeout: "60"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 80
```

Cookie Based Affinity

This annotation allows to specify whether to enable cookie based affinity.

Usage

```
appgw.ingress.kubernetes.io/cookie-based-affinity: "true"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-affinity
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/cookie-based-affinity: "true"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 80
```

Request Timeout

This annotation allows to specify the request timeout in seconds after which Application Gateway will fail the request if response is not received.

Usage

```
appgw.ingress.kubernetes.io/request-timeout: "20"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-timeout
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/request-timeout: "20"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 80
```

Use Private IP

This annotation allows us to specify whether to expose this endpoint on Private IP of Application Gateway.

NOTE

- Application Gateway doesn't support multiple IPs on the same port (example: 80/443). Ingress with annotation `appgw.ingress.kubernetes.io/use-private-ip: "false"` and another with `appgw.ingress.kubernetes.io/use-private-ip: "true"` on `HTTP` will cause AGIC to fail in updating the Application Gateway.
- For Application Gateway that doesn't have a private IP, Ingresses with `appgw.ingress.kubernetes.io/use-private-ip: "true"` will be ignored. This will reflected in the controller logs and ingress events for those ingresses with `NoPrivateIP` warning.

Usage

```
appgw.ingress.kubernetes.io/use-private-ip: "true"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-timeout
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/use-private-ip: "true"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 80
```

Backend Protocol

This annotation allows us to specify the protocol that Application Gateway should use while talking to the Pods. Supported Protocols: `http`, `https`

NOTE

- While self-signed certificates are supported on Application Gateway, currently, AGIC only support `https` when Pods are using certificate signed by a well-known CA.
- Make sure to not use port 80 with HTTPS and port 443 with HTTP on the Pods.

Usage

```
appgw.ingress.kubernetes.io/backend-protocol: "https"
```

Example

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-server-ingress-timeout
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/backend-protocol: "https"
spec:
  rules:
  - http:
      paths:
      - path: /hello/
        backend:
          serviceName: go-server-service
          servicePort: 443
```

Rewrite Rule Set

This annotation allows you to assign an existing rewrite rule set to the corresponding request routing rule.

Usage

```
appgw.ingress.kubernetes.io/rewrite-rule-set: <rewrite rule set name>
```

