

Contents

[Application Gateway Documentation](#)

[Overview](#)

[About application gateways](#)

[What's new?](#)

[Quickstarts](#)

[Create Application Gateway - Portal](#)

[Create Application Gateway - PowerShell](#)

[Create Application Gateway - Azure CLI](#)

[Tutorials](#)

[Secure with SSL](#)

[Host multiple sites](#)

[Route by URL](#)

[Redirect web traffic](#)

[Autoscaling and zone redundant](#)

[Samples](#)

[Azure PowerShell](#)

[Azure CLI](#)

[Resource Manager templates](#)

[Concepts](#)

[Basics](#)

[How Application Gateway works](#)

[Application Gateway components](#)

[Configuration overview](#)

[Autoscaling v2 SKU](#)

[Routing](#)

[Multiple site hosting](#)

[URL routing](#)

[Redirection](#)

[Rewrite HTTP headers](#)

[Custom error pages](#)

[SSL](#)

[SSL termination and end to end SSL](#)

[SSL policy overview](#)

[Using Key Vault](#)

[Ingress for AKS](#)

[Health monitoring](#)

[Health probe](#)

[Diagnostic logs and backend health](#)

[Metrics](#)

[App service webapp and multi-tenant support](#)

[WebSocket support](#)

[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](#)

[End-to-end SSL](#)

[End-to-end SSL - Portal](#)

[End-to-end SSL - PowerShell](#)

[Manage certificates](#)

[Certificates for the backend](#)

[Renew certificates](#)

[Generate self-signed certificates](#)

[SSL policy](#)

[Configure SSL policy](#)

[Ingress for AKS](#)

[Set up with existing Application Gateway](#)

[Set up with new Application Gateway](#)

[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](#)

[Azure portal](#)

[Azure PowerShell](#)

[Create and rewrite HTTP headers](#)

[Configure App service webapp and multi-tenant service](#)

[Portal](#)

[Azure PowerShell](#)

[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](#)

[Ingress for AKS](#)

[Resource Health](#)

[Use Log Analytics](#)

[Backend health issues](#)

[Migrate from v1 to v2](#)

[Reference](#)

[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](#)

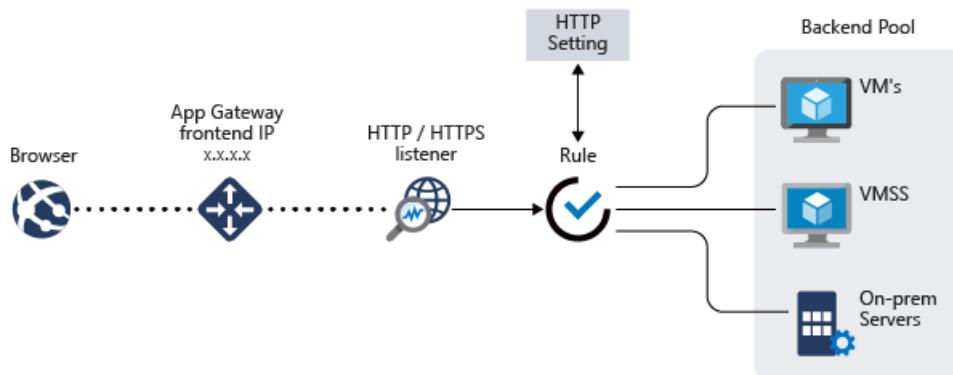
[Regional availability](#)

[Stack Overflow](#)

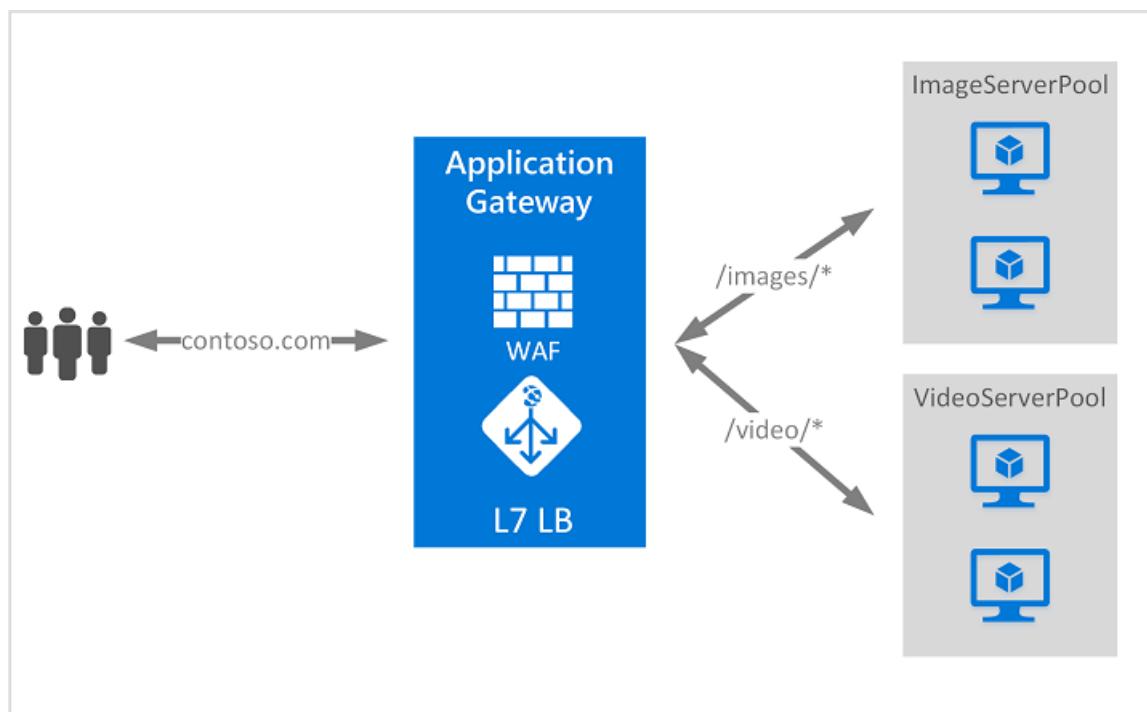
What is Azure Application Gateway?

2/18/2020 • 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. 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.



With Application Gateway, you can make routing decisions based on additional attributes of an HTTP request, such as 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 need high-performance, low-latency, Layer-4 load balancing, see [What is Azure Load Balancer?](#) If you're looking for global DNS load balancing, see [What is Traffic Manager?](#) Your end-to-end scenarios may benefit from combining these solutions.

For an Azure load-balancing options comparison, see [Overview of load-balancing options in Azure](#).

The following features are included with Azure Application Gateway:

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 is not 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.

Autoscaling

Application Gateway or WAF deployments under Standard_v2 or WAF_v2 SKU support 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 and WAF_v2 features, see [Autoscaling v2 SKU](#).

Zone redundancy

An Application Gateway or WAF deployments under Standard_v2 or WAF_v2 SKU 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 VIP on Standard_v2 or WAF_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-based routing with Application Gateway](#).

Multiple-site hosting

Multiple-site hosting enables you to configure more than one web site on the same application gateway instance. This feature allows you to configure a more efficient topology for your deployments by adding up to 100 web sites to one Application Gateway, or 40 for WAF (for optimal performance). Each web site can be directed to its own pool. For example, application gateway can serve traffic for `contoso.com` and `fabrikam.com` from two server pools called ContosoServerPool and FabrikamServerPool.

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

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 [multiple-site hosting with Application Gateway](#).

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 [redirecting traffic](#) with Application Gateway.

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.

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 do not 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 will continue to be proxied to the deregistering instances.

For more information, see the Connection Draining section of [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

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 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. It also provides you with

the capability to add conditions to ensure the specified headers are rewritten only when certain conditions are met.

For more information, see [Rewrite HTTP headers](#).

Sizing

Application Gateway Standard_v2 and WAF_v2 SKU can be configured for autoscaling or fixed size deployments. These SKUs don't offer different instance sizes. For more information on v2 performance and pricing, see [Autoscaling v2 SKU](#).

The Application Gateway Standard and WAF SKU is currently 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.

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](#)

What's new in Azure Application Gateway?

2/27/2020 • 2 minutes to read • [Edit Online](#)

Azure Application Gateway is updated on an ongoing basis. To stay up-to-date with the most recent developments, this article provides you with information about:

- The latest releases
- Known issues
- Bug fixes
- Deprecated functionality

New features

FEATURE	DESCRIPTION	DATE ADDED
Affinity cookie changes	When cookie based affinity is enabled, Application Gateway injects another identical cookie called <i>ApplicationGatewayAffinityCORS</i> in addition to the existing <i>ApplicationGatewayAffinity</i> cookie. <i>ApplicationGatewayAffinityCORS</i> has two more attributes added to it (<i>SameSite=None; Secure</i>) so that sticky session are maintained even for cross-origin requests. See Application Gateway Cookie based affinity for more information.	February 2020
Probe enhancements	With the custom probe enhancements in Application Gateway v2 SKU, we have simplified probe configuration , facilitated on-demand backend health tests and added more diagnostic information to help you troubleshoot backend health issues.	October 2019
More metrics	We've added the following new metrics to help you monitor your Application Gateway v2 SKU: Timing-related metrics , Backend response status, Bytes received, Bytes sent, Client TLS protocol and Current compute units. See Metrics supported by Application Gateway V2 SKU .	August 2019
WAF custom rules	Application Gateway WAF_v2 now supports creating custom rules. See Application Gateway custom rules .	June 2019

FEATURE	DESCRIPTION	DATE ADDED
Autoscaling, zone redundancy, static VIP support GA	General availability for v2 SKU which supports autoscaling, zone redundancy, enhance performance, static VIPs, Key Vault, Header rewrite. See Application Gateway autoscaling documentation .	April 2019
Key Vault integration	Application Gateway now supports integration with Key Vault (in public preview) for server certificates that are attached to HTTPS enabled listeners. See SSL termination with Key Vault certificates .	April 2019
Header CRUD/Rewrites	You can now rewrite HTTP headers. See Tutorial: Create an application gateway and rewrite HTTP headers for more information.	December 2018
WAF configuration and exclusion list	We've added more options to help you configure your WAF and reduce false positives. See Web application firewall request size limits and exclusion lists for more information.	December 2018
Autoscaling, zone redundancy, static VIP support	With the v2 SKU, there are many improvements such as Autoscaling, improved performance, and more. See What is Azure Application Gateway? for more information.	September 2018
Connection draining	Connection draining allows you to gracefully remove members from your backend pools. For more information, see Connection draining .	September 2018
Custom error pages	With custom error pages, you can create an error page within the format of the rest of your websites. To enable this, see Create Application Gateway custom error pages .	September 2018
Metrics Enhancements	You can get a better view of the state of your Application Gateway with enhanced metrics. To enable metrics on your Application Gateway, see Back-end health, diagnostic logs, and metrics for Application Gateway .	June 2018

Next steps

For more information about Azure Application Gateway, see [What is Azure Application Gateway?](#)

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

1/23/2020 • 8 minutes to read • [Edit Online](#)

This quickstart shows you how to use the Azure portal to create an application gateway. After creating the application gateway, you test it to make sure it's working correctly. With Azure Application Gateway, you direct your application web traffic to specific resources by assigning listeners to ports, creating rules, and adding resources to a backend pool. For the sake of simplicity, this article uses a simple setup with a public front-end IP, a basic listener to host a single site on this application gateway, two virtual machines used for the backend pool, and a basic request routing rule.

If you don't have an Azure subscription, create a [free account](#) before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Sign in to Azure

Sign in to the [Azure portal](#) with your Azure account.

Create an application gateway

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.

Home > Application gateways > Create an application gateway

Create an application gateway

Basics

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: **(New) myResourceGroupAG** [Create new](#)

Instance details

Application gateway name: **myAppGateway**

Region: **(US) West US 2**

Tier: **Standard V2**

Enable autoscaling: **Yes** No

Minimum number of VMs: **2**

Maximum number of VMs:

Availability zone: **None**

HTTP/2: **Disabled** Enabled

Configure virtual network

Virtual network: **Create new**

< Previous [Next : Frontends >](#)

- 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, 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.

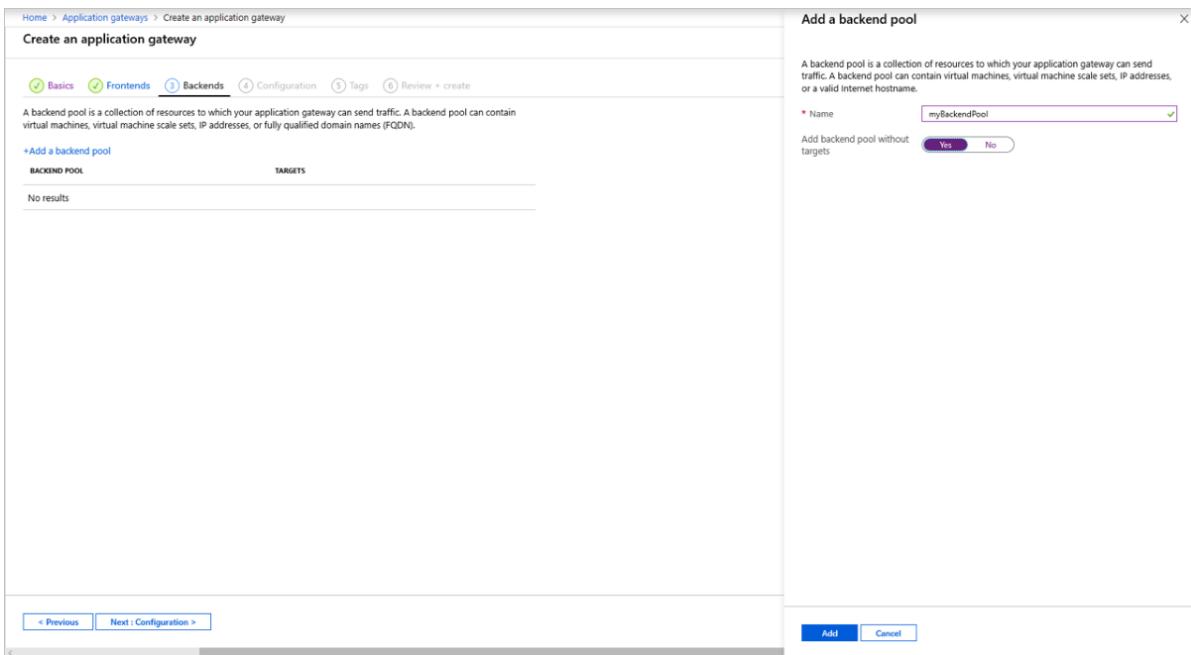
2. Choose **Create new** for the **Public IP address** and enter *myAGPublicIPAddress* for the public IP address name, and then select **OK**.

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 will 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 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 **Create 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 *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.

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. On the Azure portal menu or from the **Home** page, select **Create a resource**. The **New** window appears.
 2. Select **Compute** and then 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:
 - Resource group:** Select **myResourceGroupAG** for the resource group name.
 - Virtual machine name:** Enter *myVM* for the name of the virtual machine.
 - Username:** Enter *azureuser* for the administrator user name.
 - Password:** Enter *Azure123456!* for the administrator password.
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**.

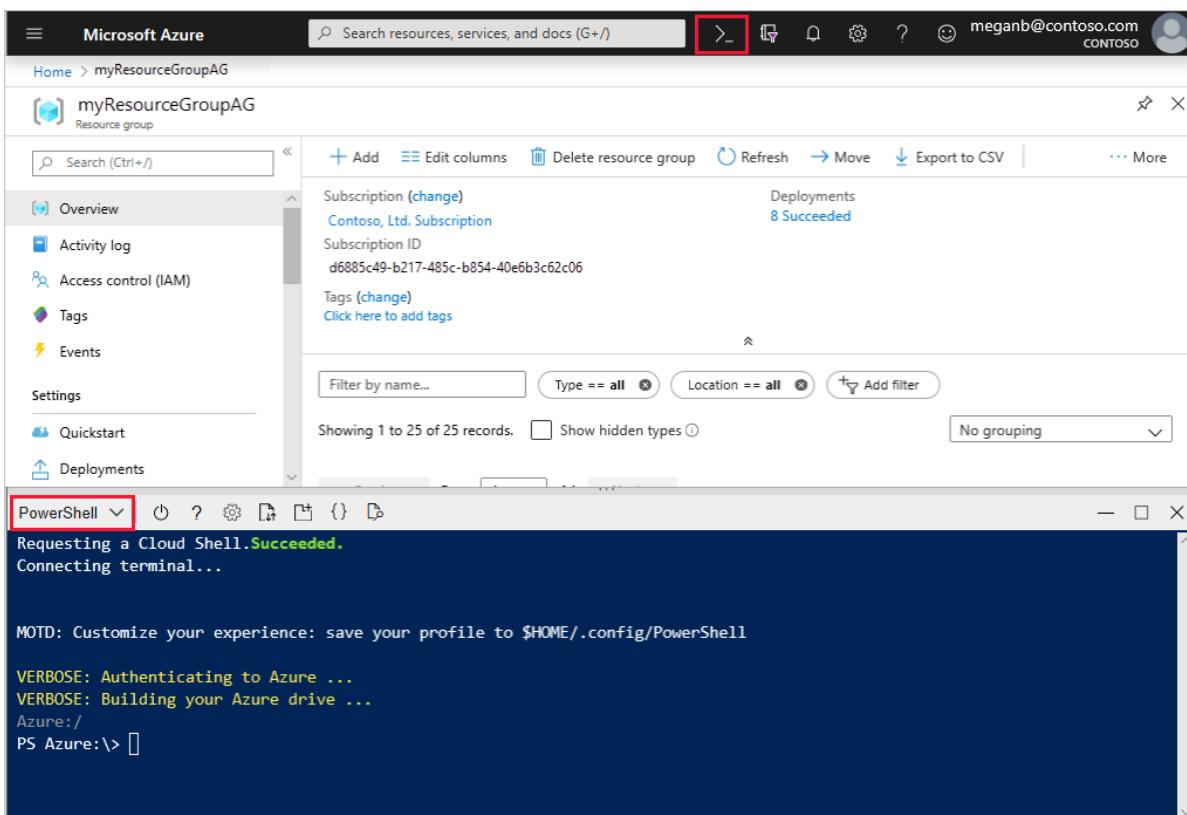
Application Gateway can communicate with instances outside of the virtual network that it is in, but you need to ensure there's IP connectivity.

7. On the **Management** tab, set **Boot diagnostics** to **Off**. 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:

```
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 **Targets**, select **Virtual machine** from the drop-down list.
5. Under **VIRTUAL MACHINE** and **NETWORK INTERFACES**, select the **myVM** and **myVM2** virtual machines and their associated network interfaces from the drop-down lists.

Home > Resource groups > myResourceGroupAG > myAppGateway - Backend pools > Edit backend pool

Edit backend pool

Save **Discard** **Delete**

Name
appGatewayBackendPool

Remove all targets from backend pool

Targets
A backend pool can be pointed to a specific virtual machine, virtual machine scale set, an IP address/FQDN or an app service.

Virtual machine

VIRTUAL MACHINE	NETWORK INTERFACES
myVM	myvm314
myVM2	myvm2554 (10.0.1.5)
Select a virtual machine	Waiting for virtual machine selection

Associated rule
[rule1](#)

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 whether 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.

Home > myAppGateway

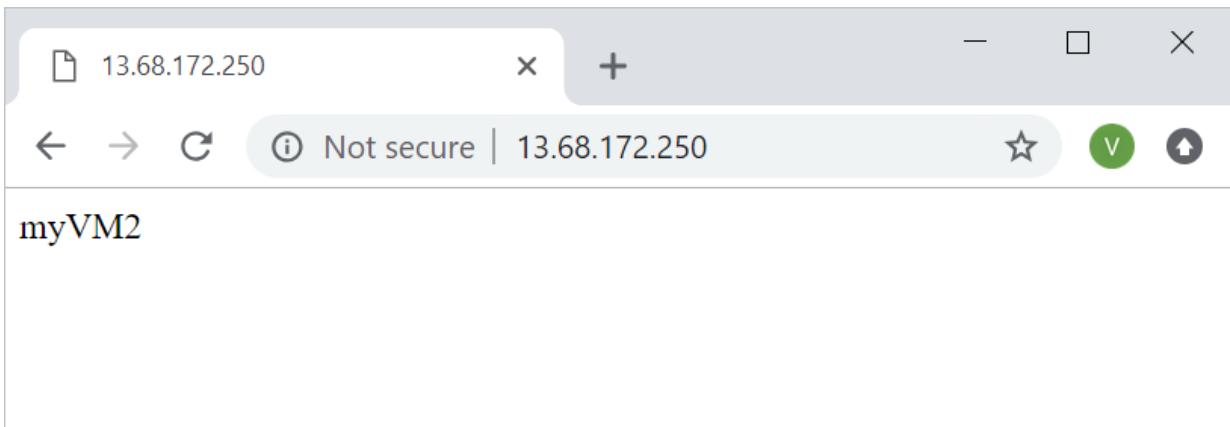
myAppGateway Application gateway

Overview

Resource group (change) : myResourceGroupAG
Location : East US
Subscription (change) : <subscription name>
Subscription ID : <subscription ID>
Tags (change) : Click here to add tags

Virtual network/subnet : myVNet/myAGSubnet
Frontend public IP addr... : 13.68.172.250 (myAppGateway-ip)
Frontend private IP addr... : -
Tier : Standard V2

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



Clean up resources

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

To remove 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* for **TYPE THE RESOURCE GROUP NAME** and then select **Delete**

Next steps

[Manage web traffic with an application gateway using the Azure CLI](#)

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

12/5/2019 • 7 minutes to read • [Edit Online](#)

This quickstart shows you how to use Azure PowerShell to quickly create an application gateway. After creating the application gateway, you then test it to make sure it's working correctly. With Azure Application Gateway, you direct your application web traffic to specific resources by assigning listeners to ports, creating rules, and adding resources to a backend pool. For the sake of simplicity, this article uses a simple setup with a public front-end IP, a basic listener to host a single site on this application gateway, two virtual machines used for the backend pool, and a basic request routing rule.

If you don't have an Azure subscription, create a [free account](#) before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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.

Prerequisites

Azure PowerShell module

If you choose to install and use Azure PowerShell locally, this tutorial requires the Azure PowerShell module version 1.0.0 or later.

1. To find the version, run `Get-Module -ListAvailable Az`. If you need to upgrade, see [Install Azure PowerShell module](#).
2. To create a connection with Azure, run `Login-AzAccount`.

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. In this example, you'll create a new resource group by using the `New-AzResourceGroup` cmdlet as follows:

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

Required 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 in this example: 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.

1. Create the subnet configurations by calling [New-AzVirtualNetworkSubnetConfig](#).
2. Create the virtual network with the subnet configurations by calling [New-AzVirtualNetwork](#).
3. Create the public IP address by calling [New-AzPublicIpAddress](#).