Example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: go-server-ingress-bkprefix
  namespace: test-ag
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/rewrite-rule-set: add-custom-response-header
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Exact
        backend:
          service:
            name: go-server-service
            port:
              number: 8080
```

What is Azure Resource Manager?

8/22/2022 • 6 minutes to read • Edit Online

Azure Resource Manager is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure account. You use management features, like access control, locks, and tags, to secure and organize your resources after deployment.

To learn about Azure Resource Manager templates (ARM templates), see the ARM template overview. To learn about Bicep, see Bicep overview.

Consistent management layer

When you send a request through any of the Azure APIs, tools, or SDKs, Resource Manager receives the request. It authenticates and authorizes the request before forwarding it to the appropriate Azure service. Because all requests are handled through the same API, you see consistent results and capabilities in all the different tools.

The following image shows the role Azure Resource Manager plays in handling Azure requests.

All capabilities that are available in the portal are also available through PowerShell, Azure CLI, REST APIs, and client SDKs. Functionality initially released through APIs will be represented in the portal within 180 days of initial release.

Terminology

If you're new to Azure Resource Manager, there are some terms you might not be familiar with.

- **resource** - A manageable item that is available through Azure. Virtual machines, storage accounts, web apps, databases, and virtual networks are examples of resources. Resource groups, subscriptions, management groups, and tags are also examples of resources.
- **resource group** - A container that holds related resources for an Azure solution. The resource group includes those resources that you want to manage as a group. You decide which resources belong in a resource group based on what makes the most sense for your organization. See Resource groups.
- **resource provider** - A service that supplies Azure resources. For example, a common resource provider is `Microsoft.Compute`, which supplies the virtual machine resource. `Microsoft.Storage` is another common resource provider. See Resource providers and types.
- **declarative syntax** - Syntax that lets you state "Here's what I intend to create" without having to write the sequence of programming commands to create it. ARM templates and Bicep files are examples of declarative

syntax. In those files, you define the properties for the infrastructure to deploy to Azure.

- **ARM template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group, subscription, management group, or tenant. The template can be used to deploy the resources consistently and repeatedly. See Template deployment overview.
- **Bicep file** - A file for declaratively deploying Azure resources. Bicep is a language that's been designed to provide the best authoring experience for infrastructure as code solutions in Azure. See Bicep overview.

For more definitions of Azure terminology, see Azure fundamental concepts.

The benefits of using Resource Manager

With Resource Manager, you can:

- Manage your infrastructure through declarative templates rather than scripts.
- Deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- Redeploy your solution throughout the development lifecycle and have confidence your resources are deployed in a consistent state.
- Define the dependencies between resources so they're deployed in the correct order.
- Apply access control to all services because Azure role-based access control (Azure RBAC) is natively integrated into the management platform.
- Apply tags to resources to logically organize all the resources in your subscription.
- Clarify your organization's billing by viewing costs for a group of resources sharing the same tag.

Understand scope

Azure provides four levels of scope: management groups, subscriptions, resource groups, and resources. The following image shows an example of these layers.

You apply management settings at any of these levels of scope. The level you select determines how widely the setting is applied. Lower levels inherit settings from higher levels. For example, when you apply a policy to the subscription, the policy is applied to all resource groups and resources in your subscription. When you apply a policy on the resource group, that policy is applied to the resource group and all its resources. However, another resource group doesn't have that policy assignment.

For information about managing identities and access, see Azure Active Directory.

You can deploy templates to tenants, management groups, subscriptions, or resource groups.

Resource groups

There are some important factors to consider when defining your resource group:

- All the resources in your resource group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a server, needs to exist on a different deployment cycle it should be in another resource group.
- Each resource can exist in only one resource group.
- You can add or remove a resource to a resource group at any time.
- You can move a resource from one resource group to another group. For more information, see Move resources to new resource group or subscription.
- The resources in a resource group can be located in different regions than the resource group.
- When you create a resource group, you need to provide a location for that resource group.

  You may be wondering, "Why does a resource group need a location? And, if the resources can have different locations than the resource group, why does the resource group location matter at all?"

  The resource group stores metadata about the resources. When you specify a location for the resource group, you're specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

  If a resource group's region is temporarily unavailable, you can't update resources in the resource group because the metadata is unavailable. The resources in other regions will still function as expected, but you can't update them. This condition doesn't apply to global resources like Azure Content Delivery Network, Azure DNS, Azure Traffic Manager, and Azure Front Door.

  For more information about building reliable applications, see Designing reliable Azure applications.
- A resource group can be used to scope access control for administrative actions. To manage a resource group, you can assign Azure Policies, Azure roles, or resource locks.