```
$agSubnetConfig = New-AzVirtualNetworkSubnetConfig `  
    -Name myAGSubnet `  
    -AddressPrefix 10.0.1.0/24  
$backendSubnetConfig = New-AzVirtualNetworkSubnetConfig `  
    -Name myBackendSubnet `  
    -AddressPrefix 10.0.2.0/24  
New-AzVirtualNetwork `  
    -ResourceGroupName myResourceGroupAG `  
    -Location eastus `  
    -Name myVNet `  
    -AddressPrefix 10.0.0.0/16 `  
    -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 to 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 will be empty for now and while you create the backend server NICs in the next section, you will add them to the backend pool.
2. Configure the settings for the backend pool with [New-AzApplicationGatewayBackendHttpSetting](#).

```
$address1 = Get-AzNetworkInterface -ResourceGroupName myResourceGroupAG -Name myNic1
$address2 = Get-AzNetworkInterface -ResourceGroupName myResourceGroupAG -Name myNic2
$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 by 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. Backend 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 create two virtual machines for Azure 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 `azureuser` for the user name and `Azure123456!` for the 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 $vm
Set-AzVMEExtension ` 
    -ResourceGroupName myResourceGroupAG ` 
    -ExtensionName IIS ` 
    -VMName myVM$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
}

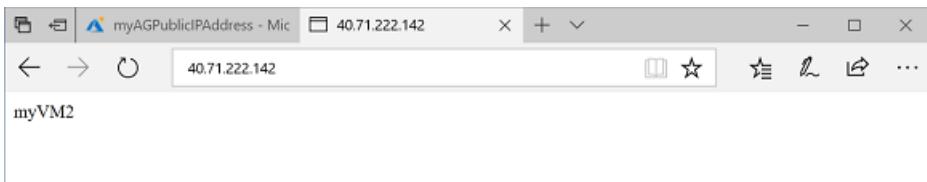
```

Test the application gateway

Although IIS isn't required to create the application gateway, you installed it in this quickstart to verify whether 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, remove the resource group. By removing the resource group, you also remove the application gateway and all its related resources.

To remove the resource group, call the [Remove-AzResourceGroup](#) cmdlet as follows:

```
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

11/13/2019 • 6 minutes to read • [Edit Online](#)

This quickstart shows you how to use Azure CLI to create an application gateway. After creating the application gateway, you test it to make sure it's working correctly. With Azure Application Gateway, you direct your application web traffic to specific resources by assigning listeners to ports, creating rules, and adding resources to a backend pool. This article uses a simple setup with a public front-end IP, a basic listener to host a single site on the application gateway, two virtual machines used for the backend pool, and a basic request routing rule.

If you don't have an Azure subscription, create a [free account](#) before you begin.

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.

Prerequisites

Azure CLI

If you choose to install and use the CLI locally, run Azure CLI version 2.0.4 or later. To find the version, run `az --version`. For information about installing or upgrading, see [Install Azure CLI](#).

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
```

Required 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 in this example: 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.

To create the virtual network and subnet, you 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 10.0.0.0/16 \
--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
```

Backend servers

A backend can have 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 create two virtual machines to use as backend servers for the application gateway. You also install IIS 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 by 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 \
--http-settings-cookie-based-affinity Enabled \
--public-ip-address myAGPublicIPAddress \
--vnet-name myVNet \
--subnet myAGSubnet \
--servers "$address1" "$address2"
```

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:

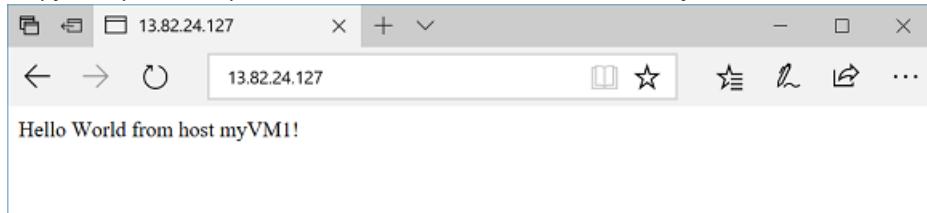
- **appGatewayBackendPool**: Located on the **Backend pools** page. It specifies the required backend pool.
- **appGatewayBackendHttpSettings**: Located on the **HTTP settings** page. It specifies that the application gateway uses port 80 and the HTTP protocol for communication.
- **appGatewayHttpListener**: 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 remove the resource group. By removing the resource group, you also remove 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](#)

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

11/12/2019 • 8 minutes to read • [Edit Online](#)

You can use the Azure portal to configure an [application gateway](#) with a certificate for SSL 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>

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:

NOTE

Do not use any special characters in your .pfx file password. Only alphanumeric characters are supported.

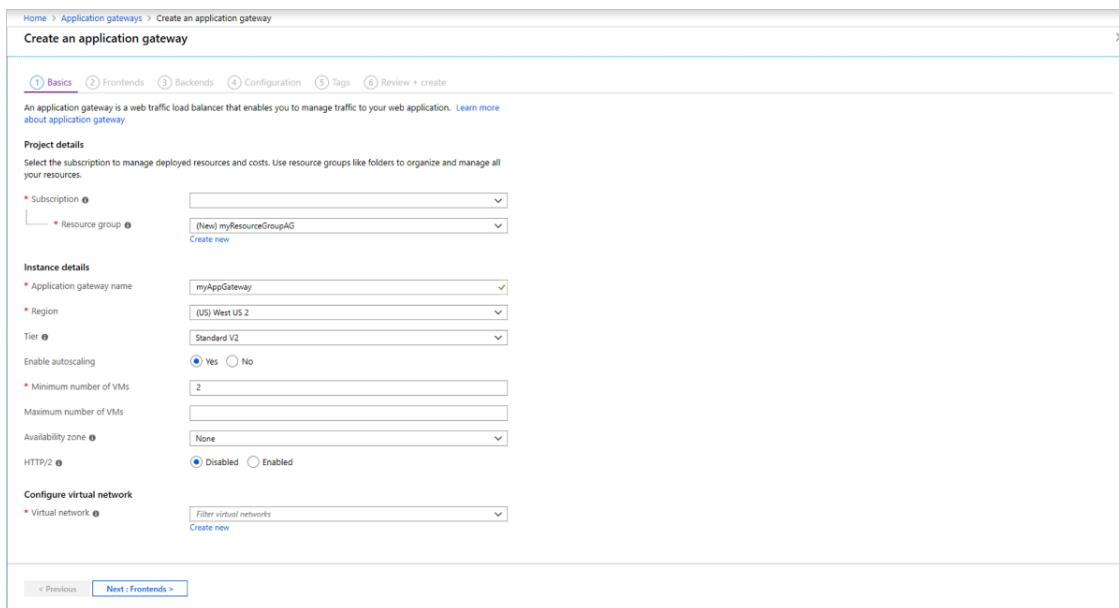
```
$pwd = ConvertTo-SecureString -String "Azure123456" -Force -AsPlainText
Export-PfxCertificate `-
    -cert cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `-
    -FilePath c:\appgwcert.pfx `-
    -Password $pwd
```

Create an application gateway

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

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.

The screenshot shows two windows from the Azure portal. On the left, the 'Create application gateway' wizard is displayed with the 'Basics' tab selected. It shows fields for Subscription (myResourceGroupAG), Application gateway name (myAppGateway), Region (US West US 2), Tier (Standard V2), and other configuration options like Minimum number of VMs (2) and HTTP/2 (Disabled). On the right, the 'Create virtual network' dialog box is open, showing the 'Subnets' section. It lists three subnets: 'myAGSubnet' with address range 10.21.0.0/24 and addresses 10.21.0.0 - 10.21.0.255 (256 addresses); 'myBackendSubnet' with address range 10.21.1.0/24 and addresses 10.21.1.0 - 10.21.1.255 (256 addresses); and a third unnamed subnet with address range 10.21.0.0/16 and addresses (0 Addresses). The 'myBackendSubnet' row is highlighted.

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

Frontends tab

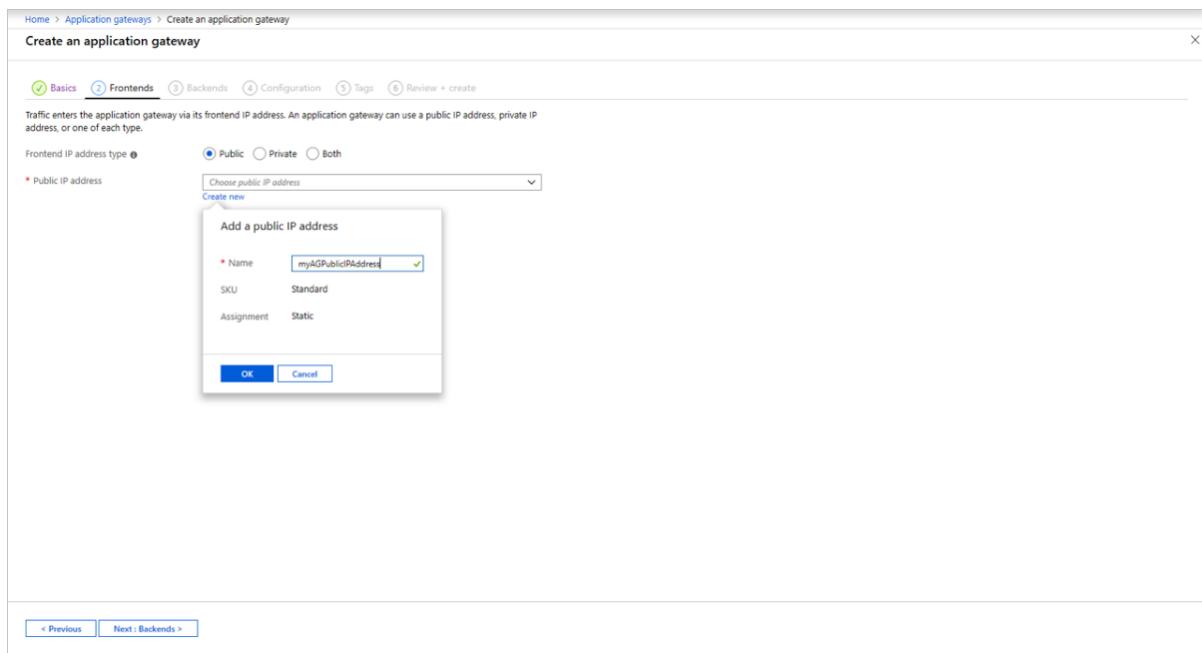
1. On the **Frontends** tab, verify **Frontend IP address type** is set to **Public**.

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

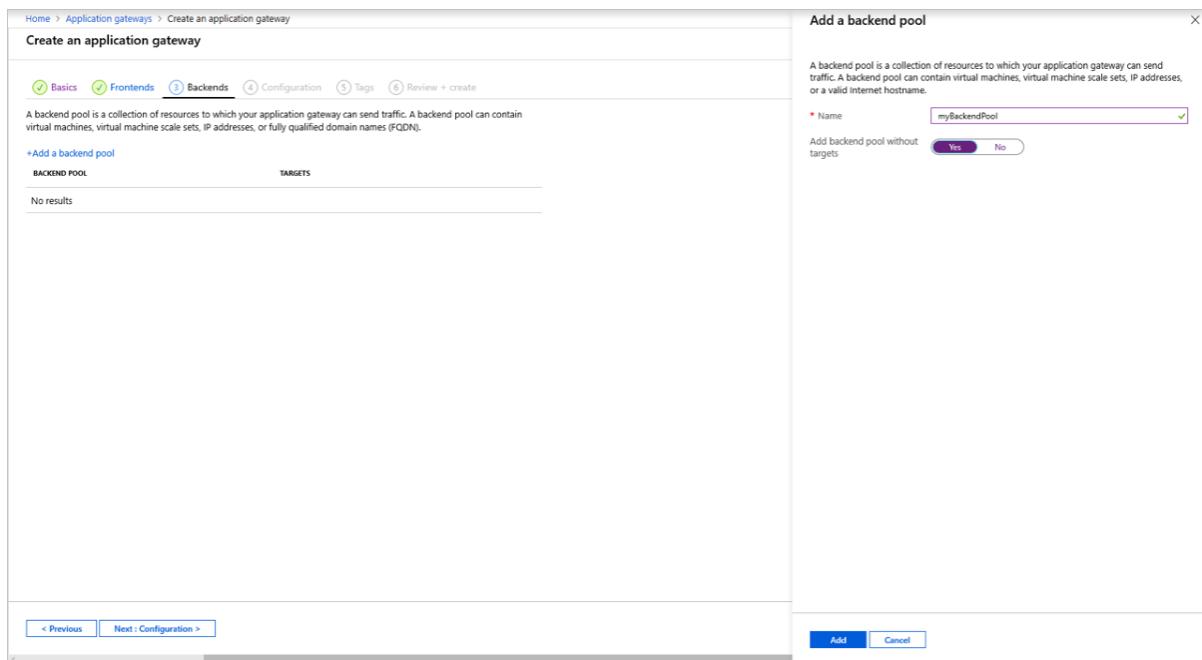


3. Select **Next: Backends**.

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

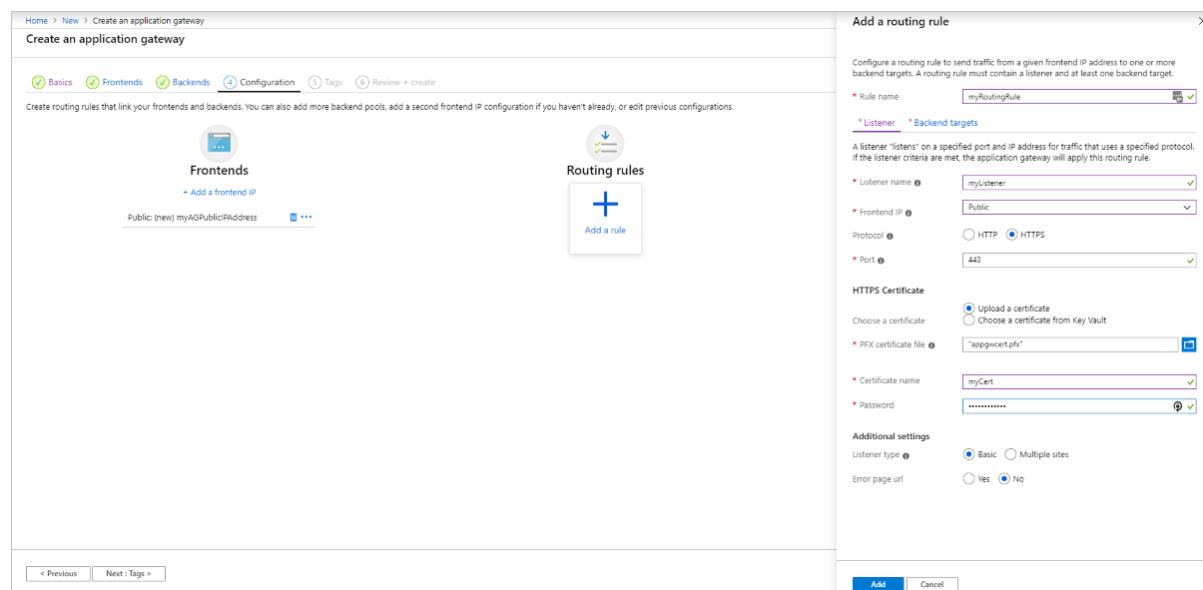
On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule.

1. Select **Add a 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 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 *Azure123456* for the password.

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 **Create 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 *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.

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. 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.

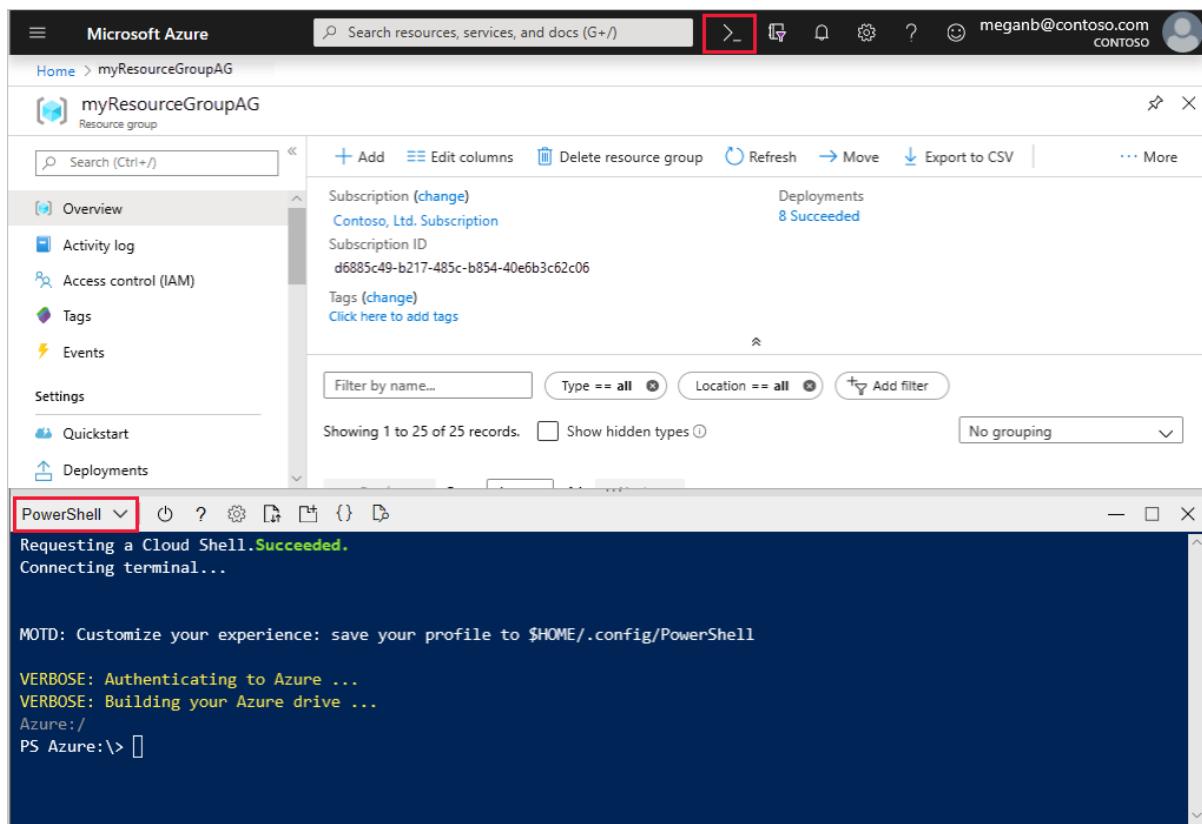
3. Enter these values in the **Basics** tab for the following virtual machine settings:
 - **Resource group**: Select **myResourceGroupAG** for the resource group name.
 - **Virtual machine name**: Enter *myVM* for the name of the virtual machine.
 - **Username**: Enter *azureuser* for the administrator user name.
 - **Password**: Enter *Azure123456* for the administrator password.
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 **Off**. 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.



- 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 EastUS
```

- 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

- Select **All resources**, and then select **myAppGateway**.
- Select **Backend pools** from the left menu.
- Select **myBackendPool**.
- Under **Targets**, select **Virtual machine** from the drop-down list.
- Under **VIRTUAL MACHINE** and **NETWORK INTERFACES**, select the **myVM** and **myVM2** virtual machines and their associated network interfaces from the drop-down lists.

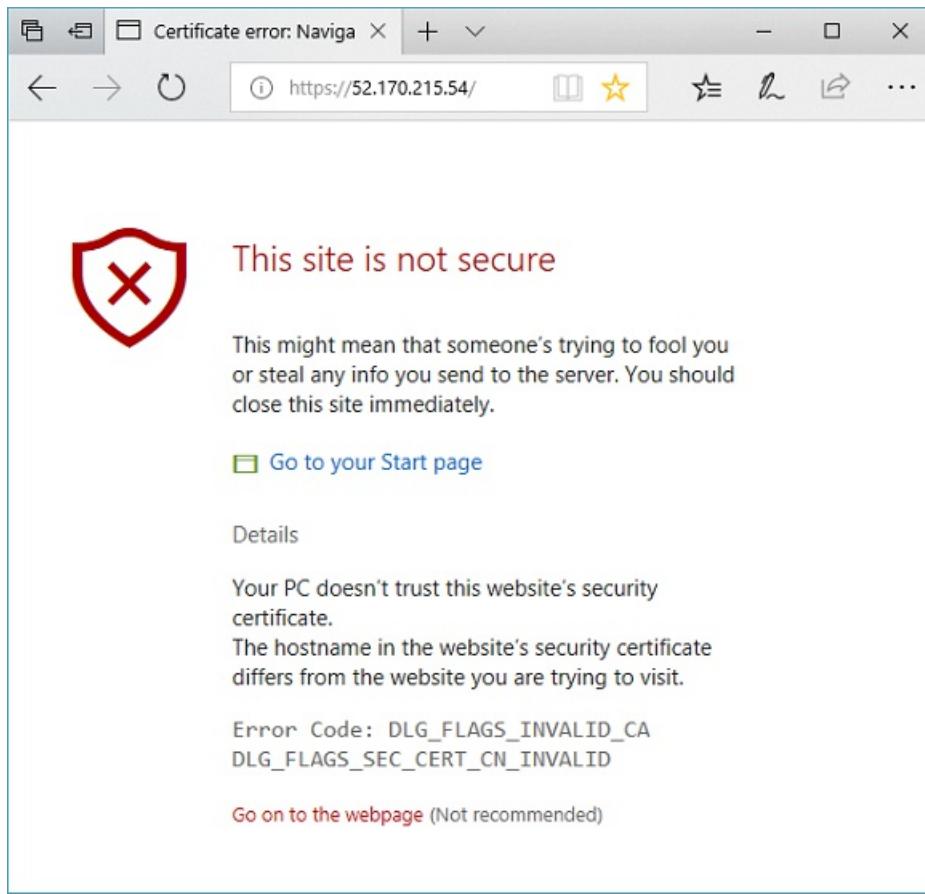
6. Select **Save**.
7. Wait for the deployment to complete before proceeding to the next step.

Test the application gateway

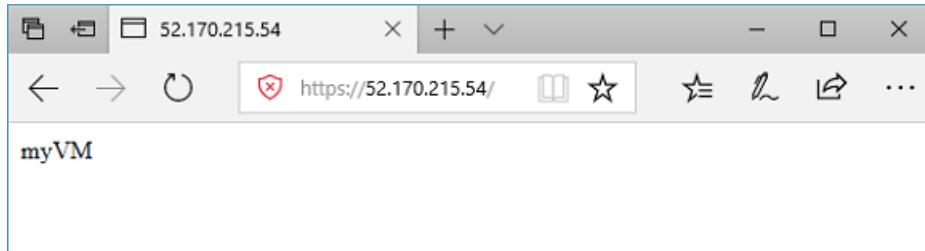
1. Select **All resources**, and then select **myAGPublicIPAddress**.

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:



Next steps

[Learn more about Application Gateway SSL support](#)

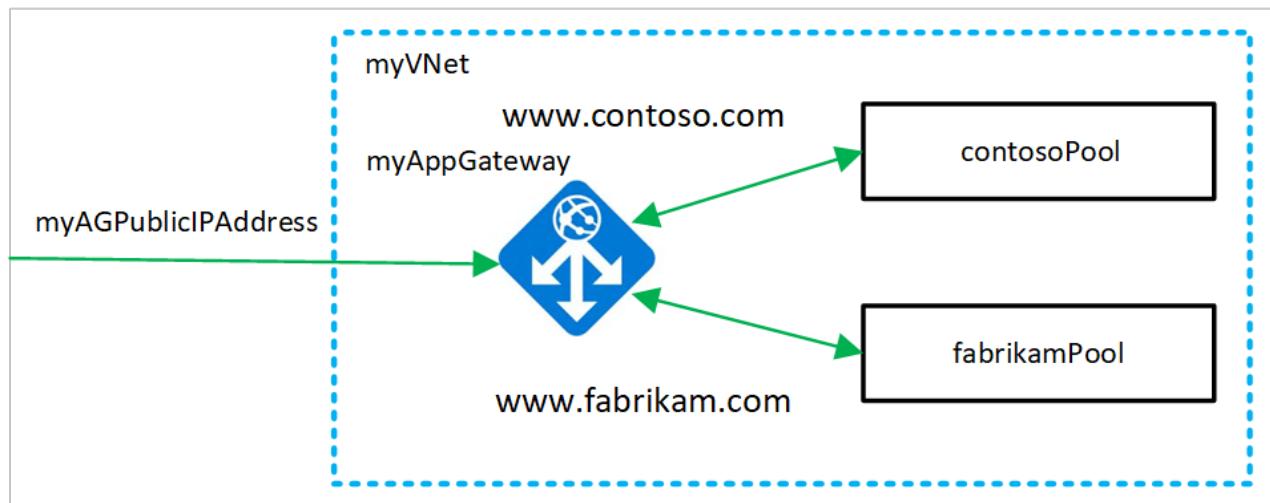
Tutorial: Create and configure an application gateway to host multiple web sites using the Azure portal

11/13/2019 • 8 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 domains that you own to make sure web traffic arrives at the appropriate servers in the pools. This tutorial assumes that you own multiple domains and 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 backend listeners
- Create routing rules
- Create a CNAME record in your domain



If you don't have an Azure subscription, create a [free account](#) before you begin.

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>

Create an application gateway

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

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.

The Microsoft Azure Virtual Network service enables Azure resources to securely communicate with each other in a virtual network which is a logical isolation of the Azure cloud dedicated to your subscription. You can connect virtual networks to other virtual networks, or on-premises network. [Learn more](#)

* Name: myVNet

ADDRESS SPACE
The virtual network's address space, specified as one or more address prefixes in CIDR notation (e.g. 192.168.1.0/24).

ADDRESS RANGE	ADDRESSES	OVERLAP
10.21.0.0/16	10.21.0 - 10.21.255.255 (65536 addresses)	None
	(0 Addresses)	None

SUBNETS
The subnet's address range in CIDR notation. It must be contained by the address space of the virtual network.

SUBNET NAME	ADDRESS RANGE	ADDRESSES
myAGSubnet	10.21.0.0/24	10.21.0 - 10.21.0.255 (256 addresses)
myBackendSubnet	10.21.1.0/24	10.21.1.0 - 10.21.1.255 (256 addresses)
	(0 Addresses)	

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

Frontends tab

- 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.

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

Traffic enters the application gateway via its frontend IP address. An application gateway can use a public IP address, private IP address, or one of each type.

Frontend IP address type: Public Private Both

* Public IP address: Choose public IP address
Create new

Add a public IP address

* Name: myAGPublicIPAddress
SKU: Standard
Assignment: Static

OK Cancel

- 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 screenshot shows the 'Create an application gateway' wizard with the 'Backends' tab selected. The top navigation bar includes 'Home > New > Create an application gateway'. Below the title 'Create an application gateway' are six tabs: Basics (green checkmark), Frontends (green checkmark), Backends (blue circle with 3), Configuration (grey), Tags (grey), and Review + create (grey). A descriptive text explains that a backend pool is a collection of resources to which your application gateway can send traffic. It lists two backend pools: 'contosoPool' and 'fabrikamPool', both currently having 0 targets. Each pool has a '...' button next to it.

5. 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 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:
 - **Listener name:** Enter *contosoListener* for the name of the listener.
 - **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.

The screenshot shows the Azure portal interface for creating an Application Gateway. On the left, the 'Create an application gateway' wizard is open, showing the 'Configuration' tab selected. It displays sections for 'Frontends' (with a 'Public' frontend), 'Backends' (with a 'contosoPool' backend pool), and 'Routing rules'. A 'Next : Tags >' button is visible at the bottom. On the right, a modal window titled 'Add a routing rule' is displayed. This window allows configuring a routing rule with fields for 'Rule name' (set to 'contosoRule'), 'Listener' (set to 'contosoListener'), 'Frontend IP' (set to 'Public'), 'Protocol' (set to 'HTTP'), 'Port' (set to '80'), 'Additional settings' (set to 'Multiple sites'), 'Host name' (set to 'www.contoso.com'), and 'Error page url' (set to 'No'). At the bottom of the modal are 'Add' and 'Cancel' buttons.

4. On the **Backend targets** tab, select **contosoPool** for the **Backend target**.
5. For the **HTTP setting**, select **Create 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 rule** and add a similar rule, listener, backend target, and HTTP setting for Fabrikam.

Add a routing rule

Configure a routing rule to send traffic from a given frontend IP address to one or more backend targets. A routing rule must contain a listener and at least one backend target.

*** Rule name** fabrikamRule ✓

*** Listener** * Backend targets

A listener "listens" on a specified port and IP address for traffic that uses a specified protocol. If the listener criteria are met, the application gateway will apply this routing rule.

*** Listener name** fabrikamListener ✓

*** Frontend IP** Public

Protocol HTTP HTTPS

*** Port** 80

Additional settings

Listener type Basic Multiple sites

*** Host name** www.fabrikam.com ✓

Error page url Yes No

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 **Compute** and then 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:

- **Resource group:** Select **myResourceGroupAG** for the resource group name.
- **Virtual machine name:** Enter **contosoVM** for the name of the virtual machine.
- **Username:** Enter **azureuser** for the administrator user name.
- **Password:** Enter **Azure123456!** for the administrator password.

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 **Off**. Accept the other defaults and then select **Review + create**.

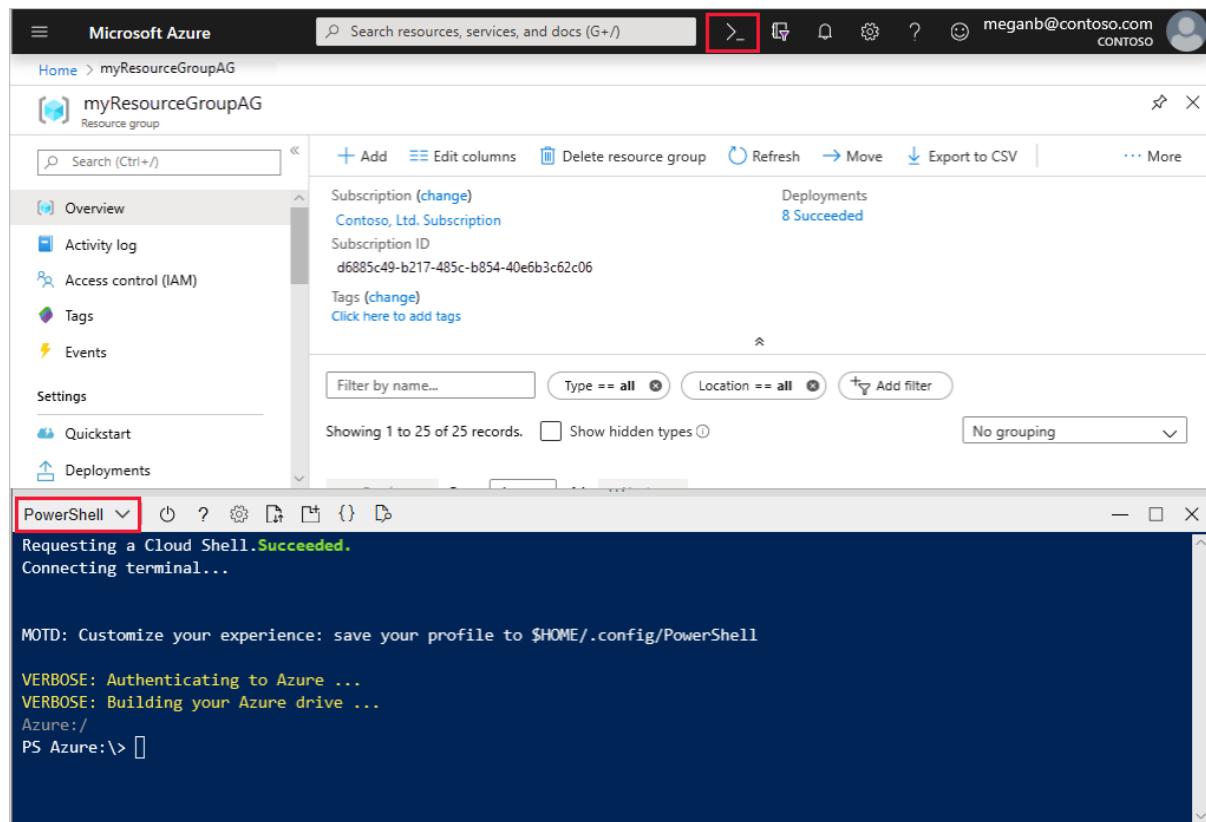
8. On the **Review + create** tab, review the settings, correct any validation errors, and then select **Create**.

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:

```

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 EastUS

```

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 **Targets**, select **Virtual machine** from the drop-down list.
5. Under **VIRTUAL MACHINE** and **NETWORK INTERFACES**, select the **contosoVM** virtual machine and its associated network interface from the drop-down lists.

Edit backend pool

Save Discard Delete

Name
contosoPool
 Remove all targets from backend pool

Targets
A backend pool can be pointed to a specific virtual machine, a virtual machine scale set, an IP address, a valid Internet hostname, or an app service.

Virtual machine ▼

VIRTUAL MACHINE	NETWORK INTERFACES
contosoVM ...	contosovm42 ...
Select a virtual machine ▼	Waiting for virtual machine selection ▼

Associated rule
contosoRule

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.

Create a www A record in your domains

After the application gateway is created with its public IP address, you can get the IP address and use it to create an A record in your domains.

1. Click **All resources**, and then click **myAGPublicIPAddress**.

The screenshot shows the Azure portal interface for a Public IP address named 'myAGPublicIPAddress'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Settings, and Configuration. The main pane displays resource details: Resource group (myResourceGroupAG), Location (East US), Subscription (VEH Doc Test), Subscription ID, Tags, and a note to 'Click here to add tags'. At the top right, there are buttons for Associate, Dissociate, Move, Delete, and Refresh. A red box highlights the 'IP address' field, which contains the value '52.191.218.52'.

2. Copy the IP address and use it as the value for a new www A record in your domains.

Test the application gateway

1. Enter your domain name into the address bar of your browser. Such as, <http://www.contoso.com>.

The screenshot shows a Microsoft Edge browser window. The address bar has 'contoso.com' in the host field and 'www.contoso.com/' in the path field. Below the browser is a text area labeled 'contosoVM'.

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

The screenshot shows a Microsoft Edge browser window. The address bar has 'fabrikam.com' in the host field and 'www.fabrikam.com/' in the path field. Below the browser is a text area labeled 'fabrikamVM'.

Clean up resources

When you no longer need the resources that you created with the application gateway, remove the resource group. When you remove the resource group, you also remove 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 **Resource group page**, select **Delete resource group**.
4. Enter *myResourceGroupAG* for **TYPE THE RESOURCE GROUP NAME** and then select **Delete**.

Next steps

[Learn more about what you can do with Azure Application Gateway](#)

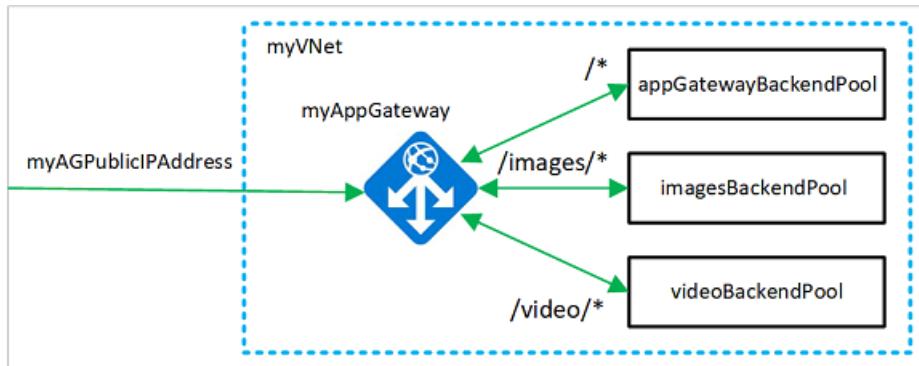
Tutorial: Create an application gateway with path-based routing rules using the Azure portal

11/13/2019 • 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



If you don't have an Azure subscription, create a [free account](#) before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>

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. On the Azure portal, select **Create a resource**.
2. Select **Windows Server 2016 Datacenter** in the Popular list.
3. Enter these values for the virtual machine:

- **Resource group**, select **Create new**, and then type *myResourceGroupAG*.
- **Virtual machine name**: *myVM1*
- **Region**: *(US) East US*
- **Username**: *azureuser*
- **Password**: *Azure123456!*

4. Select **Next:Disks**.

5. Select **Next:Networking**

6. 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.

7. Select **OK**.

8. Ensure that under **Network Interface**, **myBackendSubnet** is selected for the subnet, and then select **Next: Management**.

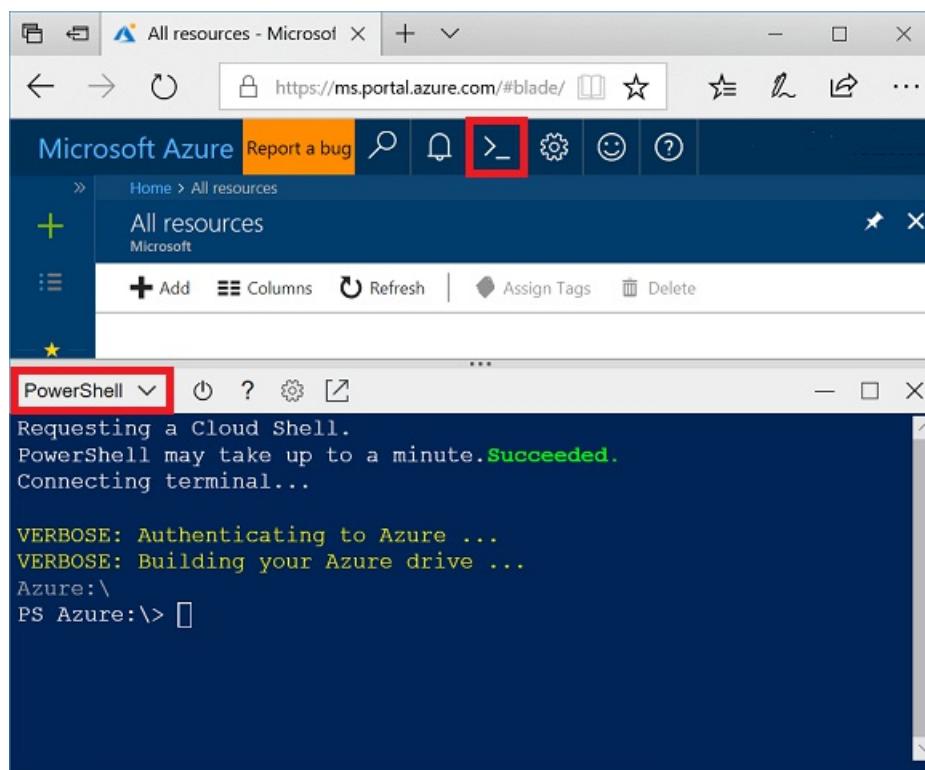
9. Select **Off** to disable boot diagnostics.

10. Click **Review + Create**, review the settings on the summary page, and then select **Create**.

11. 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

```

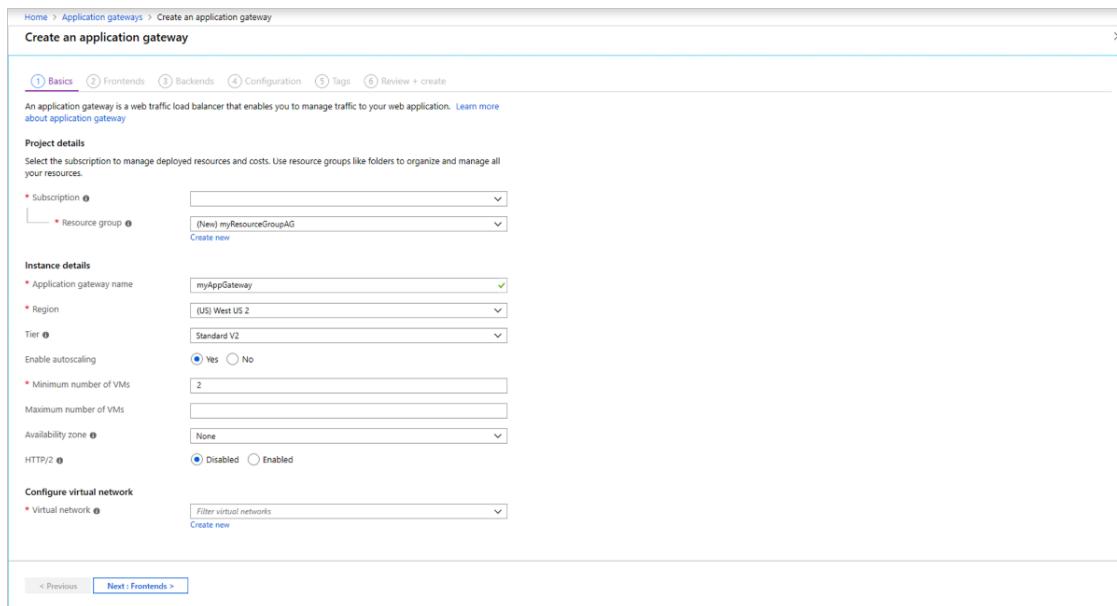
3. Create two more virtual machines and install IIS using the steps that you just finished. Enter the names of *myVM2* and *myVM3* for the names and for the values of VMName in Set-AzVMExtension.