- You can apply tags to a resource group. The resources in the resource group don't inherit those tags.
- A resource can connect to resources in other resource groups. This scenario is common when the two resources are related but don't share the same lifecycle. For example, you can have a web app that connects to a database in a different resource group.
- When you delete a resource group, all resources in the resource group are also deleted. For information about how Azure Resource Manager orchestrates those deletions, see Azure Resource Manager resource group and resource deletion.
- You can deploy up to 800 instances of a resource type in each resource group. Some resource types are exempt from the 800 instance limit. For more information, see resource group limits.
- Some resources can exist outside of a resource group. These resources are deployed to the subscription, management group, or tenant. Only specific resource types are supported at these scopes.
- To create a resource group, you can use the portal, PowerShell, Azure CLI, or an ARM template.

Resiliency of Azure Resource Manager

The Azure Resource Manager service is designed for resiliency and continuous availability. Resource Manager and control plane operations (requests sent to `management.azure.com`) in the REST API are:

- Distributed across regions. Some services are regional.
- Distributed across Availability Zones (and regions) in locations that have multiple Availability Zones.

- Not dependent on a single logical data center.
- Never taken down for maintenance activities.

This resiliency applies to services that receive requests through Resource Manager. For example, Key Vault benefits from this resiliency.

Next steps

- To learn about limits that are applied across Azure services, see Azure subscription and service limits, quotas, and constraints.
- To learn about moving resources, see Move resources to new resource group or subscription.
- To learn about tagging resources, see Use tags to organize your Azure resources.
- To learn about locking resources, see Lock resources to prevent unexpected changes.

Understand the structure and syntax of ARM templates

8/22/2022 • 14 minutes to read • Edit Online

This article describes the structure of an Azure Resource Manager template (ARM template). It presents the different sections of a template and the properties that are available in those sections.

This article is intended for users who have some familiarity with ARM templates. It provides detailed information about the structure of the template. For a step-by-step tutorial that guides you through the process of creating a template, see Tutorial: Create and deploy your first ARM template. To learn about ARM templates through a guided set of Learn modules, see Deploy and manage resources in Azure by using ARM templates.

> **TIP**
>
> Bicep is a new language that offers the same capabilities as ARM templates but with a syntax that's easier to use. If you're considering infrastructure as code options, we recommend looking at Bicep.
>
> To learn about the elements of a Bicep file, see Understand the structure and syntax of Bicep files.

Template format

In its simplest structure, a template has the following elements:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "",
  "apiProfile": "",
  "parameters": {  },
  "variables": {  },
  "functions": [  ],
  "resources": [  ],
  "outputs": {  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION

ELEMENT NAME	REQUIRED	DESCRIPTION

$schema	Yes	Location of the JavaScript Object Notation (JSON) schema file that describes the version of the template language. The version number you use depends on the scope of the deployment and your JSON editor.
		
		If you're using Visual Studio Code with the Azure Resource Manager tools extension, use the latest version for resource group deployments:
		`https://schema.management.azure.com/schemas/2004-01/deploymentTemplate.json#`
		
		Other editors (including Visual Studio) may not be able to process this schema. For those editors, use:
		`https://schema.management.azure.com/schemas/2001-01/deploymentTemplate.json#`
		
		For subscription deployments, use:
		`https://schema.management.azure.com/schemas/2005-01/subscriptionDeploymentTemplate.json#`
		
		For management group deployments, use:
		`https://schema.management.azure.com/schemas/2008-01/managementGroupDeploymentTemplate.json#`
		
		For tenant deployments, use:
		`https://schema.management.azure.com/schemas/2008-01/tenantDeploymentTemplate.json#`

contentVersion	Yes	Version of the template (such as 1.0.0.0). You can provide any value for this element. Use this value to document significant changes in your template. When deploying resources using the template, this value can be used to make sure that the right template is being used.

ELEMENT NAME	REQUIRED	DESCRIPTION
apiProfile	No	An API version that serves as a collection of API versions for resource types. Use this value to avoid having to specify API versions for each resource in the template. When you specify an API profile version and don't specify an API version for the resource type, Resource Manager uses the API version for that resource type that is defined in the profile. The API profile property is especially helpful when deploying a template to different environments, such as Azure Stack and global Azure. Use the API profile version to make sure your template automatically uses versions that are supported in both environments. For a list of the current API profile versions and the resources API versions defined in the profile, see API Profile. For more information, see Track versions using API profiles.
parameters	No	Values that are provided when deployment is executed to customize resource deployment.
variables	No	Values that are used as JSON fragments in the template to simplify template language expressions.
functions	No	User-defined functions that are available within the template.
resources	Yes	Resource types that are deployed or updated in a resource group or subscription.
outputs	No	Values that are returned after deployment.

Each element has properties you can set. This article describes the sections of the template in greater detail.

Parameters

In the `parameters` section of the template, you specify which values you can input when deploying the resources. You're limited to 256 parameters in a template. You can reduce the number of parameters by using objects that contain multiple properties.

The available properties for a parameter are:

```
"parameters": {
  "<parameter-name>" : {
    "type" : "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter>",
    "allowedValues": [ "<array-of-allowed-values>" ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array-parameters>,
    "metadata": {
      "description": "<description-of-the parameter>"
    }
  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
parameter-name	Yes	Name of the parameter. Must be a valid JavaScript identifier.
type	Yes	Type of the parameter value. The allowed types and values are **string**, **securestring**, **int**, **bool**, **object**, **secureObject**, and **array**. See Data types in ARM templates.
defaultValue	No	Default value for the parameter, if no value is provided for the parameter.
allowedValues	No	Array of allowed values for the parameter to make sure that the right value is provided.
minValue	No	The minimum value for int type parameters, this value is inclusive.
maxValue	No	The maximum value for int type parameters, this value is inclusive.
minLength	No	The minimum length for string, secure string, and array type parameters, this value is inclusive.
maxLength	No	The maximum length for string, secure string, and array type parameters, this value is inclusive.
description	No	Description of the parameter that is displayed to users through the portal. For more information, see Comments in templates.

For examples of how to use parameters, see Parameters in ARM templates.

In Bicep, see parameters.

Variables

In the `variables` section, you construct values that can be used throughout your template. You don't need to define variables, but they often simplify your template by reducing complex expressions. The format of each variable matches one of the data types.

The following example shows the available options for defining a variable:

```
"variables": {
  "<variable-name>": "<variable-value>",
  "<variable-name>": {
    <variable-complex-type-value>
  },
  "<variable-object-name>": {
    "copy": [
      {
        "name": "<name-of-array-property>",
        "count": <number-of-iterations>,
        "input": <object-or-value-to-repeat>
      }
    ]
  },
  "copy": [
    {
      "name": "<variable-array-name>",
      "count": <number-of-iterations>,
      "input": <object-or-value-to-repeat>
    }
  ]
}
```

For information about using `copy` to create several values for a variable, see Variable iteration.

For examples of how to use variables, see Variables in ARM template.

In Bicep, see variables.

Functions

Within your template, you can create your own functions. These functions are available for use in your template. Typically, you define complicated expressions that you don't want to repeat throughout your template. You create the user-defined functions from expressions and functions that are supported in templates.

When defining a user function, there are some restrictions:

- The function can't access variables.
- The function can only use parameters that are defined in the function. When you use the parameters function within a user-defined function, you're restricted to the parameters for that function.
- The function can't call other user-defined functions.
- The function can't use the reference function.
- Parameters for the function can't have default values.

```
"functions": [
  {
    "namespace": "<namespace-for-functions>",
    "members": {
      "<function-name>": {
        "parameters": [
          {
            "name": "<parameter-name>",
            "type": "<type-of-parameter-value>"
          }
        ],
        "output": {
          "type": "<type-of-output-value>",
          "value": "<function-return-value>"
        }
      }
    }
  }
],
```

ELEMENT NAME	REQUIRED	DESCRIPTION
namespace	Yes	Namespace for the custom functions. Use to avoid naming conflicts with template functions.
function-name	Yes	Name of the custom function. When calling the function, combine the function name with the namespace. For example, to call a function named `uniqueName` in the namespace contoso, use `"[contoso.uniqueName()]"`.
parameter-name	No	Name of the parameter to be used within the custom function.
parameter-value	No	Type of the parameter value. The allowed types and values are **string**, **securestring**, **int**, **bool**, **object**, **secureObject**, and **array**.
output-type	Yes	Type of the output value. Output values support the same types as function input parameters.
output-value	Yes	Template language expression that is evaluated and returned from the function.

For examples of how to use custom functions, see User-defined functions in ARM template.

In Bicep, user-defined functions aren't supported. Bicep does support a variety of functions and operators.

Resources

In the `resources` section, you define the resources that are deployed or updated.

You define resources with the following structure:

```
"resources": [
  {
      "condition": "<true-to-deploy-this-resource>",
      "type": "<resource-provider-namespace/resource-type-name>",
      "apiVersion": "<api-version-of-resource>",
      "name": "<name-of-the-resource>",
      "comments": "<your-reference-notes>",
      "location": "<location-of-resource>",
      "dependsOn": [
          "<array-of-related-resource-names>"
      ],
      "tags": {
          "<tag-name1>": "<tag-value1>",
          "<tag-name2>": "<tag-value2>"
      },
      "identity": {
        "type": "<system-assigned-or-user-assigned-identity>",
        "userAssignedIdentities": {
          "<resource-id-of-identity>": {}
        }
      },
      "sku": {
          "name": "<sku-name>",
          "tier": "<sku-tier>",
          "size": "<sku-size>",
          "family": "<sku-family>",
          "capacity": <sku-capacity>
      },
      "kind": "<type-of-resource>",
      "scope": "<target-scope-for-extension-resources>",
      "copy": {
          "name": "<name-of-copy-loop>",
          "count": <number-of-iterations>,
          "mode": "<serial-or-parallel>",
          "batchSize": <number-to-deploy-serially>
      },
      "plan": {
          "name": "<plan-name>",
          "promotionCode": "<plan-promotion-code>",
          "publisher": "<plan-publisher>",
          "product": "<plan-product>",
          "version": "<plan-version>"
      },
      "properties": {
          "<settings-for-the-resource>",
          "copy": [
              {
                  "name": ,
                  "count": ,
                  "input": {}
              }
          ]
      },
      "resources": [
          "<array-of-child-resources>"
      ]
  }
]
```

ELEMENT NAME	REQUIRED	DESCRIPTION
condition	No	Boolean value that indicates whether the resource will be provisioned during this deployment. When `true`, the resource is created during deployment. When `false`, the resource is skipped for this deployment. See condition.