Create an application gateway

1. Select **Create a resource** on the left menu of the Azure portal. 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.
 - Application gateway name:** Enter *myAppGateway* for the name of the application gateway.
 - Region - Select (US) 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**.

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. Choose **Create 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 **Backend Targets, 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 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.
 - **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 **Create 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 **Path rule 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.

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**.

The screenshot shows the Azure portal interface for the 'myAppGateway' application gateway. The top navigation bar includes 'Home', 'Resource groups', 'myResourceGroupAG3', and 'myAppGateway'. The main content area displays the 'Overview' tab of the application gateway. Key details shown include:

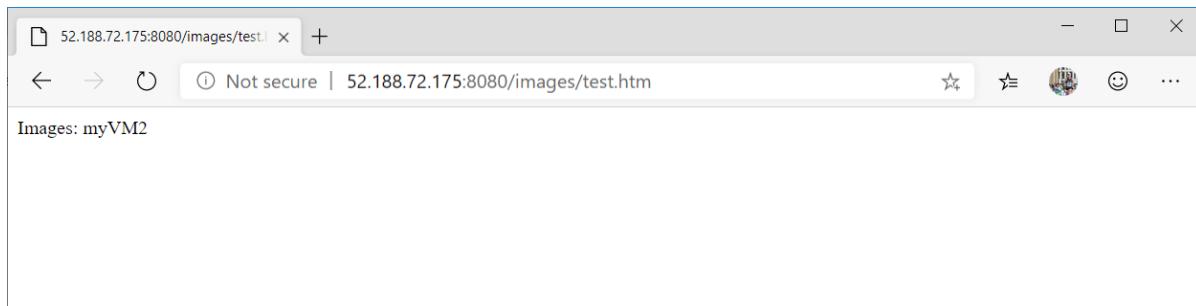
- Resource group: myResourceGroupAG3
- Location: East US
- Subscription: VEH Doc Test
- Subscription ID: f054d65b-172e-41e0-b0cb-cdd68243d492
- Virtual network/subnet: myVNet/myAGSubnet
- Frontend public IP address: 52.188.72.175 (myAGPublicIPAddress)
- Frontend private IP address: -
- Tier: Standard V2
- Tags: Click here to add tags

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 screenshot shows a web browser window with the URL <http://52.188.72.175:8080>. The page content displays the text "MyVM1".

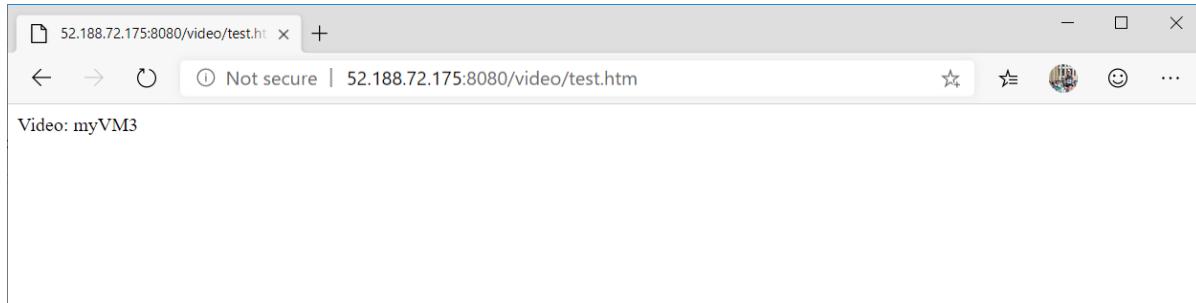
The listener on port 8080 routes this request to the default backend pool.

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



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

Next steps

- [Enabling end to end SSL on Azure Application Gateway](#)

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

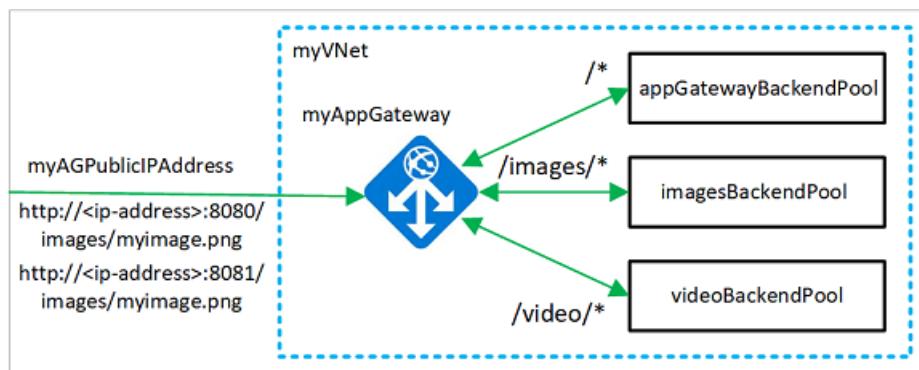
11/13/2019 • 6 minutes to read • [Edit Online](#)

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 a [free account](#) before you begin.

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.	<div style="border: 1px solid #ccc; padding: 5px; display: flex; align-items: center;">Azure CLI Copy Try It</div>
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	<div style="border: 1px solid #0072bc; border-radius: 5px; padding: 5px; display: flex; align-items: center; gap: 10px;">△ Launch Cloud Shell</div>

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.

If you choose to install and use the CLI locally, this tutorial requires you to run the Azure CLI version 2.0.4 or later.

To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

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.

```

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

```

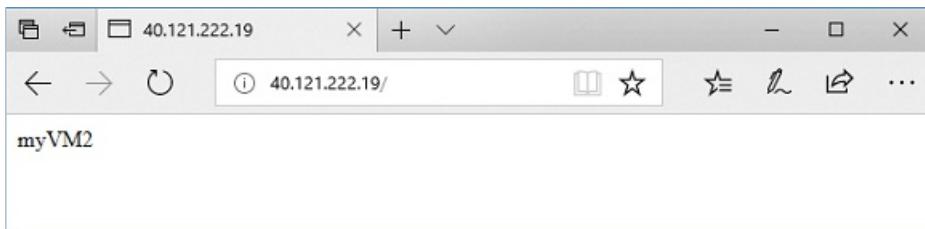
Test the application gateway

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`, `http://40.121.222.19:8080/video/test.htm`, or `http://40.121.222.19:8081/images/test.htm`.

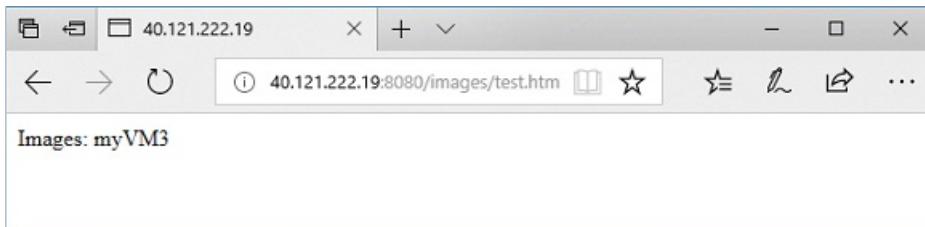
```

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

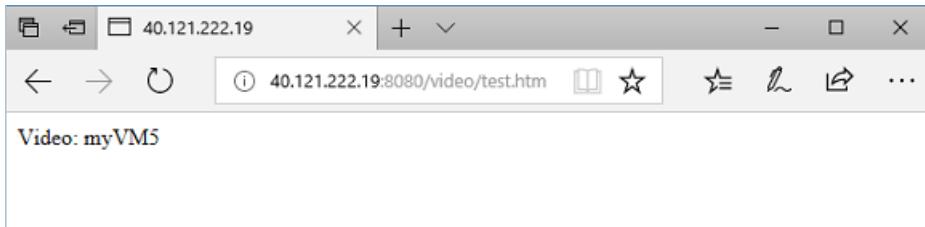
```



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

11/12/2019 • 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

This tutorial requires that you run Azure PowerShell 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:

```
$pwd = ConvertTo-SecureString -String "Azure123456!" -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
```

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, 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.

```
$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 "SSLPor" -Port 443
$fp02 = New-AzApplicationGatewayFrontendPort -Name "HTTPPor" -Port 80

$securepxpwd = ConvertTo-SecureString -String "Azure123456!" -AsPlainText -Force
$sslCert01 = New-AzApplicationGatewaySslCertificate -Name "SSLCert" -Password $securepxpwd ` 
    -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
$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. 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, $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.

```
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](#)

Azure PowerShell examples for Azure Application Gateway

11/15/2019 • 2 minutes to read • [Edit Online](#)

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

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

11/15/2019 • 2 minutes to read • [Edit Online](#)

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

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

11/15/2019 • 2 minutes to read • [Edit Online](#)

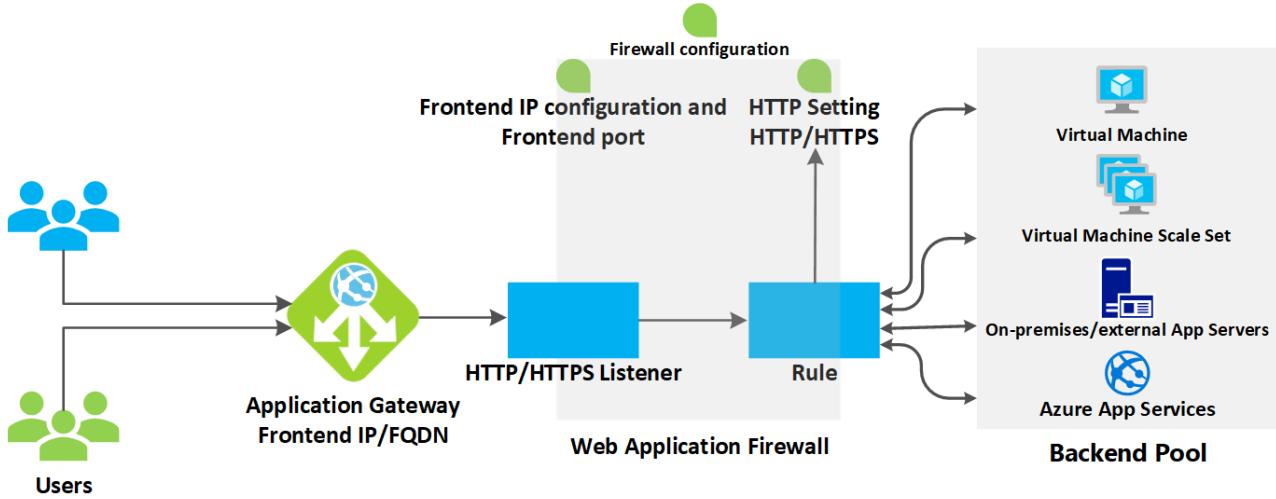
The following table includes links to Azure Resource Manager templates for Azure Application Gateway.

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

How an application gateway works

11/15/2019 • 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 to 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). 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 SSL) 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

An application gateway inserts four additional headers to all requests before it forwards the requests to the backend. These headers are x-forwarded-for, x-forwarded-proto, x-forwarded-port, and x-original-host. 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.

You can configure application gateway to modify headers by using [Rewrite HTTP headers](#) 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.

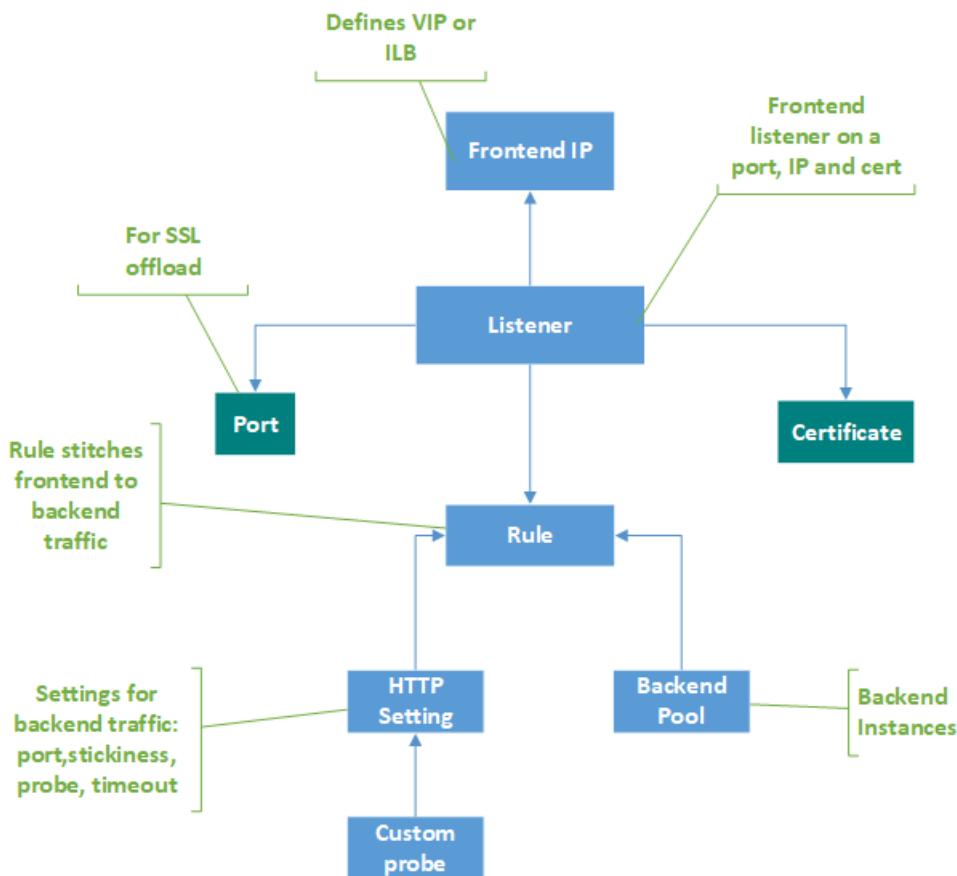
Next steps

[Learn about application gateway components](#)

Application gateway components

12/9/2019 • 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 ranging from 1 to 65502 for the v1 SKU and 1 to 65199 for the v2 SKU.

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 SSL 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 configure more than one web application on the same application gateway instance. 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 subdomains, abc.contoso.com, xyz.contoso.com, and pqr.contoso.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.

For more information, see [Multiple-site hosting](#).

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.

Application Gateway processes listeners in the [order shown](#).

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

By using the request routing rules, you can add, remove, or update HTTP(S) request and response headers as the request and response packets move between the client and backend pools via the application gateway.

The headers 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 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 SSL) 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 the probe interval, the URL and path to test, and how many failed responses to accept before the backend pool instance is marked as unhealthy. 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

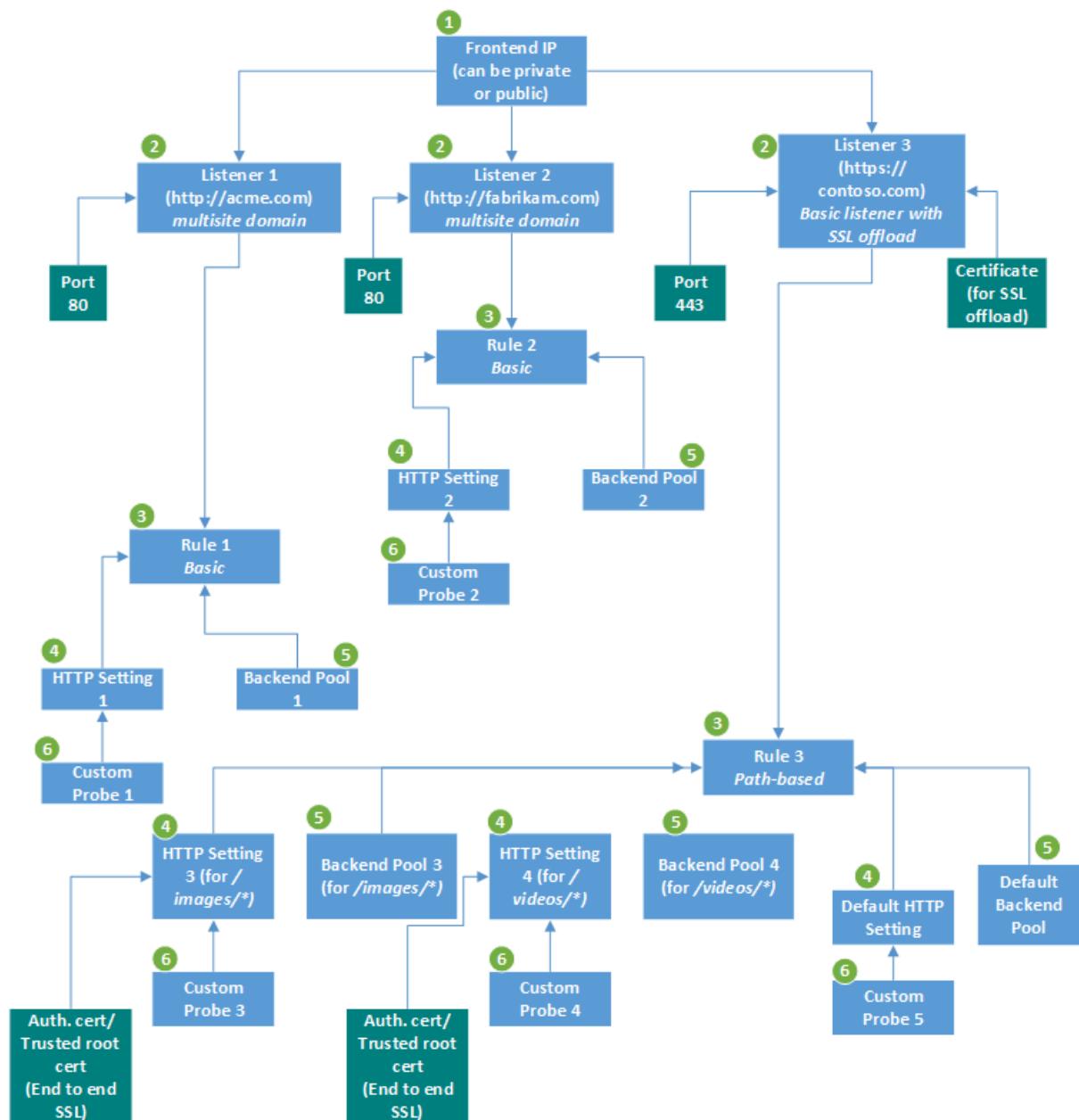
Create an application gateway:

- [In the Azure portal](#)
- [By using Azure PowerShell](#)
- [By using the Azure CLI](#)

Application Gateway configuration overview

2/27/2020 • 21 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 Secure Sockets Layer (SSL) termination.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Prerequisites

Azure 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.

NOTE

You can't mix Standard_v2 and Standard Azure Application Gateway on the same subnet.

Size of the subnet

Application Gateway consumes 1 private IP address per instance, plus another private IP address if a private front-end IP is configured.

Azure also reserves 5 IP addresses in each subnet for internal use: the first 4 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: 5 for internal use and 15 for the application gateway instances. So, you need a /27 subnet size or larger.

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, 1 for the private front end, and 5 for internal use. So, you need a /26 subnet size or larger.

We recommend that you use a subnet size of at least /28. This size gives you 11 usable IP addresses. If your application load requires more than 10 Application Gateway instances, consider a /27 or /26 subnet size.

Network security groups on the Application Gateway subnet

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 must be allowed.

Allow Application Gateway 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) and inbound virtual network traffic (*VirtualNetwork* tag) on the [network security group](#).
4. Block all other incoming traffic by using a deny-all rule.
5. Allow outbound traffic to the internet for all destinations.

User-defined routes supported on the Application Gateway subnet

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.

For the v2 SKU, UDRs are not supported on the Application Gateway subnet. For more information, see [Azure Application Gateway v2 SKU](#).

NOTE

UDRs are not supported for the v2 SKU as of now.

NOTE

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.

Front-end IP

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).

A public IP 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 SSL termination.

Only 1 public IP address or 1 private IP address is supported. You choose the front-end IP when you create the application gateway.

- For a public IP, 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, you can specify a private IP address from the subnet where the application gateway is created. If you don't specify one, an arbitrary 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.

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.

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 hostname, choose multi-site listener, where you must also specify a hostname 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.

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 is 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

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 [SSL termination](#) or [end-to-end SSL encryption](#). The traffic between the client and the application gateway is encrypted. And the SSL connection terminates at the application gateway. If you want end-to-end SSL 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 SSL termination and end-to-end SSL encryption, you must add a certificate to the listener to enable the application gateway to derive a symmetric key. This is dictated by the SSL protocol specification. The symmetric key is used to encrypt and decrypt the traffic that's sent to the gateway. The gateway certificate must be in Personal Information Exchange (PFX) format. This format lets you export the private key that the gateway uses to encrypt and decrypt traffic.

Supported certificates

See [certificates supported for SSL termination](#).

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](#).

The screenshot shows the Azure portal interface for managing an Application Gateway listener. The top navigation bar includes 'Home', 'Application gateways', 'TestAppGwv2 - Listeners', and 'appGatewayHttpListener'. The main section displays the configuration for 'appGatewayHttpListener' under 'TestAppGwv2'. Key fields include:

- Name:** appGatewayHttpListener
- Frontend IP configuration:** appGatewayFrontendIP
- Frontend port:** appGatewayFrontendPort (443)
- Protocol:** HTTPS
- Certificate:** scrap

Under the 'CUSTOM ERROR PAGES' section, there is a table:

ERROR CODE	URL
Forbidden - 403	https://mycustomerrorpages.blob.core.windows.net/
Bad Gateway - 502	https://mycustomerrorpages.blob.core.windows.net/

To configure a global custom error page, see [Azure PowerShell configuration](#).

SSL policy

You can centralize SSL certificate management and reduce encryption-decryption overhead for a back-end server farm. Centralized SSL handling also lets you specify a central SSL policy that's suited to your security requirements. You can choose *default*, *predefined*, or *custom* SSL policy.

You configure SSL policy to control SSL 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 SSL 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.

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.

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 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.

For the v2 SKU, an exact match is higher priority than path order in the URL path map. If a request matches the pattern in two or more paths, the request is forwarded to the back end that's associated with the path that exactly matches the request. If the path in the incoming request doesn't exactly match any path in the map, pattern matching of the request is processed in the path map order list for the path-based rule.

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.

The screenshot shows the 'rule1' configuration page in the Azure portal. The 'rule1' section is highlighted with a red border. Inside, the 'Listener' dropdown is set to 'appGatewayHttpListener'. The 'Configure redirection' checkbox is checked. The 'Redirection type' dropdown is set to 'Permanent'. The 'Redirection target' section shows 'Listener' selected. The 'Target listener' dropdown is set to 'SecureListener'. Below these fields are two unchecked checkboxes: 'Include query string' and 'Include path'.

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 the HTTP header setting

This setting adds, removes, or updates HTTP request and response headers while the request and response packets move between the client and back-end pools. For more information, see:

- [Rewrite HTTP headers overview](#)
- [Configure HTTP header rewrite](#)

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.

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.

The [Chromium browser v80 update](#) brought a mandate where HTTP cookies without `SameSite` attribute has 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 a 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 17th 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 session are maintained even for cross-origin requests.

Note that the default affinity cookie name is `ApplicationGatewayAffinity` and you can change it. In case you are 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 SSL offload and End-to-End SSL documentation for Application Gateway here – [Overview](#), [How-to configure SSL offload](#), [How-to configure End-to-End SSL](#).

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 during rule creation. 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 SSL](#). 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 SSL 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.

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 for app service

This is a UI only shortcut that selects the two required settings for the Azure App Service back end. It enables **pick host name from back-end address**, and it creates a new custom probe if you don't have one already. (For more information, see the [Pick host name from back-end address](#) setting section of this article.) A new probe is created, and the probe header is picked from the back-end member's address.

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.

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 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, you don't have to enable this setting.

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.

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. Each back-end pool can point to multiple members of the same type. Pointing to members of different types in the same back-end pool isn't supported.

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 by using PowerShell](#)
- [Create an application gateway by using the Azure CLI](#)

Autoscaling and Zone-redundant Application Gateway v2

2/26/2020 • 10 minutes to read • [Edit Online](#)

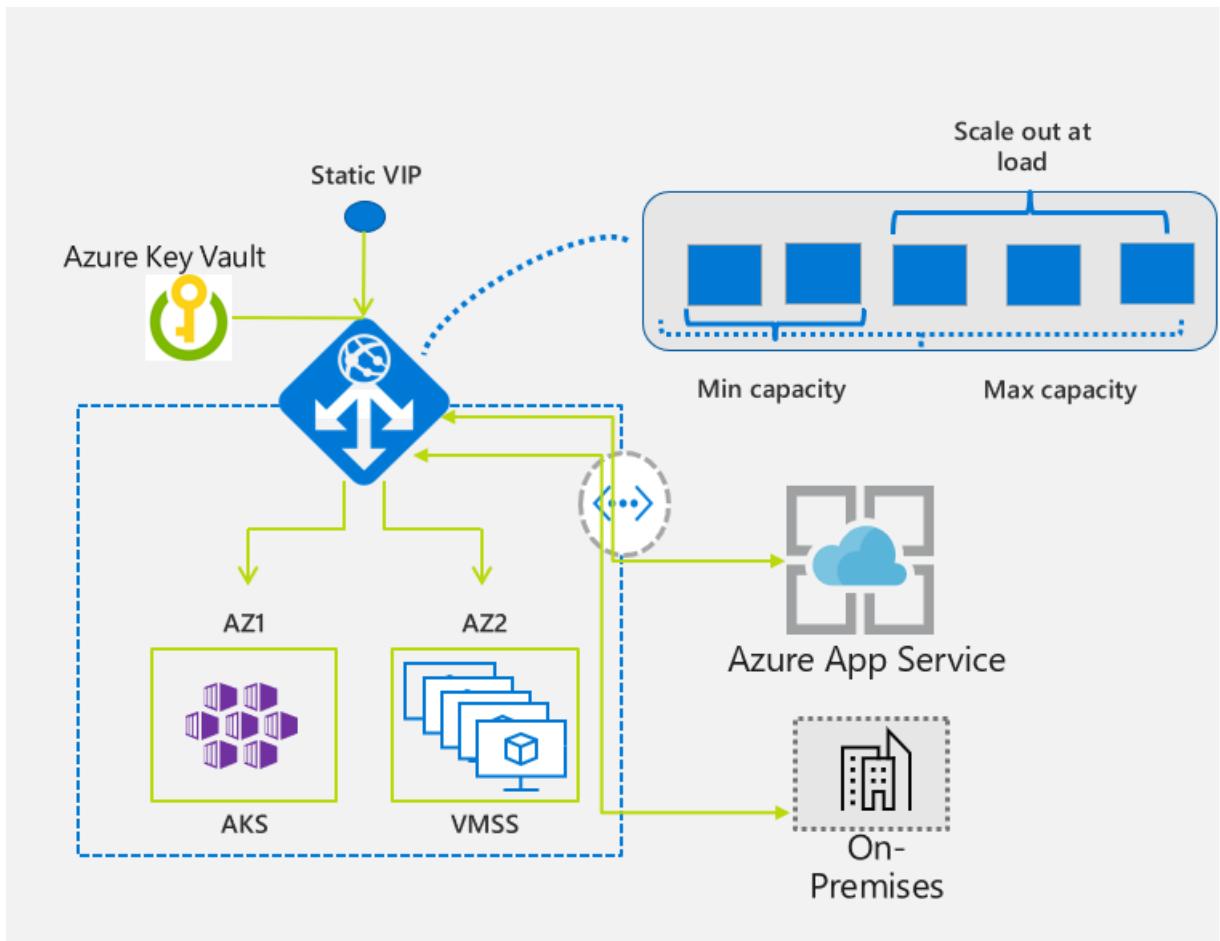
Application Gateway and Web Application Firewall (WAF) are also available under a Standard_v2 and WAF_v2 SKU. The v2 SKU offers performance enhancements and adds support for critical new features like autoscaling, zone redundancy, and support for static VIPs. Existing features under the Standard and 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 up or down 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 [What are 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 [SSL termination with Key Vault certificates](#).
- **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?](#).
- **Performance enhancements:** The v2 SKU offers up to 5X better SSL 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.



Supported regions

The Standard_v2 and WAF_v2 SKU is available in the following regions: North Central US, South Central US, West US, West US 2, East US, East US 2, Central US, North Europe, West Europe, Southeast Asia, France Central, UK West, Japan East, Japan West, Australia East, Australia Southeast, Brazil South, Canada Central, Canada East, East Asia, Korea Central, Korea South, UK South, Central India, West India, South India.

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, or 2500 persistent connections, or 2.22-Mbps throughput.

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.

NOTE

Each instance can currently support approximately 10 capacity units. 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. We recommend you perform application tests to determine request rate per compute unit. Both capacity unit and compute unit will be made available as a metric before billing starts.

The following table shows example prices and are for illustration purposes only.

Pricing in US East:

SKU NAME	FIXED PRICE (\$/HR)	CAPACITY UNIT PRICE (\$/CU-HR)
Standard_v2	0.20	0.0080
WAF_v2	0.36	0.0144

For more pricing information, see the [pricing page](#).

Example 1

An Application Gateway Standard_v2 is provisioned without autoscaling in manual scaling mode with fixed capacity of five instances.

$$\text{Fixed price} = 744(\text{hours}) * \$0.20 = \$148.8$$

$$\text{Capacity units} = 744 \text{ (hours)} * 10 \text{ capacity unit per instance} * 5 \text{ instances} * \$0.008 \text{ per capacity unit hour} = \$297.6$$

$$\text{Total price} = \$148.8 + \$297.6 = \$446.4$$

Example 2

An Application Gateway standard_v2 is provisioned for a month, with zero minimum instances, and during this time it receives 25 new SSL connections/sec, average of 8.88-Mbps data transfer. Assuming connections are short lived, your price would be:

$$\text{Fixed price} = 744(\text{hours}) * \$0.20 = \$148.8$$

$$\text{Capacity unit price} = 744(\text{hours}) * \text{Max (25/50 compute unit for connections/sec, 8.88/2.22 capacity unit for throughput)} * \$0.008 = 744 * 4 * 0.008 = \$23.81$$

$$\text{Total price} = \$148.8 + \$23.81 = \$172.61$$

As you can see, you are only billed for four Capacity Units, not for the entire instance.

NOTE

The Max function returns the largest value in a pair of values.

Example 3

An Application Gateway standard_v2 is provisioned for a month, with a minimum of five instances. Assuming that there is no traffic and connections are short lived, your price would be:

Fixed price = 744(hours) * \$0.20 = \$148.8

Capacity unit price = 744(hours) * Max (0/50 compute unit for connections/sec, 0/2.22 capacity unit for throughput) * \$0.008 = 744 * 50 * 0.008 = \$297.60

Total price = \$148.80 + 297.60 = \$446.4

In this case, you're billed for the entirety of the five instances even though there is no traffic.

Example 4

An Application Gateway standard_v2 is provisioned for a month, with a minimum of five instances, but this time there is an average of 125-mbps data transfer, and 25 SSL connections per second. Assuming that there is no traffic and connections are short lived, your price would be:

Fixed price = 744(hours) * \$0.20 = \$148.8

Capacity unit price = 744(hours) * Max (25/50 compute unit for connections/sec, 125/2.22 capacity unit for throughput) * \$0.008 = 744 * 57 * 0.008 = \$339.26

Total price = \$148.80 + 339.26 = \$488.06

In this case, you are billed for the full five instances, plus seven Capacity Units (which is 7/10 of an instance).

Example 5

An Application Gateway WAF_v2 is provisioned for a month. During this time, it receives 25 new SSL connections/sec, average of 8.88-Mbps data transfer and does 80 request per second. Assuming connections are short lived, and that compute unit calculation for the application supports 10 RPS per compute unit, your price would be:

Fixed price = 744(hours) * \$0.36 = \$267.84

Capacity unit price = 744(hours) * Max (compute unit Max(25/50 for connections/sec, 80/10 WAF RPS), 8.88/2.22 capacity unit for throughput) * \$0.0144 = 744 * 8 * 0.0144 = \$85.71

Total price = \$267.84 + \$85.71 = \$353.55

NOTE

The Max function returns the largest value in a pair of values.

Scaling Application Gateway and WAF v2

Application Gateway and WAF can be configured to scale in two modes:

- **Autoscaling** - With autoscaling enabled, the Application Gateway and WAF v2 SKUs scale up or down 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 anticipated 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 in the absence of traffic. Each instance counts as 10 additional reserved Capacity Units. Zero signifies no reserved capacity and is purely autoscaling in nature. Please note that zero additional minimum instances still ensures high availability of the service which is always included with fixed price. 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 continue to 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 alternatively choose Manual mode where the gateway won't autoscale. In this mode, if there is 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.

Feature comparison between v1 SKU and v2 SKU

The following table compares the features available with each SKU.

	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	✓	✓
Traffic redirection	✓	✓
Web Application Firewall (WAF)	✓	✓
WAF custom rules		✓
Secure Sockets Layer (SSL) termination	✓	✓
End-to-end SSL encryption	✓	✓
Session affinity	✓	✓
Custom error pages	✓	✓
WebSocket support	✓	✓
HTTP/2 support	✓	✓
Connection draining	✓	✓

NOTE

The autoscaling v2 SKU now supports [default health probes](#) to automatically monitor the health of all resources in its backend 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 with v1 SKU

DIFFERENCE	DETAILS
Authentication certificate	Not supported. For more information, see Overview of end to end SSL with Application Gateway .
Mixing Standard_v2 and Standard Application Gateway on the same subnet	Not supported
User Defined Route (UDR) on Application Gateway subnet	Not supported
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.
Netwatcher integration	Not supported.
Azure Security Center 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](#).

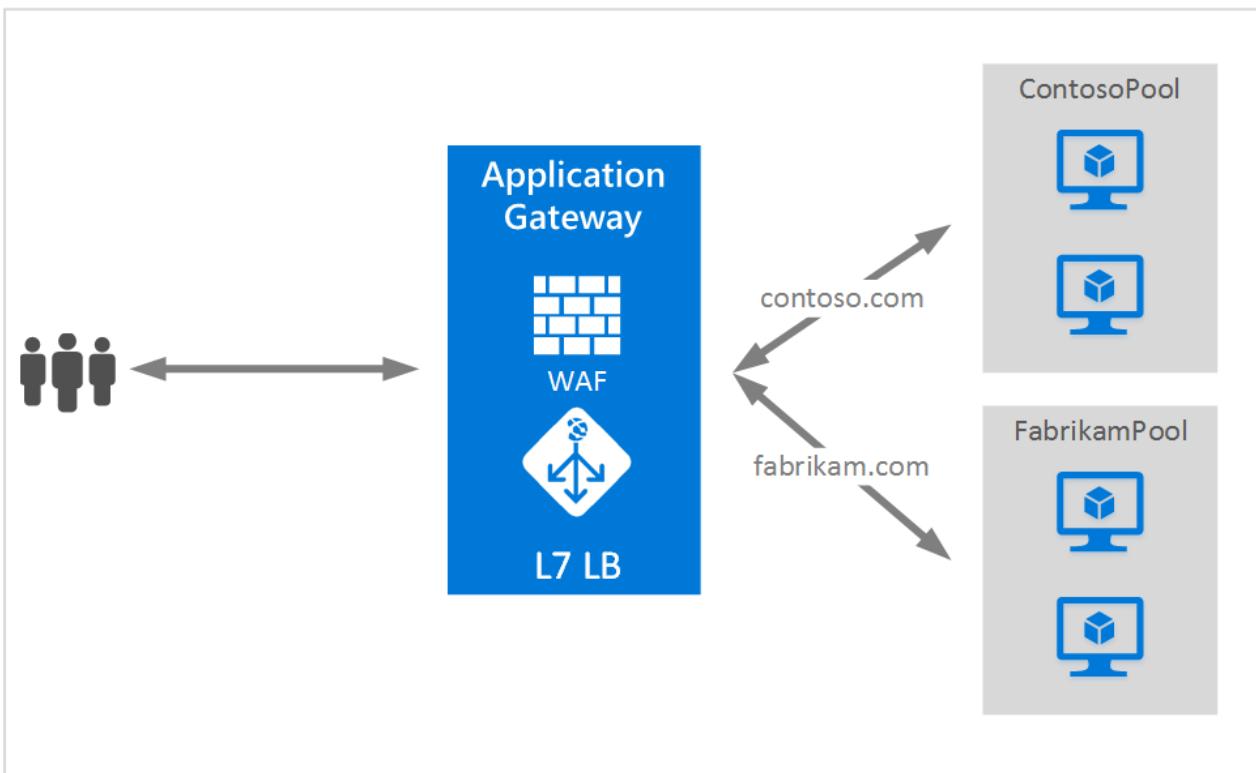
Next steps

- [Quickstart: Direct web traffic with Azure Application Gateway - Azure portal](#)
- [Create an autoscaling, zone redundant application gateway with a reserved virtual IP address using Azure PowerShell](#)
- Learn more about [Application Gateway](#).
- Learn more about [Azure Firewall](#).

Application Gateway multiple site hosting

2/21/2020 • 2 minutes to read • [Edit Online](#)

Multiple site hosting enables you to configure more than one web application on the same port of an application gateway. This feature 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. In the following example, application gateway is serving traffic for contoso.com and fabrikam.com from two back-end server pools called ContosoServerPool and FabrikamServerPool.



IMPORTANT

Rules are processed in the order they are listed in the portal for the v1 SKU. For the v2 SKU, exact matches have higher precedence. 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 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.

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 an application gateway gets a single public IP address on which it listens for traffic. Therefore supporting multiple applications, each with its own IP address, is currently not supported. Application Gateway supports hosting multiple applications each listening on different ports but this scenario would require the applications to accept traffic on non-standard ports and is often not a desired configuration. 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 SSL 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.

Listener configuration element

Existing HTTPListener configuration element is enhanced to support host name and server name indication elements, which is used by application gateway to route traffic to appropriate backend pool. The following code example is the snippet of HttpListeners element from template file.

```
"httpListeners": [
    {
        "name": "appGatewayHttpsListener1",
        "properties": {
            "FrontendIPConfiguration": {
                "Id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/frontendIPConfigurations/DefaultFrontendPublicIP"
            },
            "FrontendPort": {
                "Id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/frontendPorts/appGatewayFrontendPort443"
            },
            "Protocol": "Https",
            "SslCertificate": {
                "Id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/sslCertificates/appGatewaySslCert1"
            },
            "HostName": "contoso.com",
            "RequireServerNameIndication": "true"
        }
    },
    {
        "name": "appGatewayHttpListener2",
        "properties": {
            "FrontendIPConfiguration": {
                "Id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/frontendIPConfigurations/appGatewayFrontendIP"
            },
            "FrontendPort": {
                "Id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/frontendPorts/appGatewayFrontendPort80"
            },
            "Protocol": "Http",
            "HostName": "fabrikam.com",
            "RequireServerNameIndication": "false"
        }
    }
],
```

You can visit [Resource Manager template using multiple site hosting](#) for an end to end template-based

deployment.

Routing rule

There is no change required in the routing rule. The routing rule 'Basic' should continue to be chosen to tie the appropriate site listener to the corresponding backend address pool.

```
"requestRoutingRules": [
{
    "name": "<ruleName1>",
    "properties": {
        "RuleType": "Basic",
        "httpListener": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/httpListeners/appGatewayHttpsListener1')"
        },
        "backendAddressPool": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/backendAddressPools/ContosoServerPool')"
        },
        "backendHttpSettings": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/backendHttpSettingsCollection/appGatewayBackendHttpSettings')"
        }
    }
},
{
    "name": "<ruleName2>",
    "properties": {
        "RuleType": "Basic",
        "httpListener": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/httpListeners/appGatewayHttpListener2')"
        },
        "backendAddressPool": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/backendAddressPools/FabrikamServerPool')"
        },
        "backendHttpSettings": {
            "id": "/subscriptions/<subid>/resourceGroups/<rgName>/providers/Microsoft.Network/applicationGateways/applicationGateway1/backendHttpSettingsCollection/appGatewayBackendHttpSettings')"
        }
    }
}
]
```

Next steps

After learning about multiple site hosting, go to [create an application gateway using multiple site hosting](#) to create an application gateway with ability to support more than one web application.