ELEMENT NAME	REQUIRED	DESCRIPTION
type	Yes	Type of the resource. This value is a combination of the namespace of the resource provider and the resource type (such as `Microsoft.Storage/storageAccounts`). To determine available values, see template reference. For a child resource, the format of the type depends on whether it's nested within the parent resource or defined outside of the parent resource. See Set name and type for child resources.
apiVersion	Yes	Version of the REST API to use for creating the resource. When creating a new template, set this value to the latest version of the resource you're deploying. As long as the template works as needed, keep using the same API version. By continuing to use the same API version, you minimize the risk of a new API version changing how your template works. Consider updating the API version only when you want to use a new feature that is introduced in a later version. To determine available values, see template reference.
name	Yes	Name of the resource. The name must follow URI component restrictions defined in RFC3986. Azure services that expose the resource name to outside parties validate the name to make sure it isn't an attempt to spoof another identity. For a child resource, the format of the name depends on whether it's nested within the parent resource or defined outside of the parent resource. See Set name and type for child resources.
comments	No	Your notes for documenting the resources in your template. For more information, see Comments in templates.
location	Varies	Supported geo-locations of the provided resource. You can select any of the available locations, but typically it makes sense to pick one that is close to your users. Usually, it also makes sense to place resources that interact with each other in the same region. Most resource types require a location, but some types (such as a role assignment) don't require a location. See Set resource location.

ELEMENT NAME	REQUIRED	DESCRIPTION
dependsOn	No	Resources that must be deployed before this resource is deployed. Resource Manager evaluates the dependencies between resources and deploys them in the correct order. When resources aren't dependent on each other, they're deployed in parallel. The value can be a comma-separated list of a resource names or resource unique identifiers. Only list resources that are deployed in this template. Resources that aren't defined in this template must already exist. Avoid adding unnecessary dependencies as they can slow your deployment and create circular dependencies. For guidance on setting dependencies, see Define the order for deploying resources in ARM templates.
tags	No	Tags that are associated with the resource. Apply tags to logically organize resources across your subscription.
identity	No	Some resources support managed identities for Azure resources. Those resources have an identity object at the root level of the resource declaration. You can set whether the identity is user-assigned or system-assigned. For user-assigned identities, provide a list of resource IDs for the identities. Set the key to the resource ID and the value to an empty object. For more information, see Configure managed identities for Azure resources on an Azure VM using templates.
sku	No	Some resources allow values that define the SKU to deploy. For example, you can specify the type of redundancy for a storage account.
kind	No	Some resources allow a value that defines the type of resource you deploy. For example, you can specify the type of Cosmos DB to create.
scope	No	The scope property is only available for extension resource types. Use it when specifying a scope that is different than the deployment scope. See Setting scope for extension resources in ARM templates.

ELEMENT NAME	REQUIRED	DESCRIPTION
copy	No	If more than one instance is needed, the number of resources to create. The default mode is parallel. Specify serial mode when you don't want all or the resources to deploy at the same time. For more information, see Create several instances of resources in Azure Resource Manager.
plan	No	Some resources allow values that define the plan to deploy. For example, you can specify the marketplace image for a virtual machine.
properties	No	Resource-specific configuration settings. The values for the properties are the same as the values you provide in the request body for the REST API operation (PUT method) to create the resource. You can also specify a copy array to create several instances of a property. To determine available values, see template reference.
resources	No	Child resources that depend on the resource being defined. Only provide resource types that are permitted by the schema of the parent resource. Dependency on the parent resource isn't implied. You must explicitly define that dependency. See Set name and type for child resources.

In Bicep, see resources.

Outputs

In the `outputs` section, you specify values that are returned from deployment. Typically, you return values from resources that were deployed.