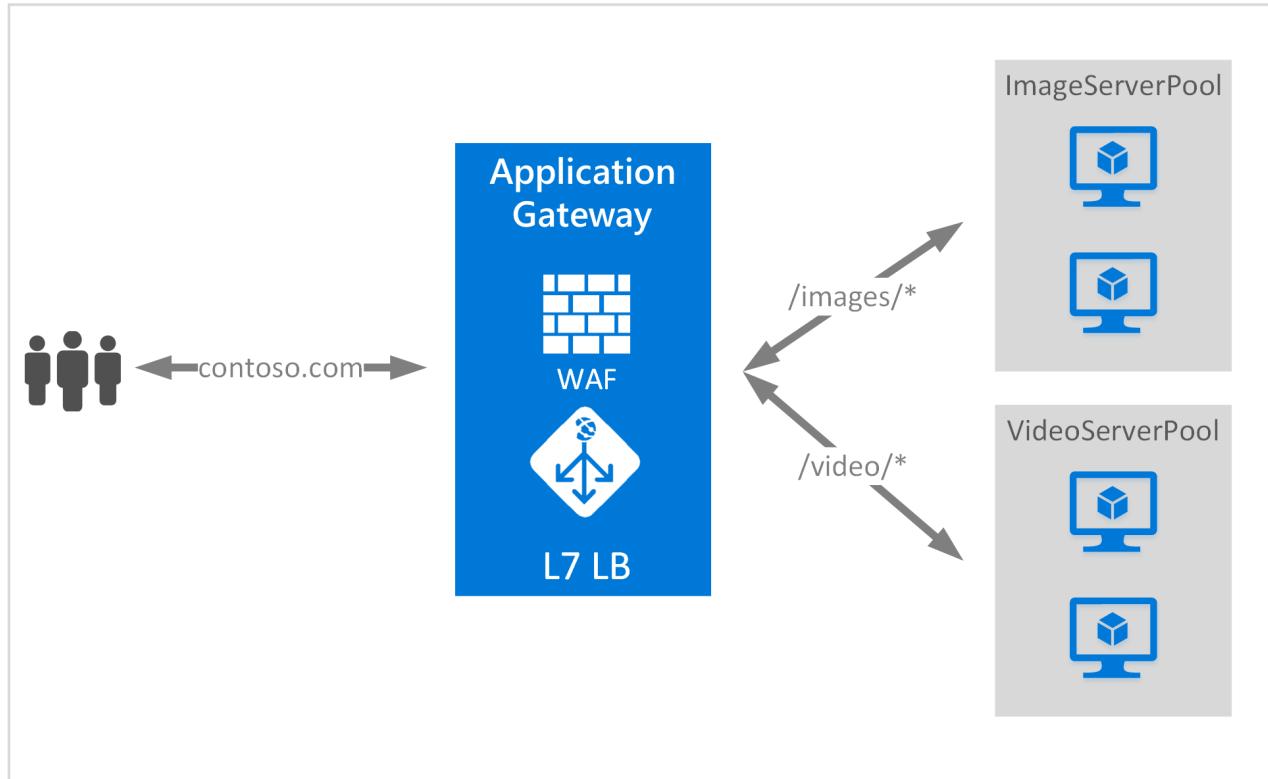
URL Path Based Routing overview

2/14/2020 • 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 the v1 SKU, rules are processed in the order they are listed in the portal. If a basic listener is listed first and matches an incoming request, it gets processed by that listener. For the v2 SKU, exact matches have higher precedence. However, it is 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/{urlpathMa
pName}",
        "properties": {
            "defaultBackendAddressPool": {
                "id": "/subscriptions/
{subscriptionId}.../microsoft.network/applicationGateways/{gatewayName}/backendAddressPools/{poolName1}"
            },
            "defaultBackendHttpSettings": {
                "id":
"/subscriptions/{subscriptionId}.../microsoft.network/applicationGateways/{gatewayName}/backendHttpSettingsList
t/{settingname1}"
            },
            "pathRules": [
                {
                    "name": "{pathRuleName}",
                    "properties": {
                        "paths": [
                            "{pathPattern}"
                        ],
                        "backendAddressPool": {
                            "id":
"/subscriptions/{subscriptionId}.../microsoft.network/applicationGateways/{gatewayName}/backendAddressPools/{p
oolName2}"
                        },
                        "backendHttpsettings": {
                            "id":
"/subscriptions/{subscriptionId}.../microsoft.network/applicationGateways/{gatewayName}/backendHttpsettingsList
t/{settingName2}"
                        }
                    }
                }
            ]
        }
    }
]
}

```

PathPattern

PathPattern is a list of path patterns to match. Each must start with / and the only place a "*" is allowed is at the end following a "/." The string fed to the path matcher does not include any text after the first ? or #, and those chars are not allowed here. Otherwise, any characters allowed in a URL are allowed in PathPattern.

The supported patterns depend on whether you deploy Application Gateway v1 or v2:

v1

Path rules are case insensitive.

V1 PATH PATTERN	IS SUPPORTED?
/images/*	yes
/images*	no
/images/*.jpg	no
/*.jpg	no
/Repos/*/Comments/*	no
/CurrentUser/Comments/*	yes

v2

Path rules are case insensitive.

V2 PATH PATTERN	IS SUPPORTED?
/images/*	yes
/images*	yes
/images/*.jpg	no
/*.jpg	no
/Repos/*/Comments/*	no
/CurrentUser/Comments/*	yes

You can check out a [Resource Manager template using URL-based routing](#) for more information.

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/<listenerName>"
      },
      "urlPathMap": {
        "id": "/subscriptions/{subscriptionId}.../microsoft.network/applicationGateways/{gatewayName}/urlPathMaps/{urlpathMapName}"
      }
    }
  }
]
```

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

11/15/2019 • 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.

The following types of redirection are supported:

- 301 Permanent Redirect
- 302 Found
- 303 See Other
- 307 Temporary Redirect

Application Gateway redirection support offers the following capabilities:

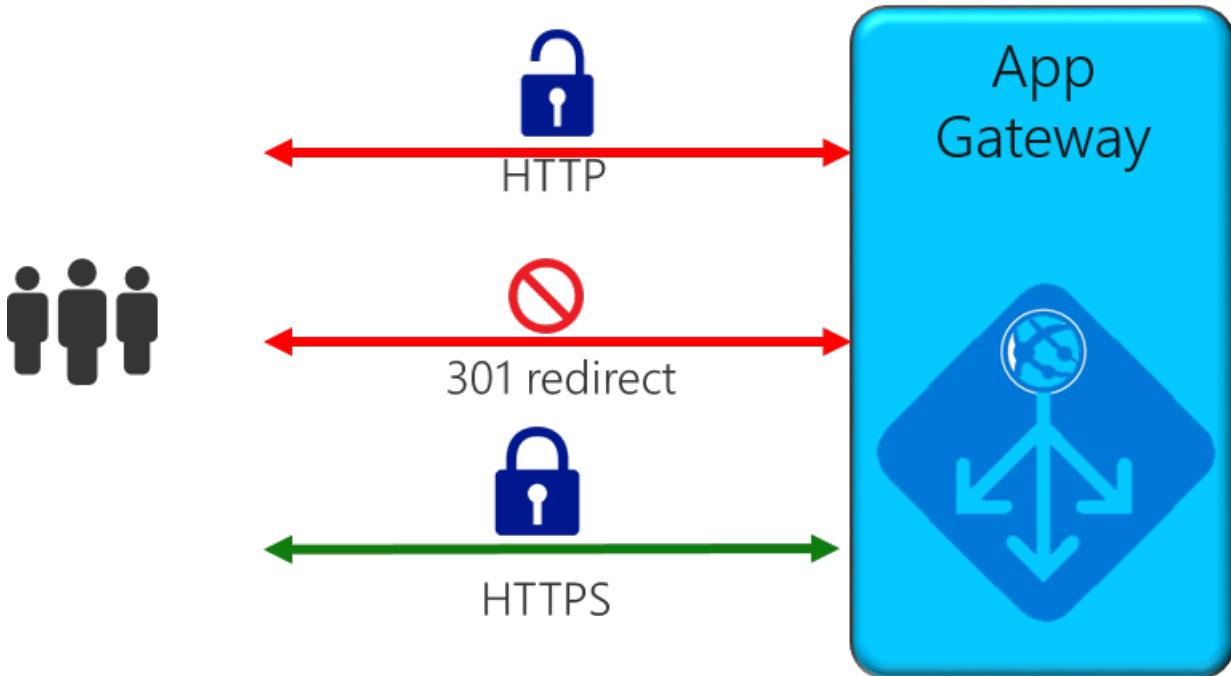
- **Global redirection**

Redirects from one listener to another listener 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 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 with Application Gateway

8/9/2019 • 9 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

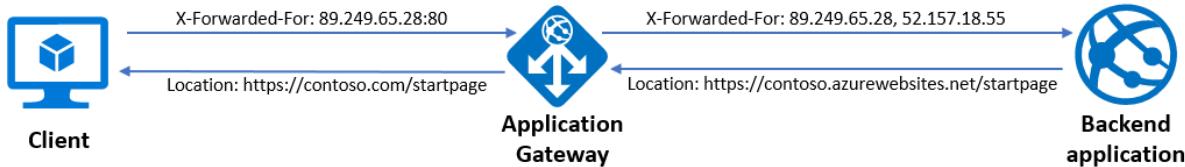
HTTP headers allow a client and server to pass additional information with a request or response. By rewriting these headers, you can accomplish important tasks, such as adding security-related header fields like HSTS/ X-XSS-Protection, removing response header fields that might reveal sensitive information, 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. And it allows you to add conditions to ensure that the specified headers are rewritten only when certain conditions are met.

Application Gateway also supports several [server variables](#) that help you store additional information about requests and responses. This makes it easier for you to create powerful rewrite rules.

NOTE

The HTTP header rewrite support is only available for the [Standard_V2](#) and [WAF_v2](#) SKU.



Supported headers

You can rewrite all headers in requests and responses, except for the Host, 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.

Rewrite conditions

You can use rewrite conditions to evaluate the content of HTTP(S) requests and responses and perform a header rewrite only when one or more conditions are met. The application gateway uses these types of variables to evaluate the content of HTTP(S) 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. You 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](#).

Rewrite actions

You use rewrite actions to specify the request and response headers that you want to rewrite and the new value for the headers. You can either create a new header, modify the value of an existing header, or delete an existing header. The value of a new header or an 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}.
- A combination of text, a request header, a response header, and a server variable.

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.

Application gateway supports these server variables:

VARIABLE NAME	DESCRIPTION
add_x_forwarded_for_proxy	The X-Forwarded-For client request header field with the <code>client_ip</code> 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 <code>add_x_forwarded_for_proxy</code> variable is equal to the <code>\$client_ip</code> 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 SSL 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, <code>client_ip</code> 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.
cookie_name	The <code>name</code> cookie.
http_method	The method used to make the URL request. For example, GET or POST.

VARIABLE NAME	DESCRIPTION
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.
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).
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 SSL connection.
ssl_enabled	"On" if the connection operates in SSL mode. Otherwise, an empty string.

Rewrite configuration

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 want 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 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 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 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 (`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.

NOTE

URL Rewrite alter the headers; it does not change the URL for the path.

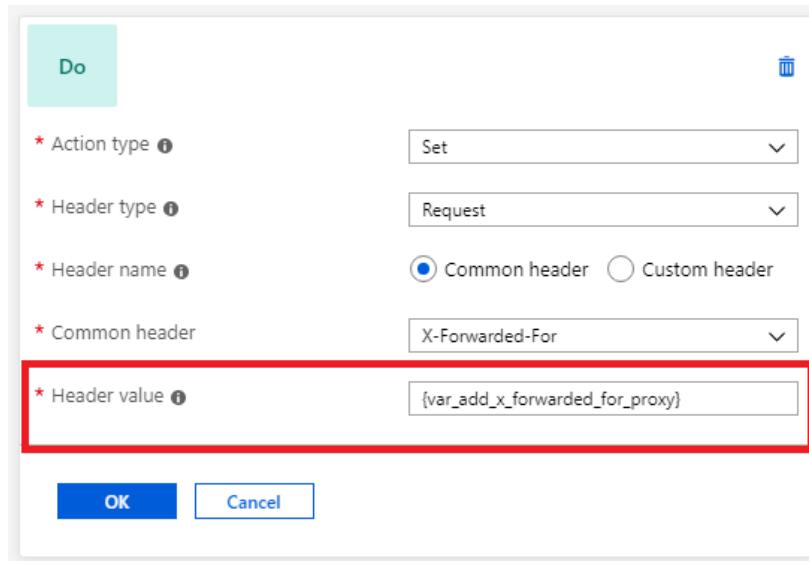
You can create multiple HTTP header rewrite sets and apply each rewrite set to multiple listeners. But you can apply only one rewrite set to a specific listener.

Common scenarios

Here are some common scenarios for using 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:



Modify a redirection URL

When a back-end application sends a redirection response, you might want to redirect the client to a different URL than the one specified by the back-end application. For example, you might want to do this when an app service is hosted behind an application gateway and requires the client to do a redirection to its relative path. (For example, a redirect from contoso.azurewebsites.net/path1 to contoso.azurewebsites.net/path2.)

Because App Service is a multitenant service, it uses the host header in the request to route the request to the correct endpoint. App services have a default domain name of *.azurewebsites.net (say contoso.azurewebsites.net) that's different from the application gateway's domain name (say contoso.com). Because the original request from the client has the application gateway's domain name (contoso.com) as the hostname, the application gateway changes the hostname to contoso.azurewebsites.net. It makes this change so that the app service can route the request to the correct endpoint.

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:

- 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(.*?)$`.
- 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.

If

* Type of variable to check <small>i</small>	HTTP header
* Header type	Response
* Header name	<input checked="" type="radio"/> Common header <input type="radio"/> Custom header
* Common header	Location
* Case-sensitive <small>i</small>	<input type="radio"/> Yes <input checked="" type="radio"/> No
* Operator	equal (=)
* Pattern to match <small>i</small> (https?://)?(www\.)?azurewebsites\.net(.*\$)	

Then

* Action type <small>i</small>	Set
* Header type <small>i</small>	Response
* Header name <small>i</small>	<input checked="" type="radio"/> Common header <input type="radio"/> Custom header
* Common header	Location
* Header value <small>i</small> {http_resp_Location_1}://contoso.com{http_r...}	

Implement security HTTP headers to prevent vulnerabilities

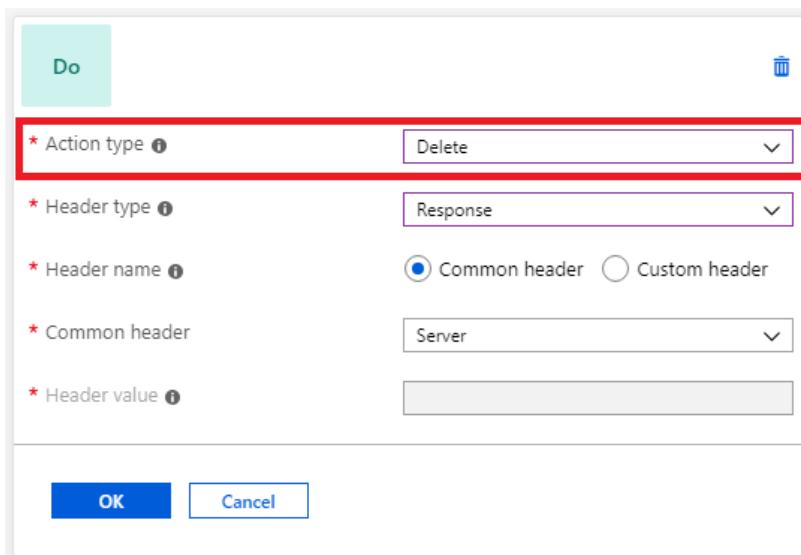
You can fix several security vulnerabilities by implementing necessary headers in the application response. These security headers include X-XSS-Protection, Strict-Transport-Security, and Content-Security-Policy. You can use Application Gateway to set these headers for all responses.

Do

* Action type <small>i</small>	Set
* Header type <small>i</small>	Response
* Header name <small>i</small>	<input checked="" type="radio"/> Common header <input type="radio"/> Custom header
* Common header	Strict-Transport-Security
* Header value <small>i</small> max-age=31536000	

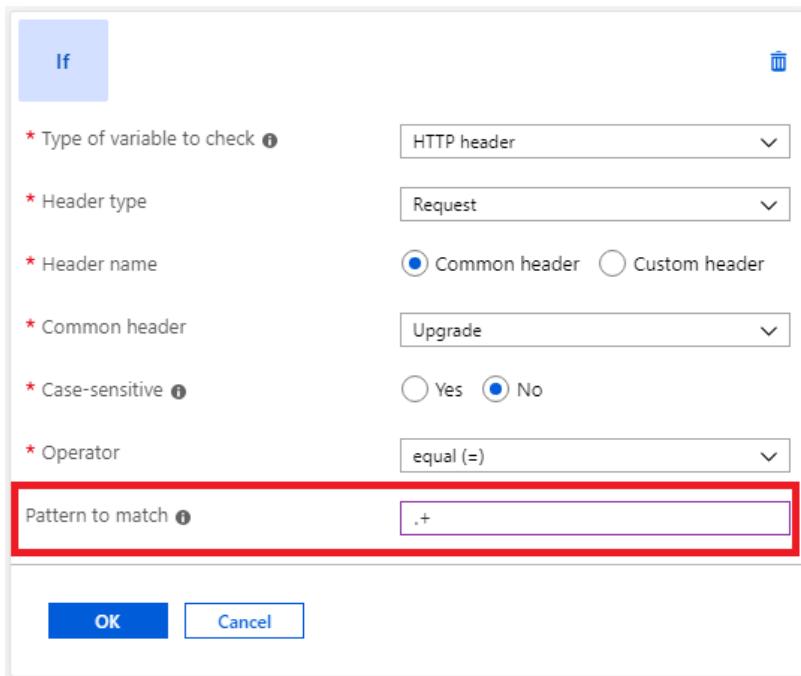
Delete unwanted headers

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.



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.net
```

and another for application gateway affinity, for example,

```
Set-Cookie: ApplicationGatewayAffinity=c1a2bd511fd396387f96b19cc3d2c516; Path=/ . Rewriting one of the Set-Cookie  
headers in this scenario can result in removing the other Set-Cookie header from the response.
```

- Rewriting the Connection, Upgrade, and Host headers isn't currently supported.
- Header names can contain any alphanumeric characters and specific symbols as defined in [RFC 7230](#). We don't currently support the underscore (_) special character in Header names.

Next steps

To learn how to rewrite HTTP headers, see:

- [Rewrite HTTP headers using Azure portal](#)
- [Rewrite HTTP headers using Azure PowerShell](#)

Create Application Gateway custom error pages

11/15/2019 • 2 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.
- the corresponding location for the error page.
- a publicly accessible Azure storage blob for the location.
- an *.htm or *.html extension type.

The size of the error page must be less than 1 MB. If there are images linked in the error page, they must be either publicly accessible absolute URLs or base64 encoded image inline in the custom error page. Relative links with images in the same blob location are currently not supported.

After you specify an error page, the application gateway downloads it from the storage blob location and saves it to the local application gateway cache. Then the error page is served directly from the application gateway. 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.

TestAppGwv2 Application gateway

Search (Ctrl+F)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Settings
- Configuration
- Web application firewall
- Backend pools
- HTTP settings
- Frontend IP configurations
- Listeners
- Rules
- Health probes
- Properties
- Locks
- Automation script
- Monitoring
- Alerts
- Metrics
- Backend health
- Diagnostics logs
- Support + troubleshooting
- New support request

Move Delete

Resource group (change) TestAppGwv2

Location France Central

Subscription (change) Amt-Saturne

Subscription ID

Virtual network/subnet TestAppGwv2/appgwsubnet

Frontend public IP address 40.66.57.0 (TestAppGwv2-ip)

Frontend private IP address -

Tier Standard_2

Availability zone -

Tags (change) Click here to add tags

Show data for last 1 hour 8 hours 12 hours 1 day 7 days 30 days

Sum Total Requests

Time	Total Requests (Sum)
10:15 AM	0.0
10:30 AM	0.9
10:45 AM	1.1
11:00 AM	0.0

Sum Failed Requests

Time	Failed Requests (Sum)
10:15 AM	0.0
10:30 AM	0.3
10:45 AM	0.5
11:00 AM	0.0

Sum Response Status by HttpStatus

Time	200 (Sum)	404 (Sum)
10:15 AM	1.0	0.0
10:30 AM	1.0	0.0
10:45 AM	1.0	0.1
11:00 AM	1.0	0.0

Sum Throughput

Time	Throughput (Sum)
10:15 AM	1.0
10:30 AM	1.0
10:45 AM	1.0
11:00 AM	1.0

Sum CurrentConnections

Time	CurrentConnections (Sum)
10:15 AM	0.0
10:30 AM	0.0
10:45 AM	0.0
11:00 AM	0.0

Avg Healthy Host Count by BackendPool HttpStatus

Time	HealthyHostCount (Avg)
10:15 AM	1.0
10:30 AM	1.0
10:45 AM	1.0
11:00 AM	1.0

2. Click **Listeners** and navigate to a particular listener where you want to specify an error page.

SSL Policy

Configure a centralized SSL policy to match your organizational security requirements. An SSL policy offers control over the SSL protocol version as well as which ciphers are used during SSL handshakes. You can choose from one of the predefined security policies or create a custom security policy based on your security requirements. If you don't specify an SSL policy, the default policy will be used for your gateway.

Default Predefined Custom

Min protocol version: TLS_1_3

Cipher suites:

- 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_SHA
- TLS_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_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA384

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. Specify a publicly accessible blob URL for a given error status code and click **Save**. The Application Gateway is now configured with the custom error page.

The screenshot shows the Azure portal interface for configuring an Application Gateway listener. The top navigation bar includes 'Home', 'Application gateways', 'TestAppGwv2 - Listeners', and 'appGatewayHttpListener'. The main title is 'appGatewayHttpListener' under 'TestAppGwv2'. Below the title are three buttons: 'Save' (blue), 'Discard' (red), and 'Delete' (green). The configuration fields include:

- Name:** appGatewayHttpListener
- Frontend IP configuration:** appGatewayFrontendIP
- Frontend port:** appGatewayFrontendPort (443)
- Protocol:** HTTPS
- Certificate:** scrap
- Associated rule:** rule1
- CUSTOM ERROR PAGES**

ERROR CODE	URL
Forbidden - 403	https://mycustomerrorpages.blob.core.windows.net/403.html
Bad Gateway - 502	https://mycustomerrorpages.blob.core.windows.net/502.html

Azure PowerShell configuration

You can use Azure PowerShell to configure a custom error page. For example, a global custom error page:

```
$updatedgateway = Add-AzApplicationGatewayCustomError -ApplicationGateway $appgw -StatusCode HttpStatus502 -CustomErrorResponseUrl $customError502Url
```

Or a listener level error page:

```
$updatedlistener = Add-AzApplicationGatewayHttpListenerCustomError -HttpListener $listener01 -StatusCode HttpStatus502 -CustomErrorResponseUrl $customError502Url
```

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](#).

Overview of SSL termination and end to end SSL with Application Gateway

1/3/2020 • 8 minutes to read • [Edit Online](#)

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 SSL termination at the gateway as well as end to end SSL encryption.

SSL termination

Application Gateway supports SSL termination at the gateway, after which traffic typically flows unencrypted to the backend servers. There are a number of advantages of doing SSL termination at the application gateway:

- **Improved performance** – The biggest performance hit when doing SSL decryption is the initial handshake. To improve performance, the server doing the decryption caches SSL 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 has to re-authenticate. 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 SSL termination, an SSL certificate is required to be added to the listener to enable the application gateway to derive a symmetric key as per SSL protocol specification. The symmetric key is then used to encrypt and decrypt the traffic sent to the gateway. The SSL 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.

NOTE

Application gateway does not provide any capability to create a new certificate or send a certificate request to a certification authority.

For the SSL connection to work, you need to ensure that the 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 SSL 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 an industry standard certificate guidelines. This will turn the browser locator bar green and publish 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 SSL termination with application gateway](#).

Size of the certificate

Check the [Application Gateway limits](#) section to know the maximum SSL certificate size supported.

End to end SSL encryption

Some customers may not desire unencrypted communication to the backend servers. This could be due to security requirements, compliance requirements, or the application may only accept a secure connection. For such applications, application gateway supports end to end SSL encryption.

End to end SSL allows you to securely transmit sensitive data to the backend encrypted while still taking advantage of the benefits of Layer 7 load-balancing features which application gateway provides. Some of these features are cookie-based session affinity, URL-based routing, support for routing based on sites, or ability to inject X-Forwarded-* headers.

When configured with end to end SSL communication mode, application gateway terminates the SSL 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 SSL 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 SSL is enabled by setting protocol setting in [Backend HTTP Setting](#) to HTTPS, which is then applied to a backend pool.

The SSL 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 SSL handshake.

Application gateway only communicates with those backend instances that have either whitelisted their certificate with the application gateway or whose certificates are signed by well known CA authorities where the certificate 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.

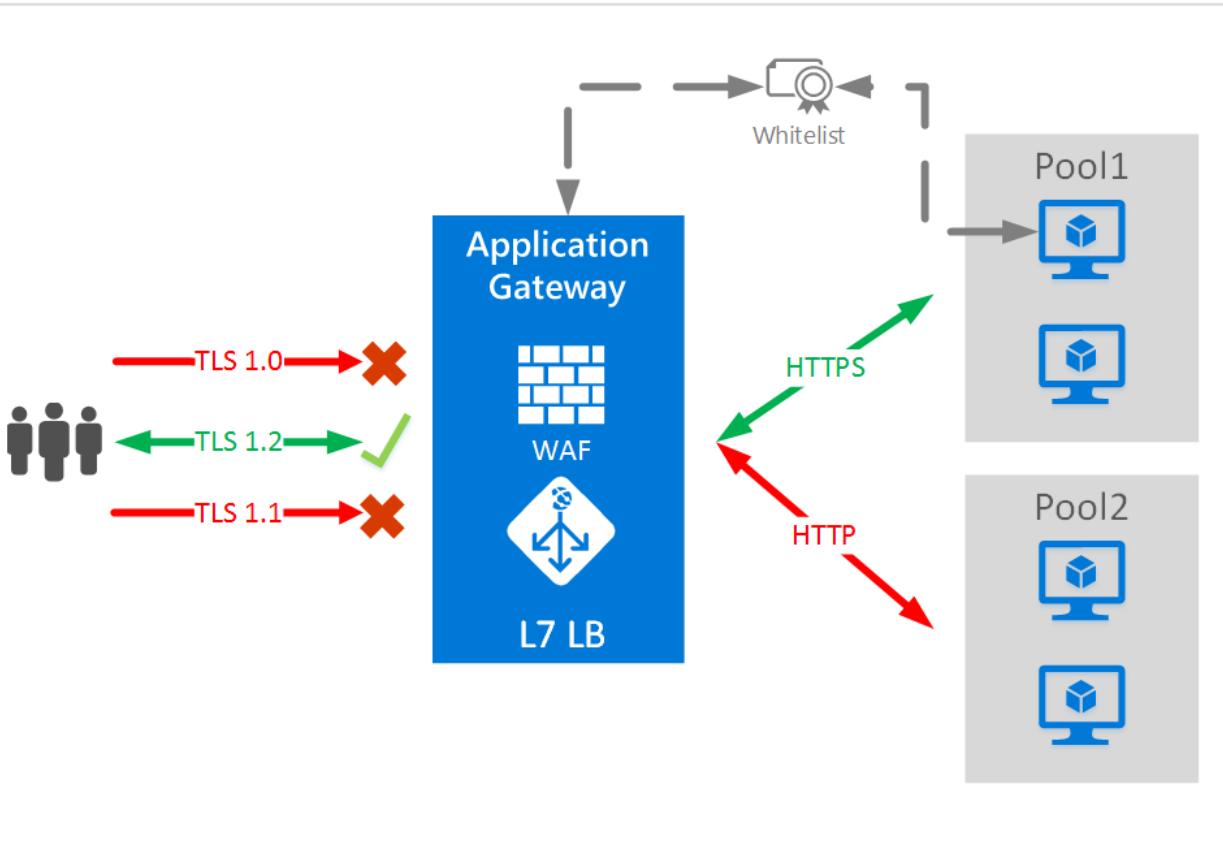
If the certificates of the members in the backend pool are not signed by well known CA authorities, then each instance in the backend pool with end to end SSL enabled must be configured with a certificate to allow secure communication. Adding the certificate ensures that the application gateway only communicates with known back-end instances. This further secures the end-to-end communication.

NOTE

Authentication certificate setup is not required for trusted Azure services such as Azure App service web apps and Azure API Management.

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 SSL termination at application gateway or different for enhanced security.



In this example, requests using TLS 1.2 are routed to backend servers in Pool1 using end to end SSL.

End to end SSL and whitelisting of certificates

Application gateway only communicates with known backend instances that have whitelisted their certificate with the application gateway. To enable whitelisting of certificates, you must upload the public key of backend server certificates to the application gateway (not the root certificate). Only connections to known and whitelisted backends are then allowed. The remaining backends results in a gateway error. Self-signed certificates are for test purposes only and not recommended for production workloads. Such certificates must be whitelisted with the application gateway as described in the preceding steps before they can be used.

NOTE

Authentication certificate setup is not required for trusted Azure services such as Azure App Service.

End to end SSL 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 SSL 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 SSL 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 SSL.

NOTE

In order for an SSL certificate to be trusted, that certificate of the backend server must have been issued by a CA that is included in the trusted store of the Application Gateway. 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 certificate is self-signed, or signed by unknown intermediaries, then to enable end to end SSL in v2 SKU a trusted root certificate must be defined. 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.

NOTE

The self-signed certificate must be a part of a certificate chain. A single self-signed certificate with no chain is not supported in V2 SKU.

- In addition to root certificate match, Application Gateway 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 SSL certificate. When trying to establish an SSL connection to the backend, Application Gateway sets the Server Name Indication (SNI) extension to the Host specified in the backend http setting.
- If **pick hostname from backend address** 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 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.

Next steps

After learning about end to end SSL, go to [Configure end to end SSL by using Application Gateway with PowerShell](#) to create an application gateway using end to end SSL.

Application Gateway SSL policy overview

1/8/2020 • 2 minutes to read • [Edit Online](#)

You can use Azure Application Gateway to centralize SSL certificate management and reduce encryption and decryption overhead from a back-end server farm. This centralized SSL handling also lets you specify a central SSL policy that's suited to your organizational security requirements. This helps you meet compliance requirements as well as security guidelines and recommended practices.

The SSL policy includes control of the SSL protocol version as well as the cipher suites and the order in which ciphers are used during an SSL handshake. Application Gateway offers two mechanisms for controlling SSL policy. You can use either a predefined policy or a custom policy.

Predefined SSL policy

Application Gateway has three 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. Each policy offers different SSL protocol versions and cipher suites. We recommend that you use the newest SSL policies to ensure the best SSL security.

AppGwSslPolicy20150501

PROPERTY	VALUE
Name	AppGwSslPolicy20150501
MinProtocolVersion	TLSv1_0
Default	True (if no predefined policy is specified)

PROPERTY	VALUE
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 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

AppGwSslPolicy20170401

PROPERTY	VALUE
Name	AppGwSslPolicy20170401
MinProtocolVersion	TLSv1_1
Default	False
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 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_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

AppGwSslPolicy20170401S

PROPERTY	VALUE
Name	AppGwSslPolicy20170401S
MinProtocolVersion	TLSv1_2
Default	False
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 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_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_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA

Custom SSL policy

If a predefined SSL policy needs to be configured for your requirements, you must define your own custom SSL policy. With a custom SSL policy, you have complete control over the minimum SSL protocol version to support, as well as the supported cipher suites and their priority order.

SSL protocol versions

- SSL 2.0 and 3.0 are disabled by default for all application gateways. These protocol versions are not configurable.
- A custom SSL policy gives you the option to select any one of the following three protocols as the minimum SSL protocol version for your gateway: TLSv1_0, TLSv1_1, and TLSv1_2.
- If no SSL policy is defined, all three protocols (TLSv1_0, TLSv1_1, and TLSv1_2) are enabled.

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 SSL negotiation.

- 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

NOTE

SSL cipher suites used for the connection are also based on the type of the certificate being used. In client to application gateway connections, the cipher suites used are based on the type of server certificates on the application gateway listener. In application gateway to backend pool connections, the cipher suites used are based on the type of server certificates on the backend pool servers.

Known issue

Application Gateway v2 does not currently support the following ciphers:

- DHE-RSA-AES128-GCM-SHA256
- DHE-RSA-AES128-SHA
- DHE-RSA-AES256-GCM-SHA384
- DHE-RSA-AES256-SHA
- DHE-DSS-AES128-SHA256
- DHE-DSS-AES128-SHA
- DHE-DSS-AES256-SHA256
- DHE-DSS-AES256-SHA

Next steps

If you want to learn to configure an SSL policy, see [Configure SSL policy on an application gateway](#).

SSL termination with Key Vault certificates

2/11/2020 • 2 minutes to read • [Edit Online](#)

Azure Key Vault is a platform-managed secret store that you can use to safeguard secrets, keys, and 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.

Key Vault integration offers two models for SSL termination:

- You can explicitly provide SSL certificates attached to the listener. This model is the traditional way to pass SSL certificates to Application Gateway for SSL termination.
- You can optionally provide a reference to an existing Key Vault certificate or secret when you create an HTTPS-enabled listener.

Application Gateway integration with Key Vault offers many benefits, including:

- Stronger security, because 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.

Application Gateway currently supports software-validated certificates only. Hardware security module (HSM)-validated certificates are not supported. After Application Gateway is configured to use Key Vault certificates, its instances retrieve the certificate from Key Vault and install them locally for SSL termination. The instances also poll Key Vault at 24-hour intervals to retrieve a renewed version of the certificate, if it exists. If an updated certificate is found, the SSL certificate currently associated with the HTTPS listener is automatically rotated.

NOTE

The Azure portal only supports KeyVault Certificates, not secrets. Application Gateway still supports referencing secrets from KeyVault, but only through non-Portal resources like PowerShell, CLI, API, ARM templates, etc.

How integration works

Application Gateway integration with Key Vault requires a three-step configuration process:

1. Create a user-assigned managed identity

You create or reuse an existing user-assigned managed identity, which Application Gateway uses to retrieve certificates from Key Vault on your behalf. For more information, see [What is managed identities for Azure resources?](#). This step creates a new identity in the Azure Active Directory tenant. The identity is trusted by the subscription that's used to create the identity.

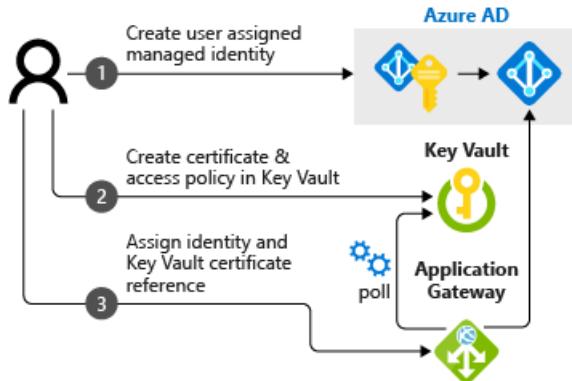
2. Configure your key vault

You then either import an existing certificate or create a new one in your key vault. The certificate will be

used by applications that run through the application gateway. In this step, you can also use a key vault secret that's stored as a password-less, base 64-encoded PFX file. We recommend using a certificate type because of the autorenewal capability that's available with certificate type objects in the key vault. After you've created a certificate or a secret, you define access policies in the key vault to allow the identity to be granted *get* access to the secret.

3. Configure the application gateway

After you complete the two preceding steps, you can set up or modify an existing application gateway to use the user-assigned managed identity. You can also configure the HTTP listener's SSL certificate to point to the complete URI of the Key Vault certificate or secret ID.



Next steps

[Configure SSL termination with Key Vault certificates by using Azure PowerShell](#)

What is Application Gateway Ingress Controller?

12/31/2019 • 2 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.

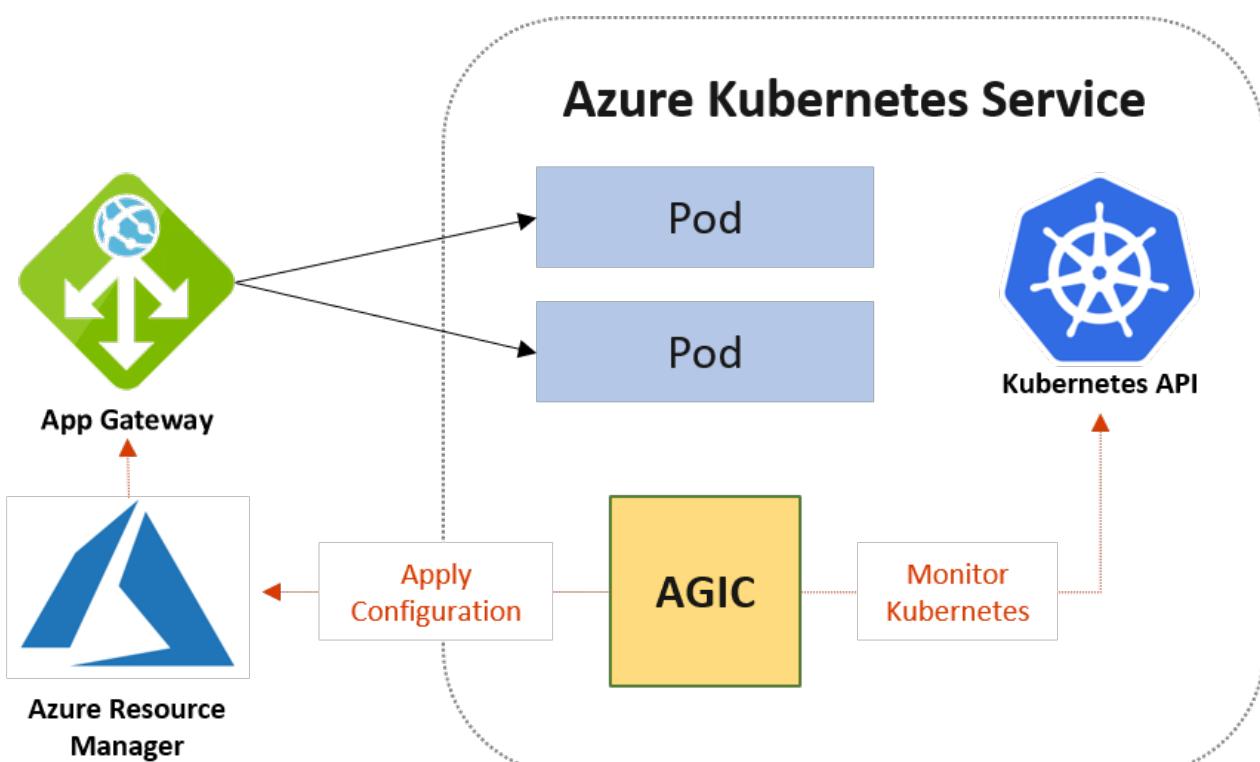
The Ingress Controller runs in its own pod on the customer's AKS. AGIC monitors a subset of Kubernetes Resources for changes. The state of the AKS cluster is translated to Application Gateway specific configuration and applied to the [Azure Resource Manager \(ARM\)](#).

Benefits of Application Gateway Ingress Controller

AGIC allows your deployment to control multiple AKS clusters with a single Application Gateway Ingress Controller. AGIC also helps eliminate the need to have another load balancer/public IP in front of 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
- SSL termination
- End-to-end SSL
- Support for public, private, and hybrid web sites
- Integrated web application firewall

AGIC is able to handle multiple namespaces and has ProhibitedTargets, which means AGIC can configure the Application Gateway specifically for AKS clusters without affecting other existing backends.

Next Steps

- **Greenfield Deployment:** Instructions on installing AGIC, AKS, and Application Gateway on blank-slate infrastructure.
- **Brownfield Deployment:** Install AGIC on an existing AKS and Application Gateway.

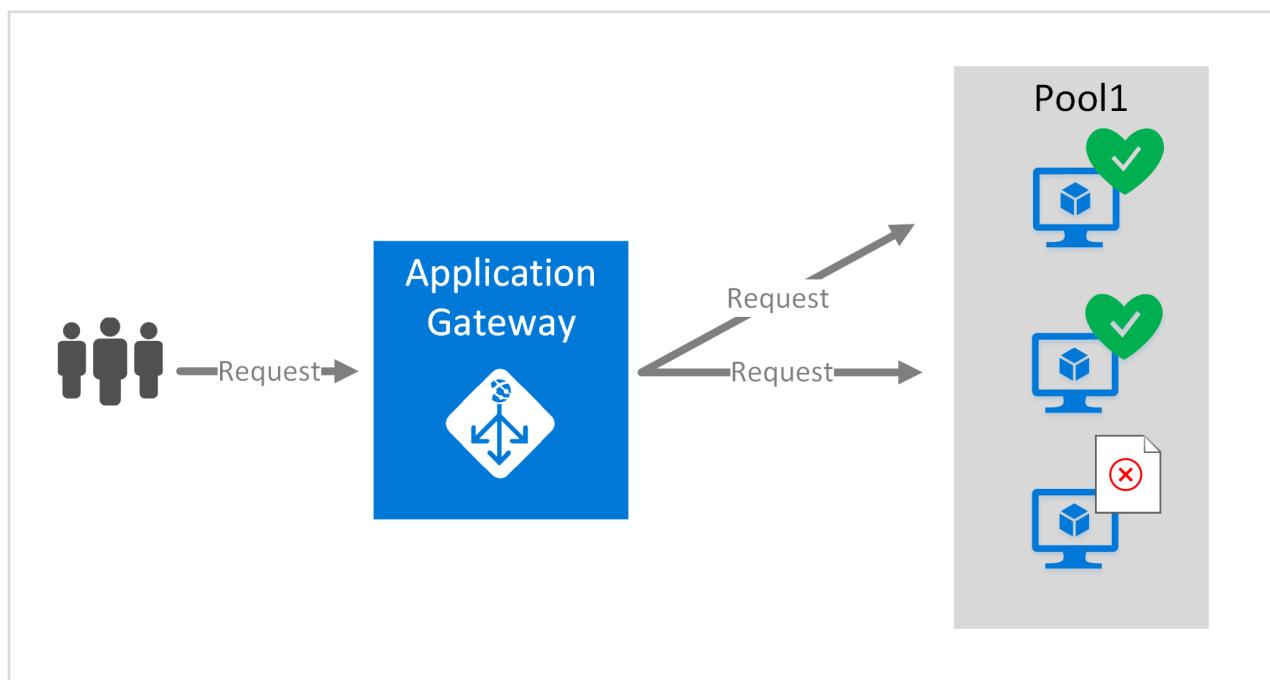
Application Gateway health monitoring overview

2/20/2020 • 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. Application gateway sends the health probes with the same port that is defined in the back-end HTTP settings. This configuration ensures that the probe is testing the same port that customers would be using to connect to the backend.

The source IP address Application Gateway uses for health probes depends on the backend pool:

- If the backend pool is a public endpoint, then the source address is the application gateway frontend public IP address.
- If the backend pool is a private endpoint, then the source IP address is from the application gateway subnet 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 request to the IP addresses configured for the back-end pool. For default probes if the backend http settings are configured for HTTPS, the probe uses HTTPS as

well to test health of the backends.

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. A healthy HTTP response has a [status code](#) between 200 and 399.

If the default probe check fails for server A, the application gateway removes it from its back-end pool, and network traffic stops flowing 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, it's added back as healthy to the back-end pool, and traffic starts flowing to the server again.

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.

Default health probe settings

PROBE PROPERTY	VALUE	DESCRIPTION
Probe URL	<code>http://127.0.0.1:<port>/</code>	URL path
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.

PROBE PROPERTY	VALUE	DESCRIPTION
Unhealthy threshold	3	Governs how many probes to send in case there's a failure of the regular health probe. 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. The back-end server is marked down after the consecutive probe failure count reaches the unhealthy threshold.

NOTE

The port is the same port as the back-end HTTP settings.

The default probe looks only at `http://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.

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 a more granular control over the health monitoring. When using custom probes, you can configure the probe interval, the URL and path to test, and how many failed responses to accept before marking the back-end pool instance as unhealthy.

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 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 Application Gateway, otherwise use '127.0.0.1'. This value is different from VM host name.
Path	Relative path of the probe. The valid path starts from '/'.

PROBE PROPERTY	DESCRIPTION
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.

IMPORTANT

If 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. For reference a custom probe is sent to <protocol>://<host>:<port><path>. The port used will be the same port as defined in the back-end HTTP settings.

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

1/14/2020 • 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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.

SERVER (BACKEND POOL)	PORT (HTTP SETTING)	STATUS
backend-qghdzsq7jvsy-1.westcentralus.cloudapp.azure.com	80 (appGatewayBackendHttpSettings)	Healthy
backend-qghdzsq7jvsy-2.westcentralus.cloudapp.azure.com	80 (appGatewayBackendHttpSettings)	Healthy

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/back-
endAddressPools/appGatewayBackendPool"
  },
  "BackendHttpSettingsCollection": [
    {
      "BackendHttpSettings": {
        "Id": "/00000000-0000-0000-
000000000000/resourceGroups/ContosoRG/providers/Microsoft.Network/applicationGateways/applicationGateway1/back-
endHttpSettingsCollection/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 300 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.

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.

appgatewaylogs2 - Properties

Storage account

Search (Ctrl+ /)

PRIMARY STATUS
Available

SECONDARY STATUS
Available

LAST FAILOVER
Never

PROVISIONING STATE
Succeeded

KIND
Storage

SKU
Standard_RAGRS

CREATED
6/9/2017, 10:06:15 AM

RESOURCE ID
/subscriptions/
/resourceGroups/ad... 

PRIMARY LOCATION
East US

OVERVIEW

ACTIVITY LOG

ACCESS CONTROL (IAM)

TAGS

DIAGNOSE AND SOLVE PROBLEMS

SETTINGS

ACCESS KEYS

CONFIGURATION

SHARED ACCESS SIGNATURE

PROPERTIES

LOCKS

AUTOMATION SCRIPT

BLOB SERVICE

CONTAINERS

CORS

CUSTOM DOMAIN

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.

The screenshot shows the Azure portal's properties page for an Application Gateway named "AdatumAppGateway". The left sidebar lists various configuration options under "SETTINGS" and "MONITORING". The "Properties" option is highlighted. The right panel displays key resource details:

- PROVISIONING STATE**: Succeeded
- SKU SIZE**: Medium
- INSTANCE COUNT**: 2
- RESOURCE ID**: /subscriptions/... (with a copy icon)
- LOCATION**: East US
- SUBSCRIPTION NAME**: Microsoft Azure (with a copy icon)
- SUBSCRIPTION ID**: (redacted)

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**.

The screenshot shows the 'Diagnostic settings' page for an Application Gateway named 'myAppGateway'. The left sidebar lists various configuration options. The 'Diagnostic settings' option is selected and highlighted with a blue background. The main pane displays the 'Turn on diagnostics' section, which contains a list of log types: ApplicationGatewayAccessLog, ApplicationGatewayPerformanceLog, ApplicationGatewayFirewallLog, and AllMetrics. The 'Subscription' dropdown is set to 'Microsoft Azure Internal Consumption', and the 'Resource group' dropdown is set to 'myResourceGroupAG'. The 'Resource type' dropdown shows '0 selected' and the 'Type to start filtering ...' dropdown.

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.

The screenshot shows the 'Diagnostics settings' page. At the top, there are 'Save', 'Discard', and 'Delete' buttons. The 'Name' field is filled with 'myAGDiagnostics'. Below it, there's a checkbox for 'Archive to a storage account' and another for 'Stream to an event hub', both of which are unchecked. A checked checkbox for 'Send to Log Analytics' is present. The 'Subscription' dropdown is set to 'Microsoft Azure Internal Consumption'. The 'Log Analytics Workspace' dropdown is set to 'DefaultWorkspace-'. The 'LOG' section contains three checked checkboxes: 'ApplicationGatewayAccessLog', 'ApplicationGatewayPerformanceLog', and 'ApplicationGatewayFirewallLog'. The 'METRIC' section contains one checked checkbox: 'AllMetrics'.

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 in the following example for 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 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, the logs show a little more information:

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.
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 seconds) 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.

VALUE	DESCRIPTION
sslEnabled	Whether communication to the back-end pools used SSL. Valid values are on and off.
sslCipher	Cipher suite being used for SSL communication (if SSL is enabled).
sslProtocol	SSL/TLS protocol being used (if SSL 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 from the backend server.
host	Address listed in the host header of the request.