The following example shows the structure of an output definition:

```
"outputs": {
  "<output-name>": {
    "condition": "<boolean-value-whether-to-output-value>",
    "type": "<type-of-output-value>",
    "value": "<output-value-expression>",
    "copy": {
      "count": <number-of-iterations>,
      "input": <values-for-the-variable>
    }
  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
output-name	Yes	Name of the output value. Must be a valid JavaScript identifier.

ELEMENT NAME	REQUIRED	DESCRIPTION
condition	No	Boolean value that indicates whether this output value is returned. When `true`, the value is included in the output for the deployment. When `false`, the output value is skipped for this deployment. When not specified, the default value is `true`.
type	Yes	Type of the output value. Output values support the same types as template input parameters. If you specify **securestring** for the output type, the value isn't displayed in the deployment history and can't be retrieved from another template. To use a secret value in more than one template, store the secret in a Key Vault and reference the secret in the parameter file. For more information, see Use Azure Key Vault to pass secure parameter value during deployment.
value	No	Template language expression that is evaluated and returned as output value. Specify either **value** or **copy**.
copy	No	Used to return more than one value for an output. Specify **value** or **copy**. For more information, see Output iteration in ARM templates.

For examples of how to use outputs, see Outputs in ARM template.

In Bicep, see outputs.

Comments and metadata

You have a few options for adding comments and metadata to your template.

Comments

For inline comments, you can use either `//` or `/* ... */`. In Visual Studio Code, save the parameter files with comments as the **JSON with comments (JSONC)** file type, otherwise you will get an error message saying "Comments not permitted in JSON".

> **NOTE**
>
> When using Azure CLI to deploy templates with comments, use version 2.3.0 or later, and specify the `--handle-extended-json-format` switch.

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2018-10-01",
  "name": "[variables('vmName')]", // to customize name, change it in variables
  "location": "[parameters('location')]", //defaults to resource group location
  "dependsOn": [ /* storage account and network interface must be deployed first */
    "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
    "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
  ],
```

In Visual Studio Code, the Azure Resource Manager Tools extension can automatically detect an ARM template and change the language mode. If you see **Azure Resource Manager Template** at the bottom-right corner of Visual Studio Code, you can use the inline comments. The inline comments are no longer marked as invalid.

In Bicep, see comments.

Metadata

You can add a `metadata` object almost anywhere in your template. Resource Manager ignores the object, but your JSON editor may warn you that the property isn't valid. In the object, define the properties you need.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "comments": "This template was developed for demonstration purposes.",
    "author": "Example Name"
  },
```

For `parameters`, add a `metadata` object with a `description` property.

```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "User name for the Virtual Machine."
    }
  },
```

When deploying the template through the portal, the text you provide in the description is automatically used as a tip for that parameter.

For `resources`, add a `comments` element or a `metadata` object. The following example shows both a `comments` element and a `metadata` object.

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2018-07-01",
    "name": "[concat('storage', uniqueString(resourceGroup().id))]",
    "comments": "Storage account used to store VM disks",
    "location": "[parameters('location')]",
    "metadata": {
      "comments": "These tags are needed for policy compliance."
    },
    "tags": {
      "Dept": "[parameters('deptName')]",
      "Environment": "[parameters('environment')]"
    },
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {}
  }
]
```

For `outputs`, add a `metadata` object to the output value.

```
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]",
    "metadata": {
      "comments": "Return the fully qualified domain name"
    }
  },
```

You can't add a `metadata` object to user-defined functions.

Multi-line strings

You can break a string into multiple lines. For example, see the `location` property and one of the comments in the following JSON example.

> **NOTE**
>
> To deploy templates with multi-line strings, use Azure PowerShell or Azure CLI. For CLI, use version 2.3.0 or later, and specify the `--handle-extended-json-format` switch.
>
> Multi-line strings aren't supported when you deploy the template through the Azure portal, a DevOps pipeline, or the REST API.

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2018-10-01",
  "name": "[variables('vmName')]", // to customize name, change it in variables
  "location": "[
    parameters('location')
    ]", //defaults to resource group location
  /*
    storage account and network interface
    must be deployed first
  */
  "dependsOn": [
    "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
    "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
  ],
```

In Bicep, see multi-line strings.

Next steps

- To view complete templates for many different types of solutions, see the Azure Quickstart Templates.
- For details about the functions you can use from within a template, see ARM template functions.
- To combine several templates during deployment, see Using linked and nested templates when deploying Azure resources.
- For recommendations about creating templates, see ARM template best practices.
- For answers to common questions, see Frequently asked questions about ARM templates.