```
{
  "resourceId": "/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/PEERINGTEST/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONGATEWAYS/{applicationGatewayName}",
  "operationName": "ApplicationGatewayAccess",
  "time": "2017-04-26T19:27:38Z",
  "category": "ApplicationGatewayAccessLog",
  "properties": {
    "instanceId": "appgw_1",
    "clientIP": "191.96.249.97",
    "clientPort": 46886,
    "httpMethod": "GET",
    "requestUri": "/phpmyadmin/scripts/setup.php",
    "userAgent": "-",
    "httpStatus": 404,
    "httpVersion": "HTTP/1.0",
    "receivedBytes": 65,
    "sentBytes": 553,
    "timeTaken": 205,
    "sslEnabled": "off",
    "sslCipher": "",
    "sslProtocol": "",
    "serverRouted": "104.41.114.59:80",
    "serverStatus": "200",
    "serverResponseLatency": "0.023",
    "host": "www.contoso.com",
  }
}
```

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.
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/APPLICATIONGATEWAYS/{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
-------	-------------

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.
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 Matched and Blocked.
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.

```
{
  "resourceId": "/SUBSCRIPTIONS/{subscriptionId}/RESOURCEGROUPS/{resourceGroupName}/PROVIDERS/MICROSOFT.NETWORK/APPLICATIONGATEWAYS/{applicationGatewayName}",
  "operationName": "ApplicationGatewayFirewall",
  "time": "2017-03-20T15:52:09.1494499Z",
  "category": "ApplicationGatewayFirewallLog",
  "properties": {
    "instanceId": "ApplicationGatewayRole_IN_0",
    "clientIp": "104.210.252.3",
    "clientPort": "4835",
    "requestUri": "/?a=%3Cscript%3Ealert(%22Hello%22);%3C/script%3E",
    "ruleSetType": "OWASP",
    "ruleSetVersion": "3.0",
    "ruleId": "941320",
    "message": "Possible XSS Attack Detected - HTML Tag Handler",
    "action": "Blocked",
    "site": "Global",
    "details": {
      "message": "Warning. Pattern match \  
<(a|abbr|acronym|address|applet|area|audioscope|b|base|basefront|bdo|bgsound|big|blackface|blink|blockquote|body|bq|br|button|caption|center|cite|code|col|colgroup|comment|dd|del|dfn|dir|div|dl|dt|em|embed|fieldset|fn|font|form|frame|frameset|h1|head|h ...\" at ARGS:a.",
      "data": "Matched Data: <script> found within ARGS:a: <script>alert(\\"x22hello\\x22);</script>",
      "file": "rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf",
      "line": "865"
    }
  },
  "hostname": "40.90.218.100",
  "transactionId": "AYAcUqAcAcAcAcAcASAcAcAc"
}
}
```

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

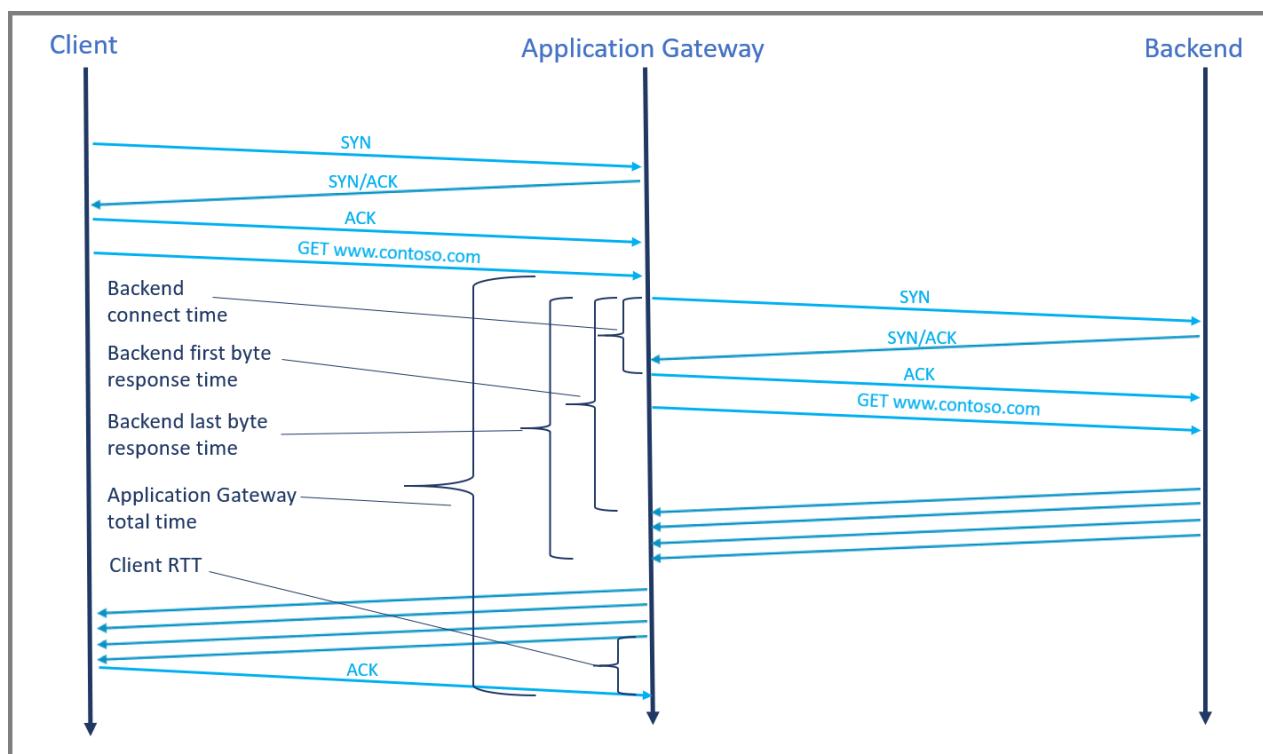
2/19/2020 • 9 minutes to read • [Edit Online](#)

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 SSL, 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*, time taken by Application Gateway to send all the response and the *Client RTT*.

- **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**

Count of failed requests that Application Gateway has served. 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.

- **Web Application Firewall matched rules**

- **Web Application Firewall triggered rules**

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.

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**

Count of failed requests that Application Gateway has served. The request count can be further filtered to show count per each/specific backend pool-http setting combination.

- **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.

- **Web Application Firewall matched rules**

- **Web Application Firewall triggered rules**

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.

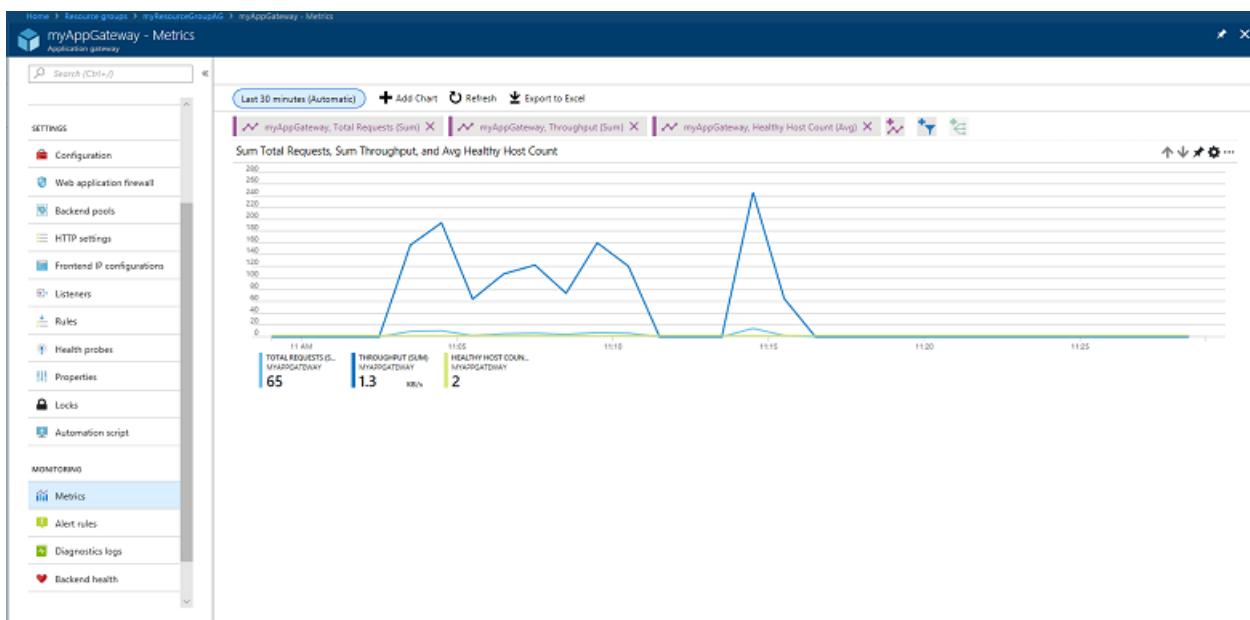
- **Unhealthy host count**

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.

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.

The screenshot shows the 'Alert rules' page for an Application gateway named 'AdatumAppGateway'. The left sidebar has a tree view with 'Properties', 'Locks', 'Automation script', 'MONITORING' (Metrics, Alert rules, Diagnostics logs, Backend health), and 'Logs'. The 'Alert rules' item is selected and highlighted with a blue background. The main area has a search bar, filter buttons for 'Add metric alert' and 'Add activity log alert', and dropdowns for 'Subscription' (Microsoft Azure Internal Consumption), 'Source' (All sources), 'Resource group' (AdatumApp...), 'Resource type' (Application gateways), and 'Resource' (AdatumApp...). Below these is a 'Turn on diagnostics' section with a 'Filter alerts...' search bar. A table titled 'NAME' with columns for STATUS, CONDITION, RESOURCE GROUP, RESOURCE, and LAST FIRED shows 'No results to display'. At the bottom right of the table is a small 'X' icon.

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.

Add rule

Name Throughput over 10

Description

Source

Alert on Metrics

Criteria

Subscription Microsoft

Resource group AdatumAppGatewayRG

Resource AdatumAppGateway

Metric Throughput

Condition Greater than

Threshold 10 bytes/second

Period Over the last 5 minutes

Notify via

Email owners, contributors, and readers

Additional administrator email(s) admin@adatum.com

Webhook HTTP or HTTPS endpoint to route alerts to

OK

If the threshold is breached, an email that's similar to the one in the following image arrives:



Azure

Dear Customer,

⚠ 'Throughput GreaterThan 10 ([context.metricUnit]) in the last 5 minutes' was activated for adatumappgateway

You can view more details for this alert in the [Microsoft Azure Management Portal](#).

RULE NAME: Throughput over 10

RULE DESCRIPTION:

SERVICE: applicationGateways: AdatumAppGateway (AdatumAppGatewayRG)

METRIC: Average Throughput

ALERT ACTIVATED TIME (UTC): 9/26/2016 6:01:00 PM

A list of alerts appears after you create a metric alert. It provides an overview of all the alert rules.

NAME	STATUS	CONDITION	RESOURCE	LAST FIRED
AlertRule1	Active	Throughput > 1...	AdatumAppGat...	AdatumAppGat... Never

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

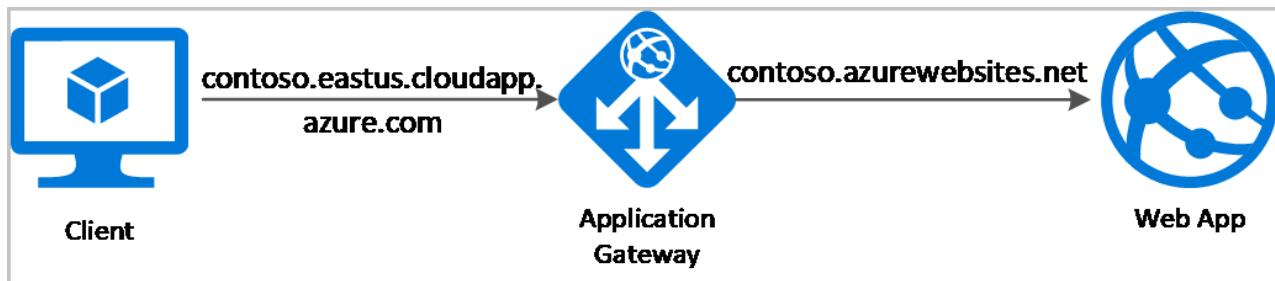
- Visualize counter and event logs by using [Azure Monitor logs](#).
- Visualize your Azure activity log with [Power BI](#) blog post.
- View and analyze Azure activity logs in [Power BI](#) and [more](#) blog post.

Application Gateway support for multi-tenant back ends such as App service

11/13/2019 • 4 minutes to read • [Edit Online](#)

In multi-tenant architectural designs in web servers, multiple websites are running on the same web server instance. Hostnames are used to differentiate between the different applications which are hosted. By default, application gateway does not change the incoming HTTP host header from the client and sends the header unaltered to the back end. This works well for backend pool members such as NICs, virtual machine scale sets, public IP addresses, internal IP addresses and FQDN as these do not rely on a specific host header or SNI extension to resolve to the correct endpoint. However, there are many services such as Azure App service web apps and Azure API management that are multi-tenant in nature and rely on a specific host header or SNI extension to resolve to the correct endpoint. Usually, the DNS name of the application, which in turn is the DNS name associated with the application gateway, is different from the domain name of the backend service. Therefore, the host header in the original request received by the application gateway is not the same as the host name of the backend service. Because of this, unless the host header in the request from the application gateway to the backend is changed to the host name of the backend service, the multi-tenant backends are not able to resolve the request to the correct endpoint.

Application gateway provides a capability which allows users to override the HTTP host header in the request based on the host name of the back-end. This capability enables support for multi-tenant back ends such as Azure App service web apps and API management. This capability is available for both the v1 and v2 standard and WAF SKUs.



NOTE

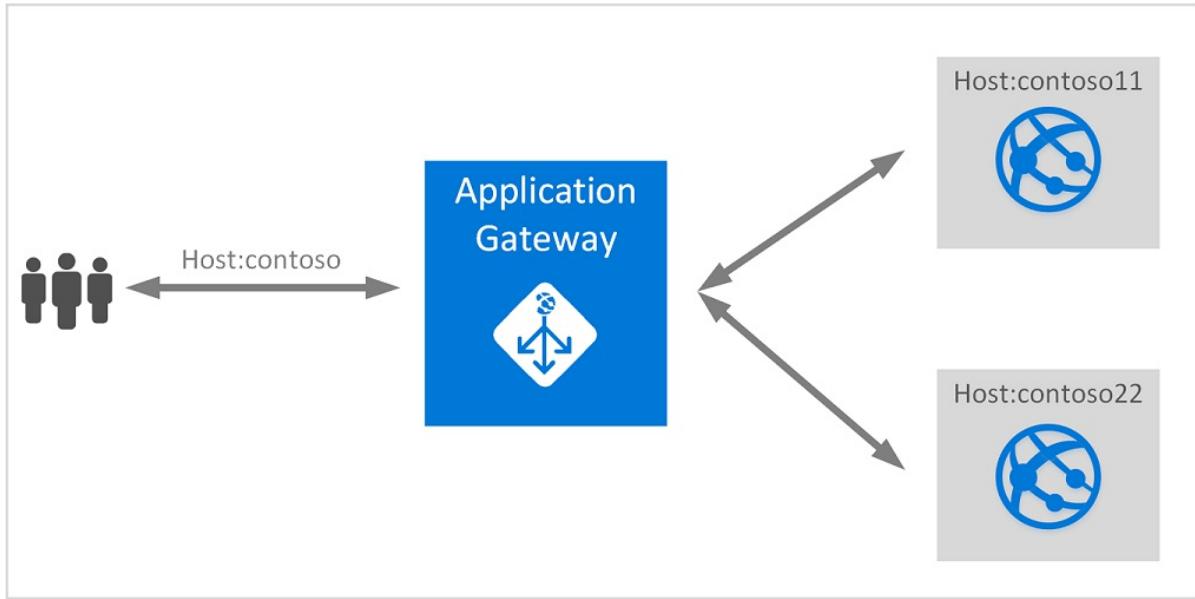
This is not applicable to Azure App service environment (ASE) since ASE is a dedicated resource unlike Azure App service which is a multi-tenant resource.

Override host header in the request

The ability to specify a host override is defined in the [HTTP settings](#) and can be applied to any back-end pool during rule creation. The following two ways of overriding host header and SNI extension for multi-tenant back ends is supported:

- The ability to set the host name to a fixed value explicitly entered in the HTTP settings. This capability ensures that the host header is overridden to this value for all traffic to the back-end pool where the particular HTTP settings are applied. When using end to end SSL, this overridden host name is used in the SNI extension. This capability enables scenarios where a back-end pool farm expects a host header that is different from the incoming customer host header.
- The ability to derive the host name from the IP or FQDN of the back-end pool members. HTTP settings also

provide an option to dynamically pick the host name from a back-end pool member's FQDN if configured with the option to derive host name from an individual back-end pool member. When using end to end SSL, this host name is derived from the FQDN and is used in the SNI extension. This capability enables scenarios where a back-end pool can have two or more multi-tenant PaaS services like Azure web apps and the request's host header to each member contains the host name derived from its FQDN. For implementing this scenario, we use a switch in the HTTP Settings called [Pick hostname from backend address](#) which will dynamically override the host header in the original request to the one mentioned in the backend pool. For example, if your backend pool FQDN contains "contoso11.azurewebsites.net" and "contoso22.azurewebsites.net", the original request's host header which is contoso.com will be overridden to contoso11.azurewebsites.net or contoso22.azurewebsites.net when the request is sent to the appropriate backend server.

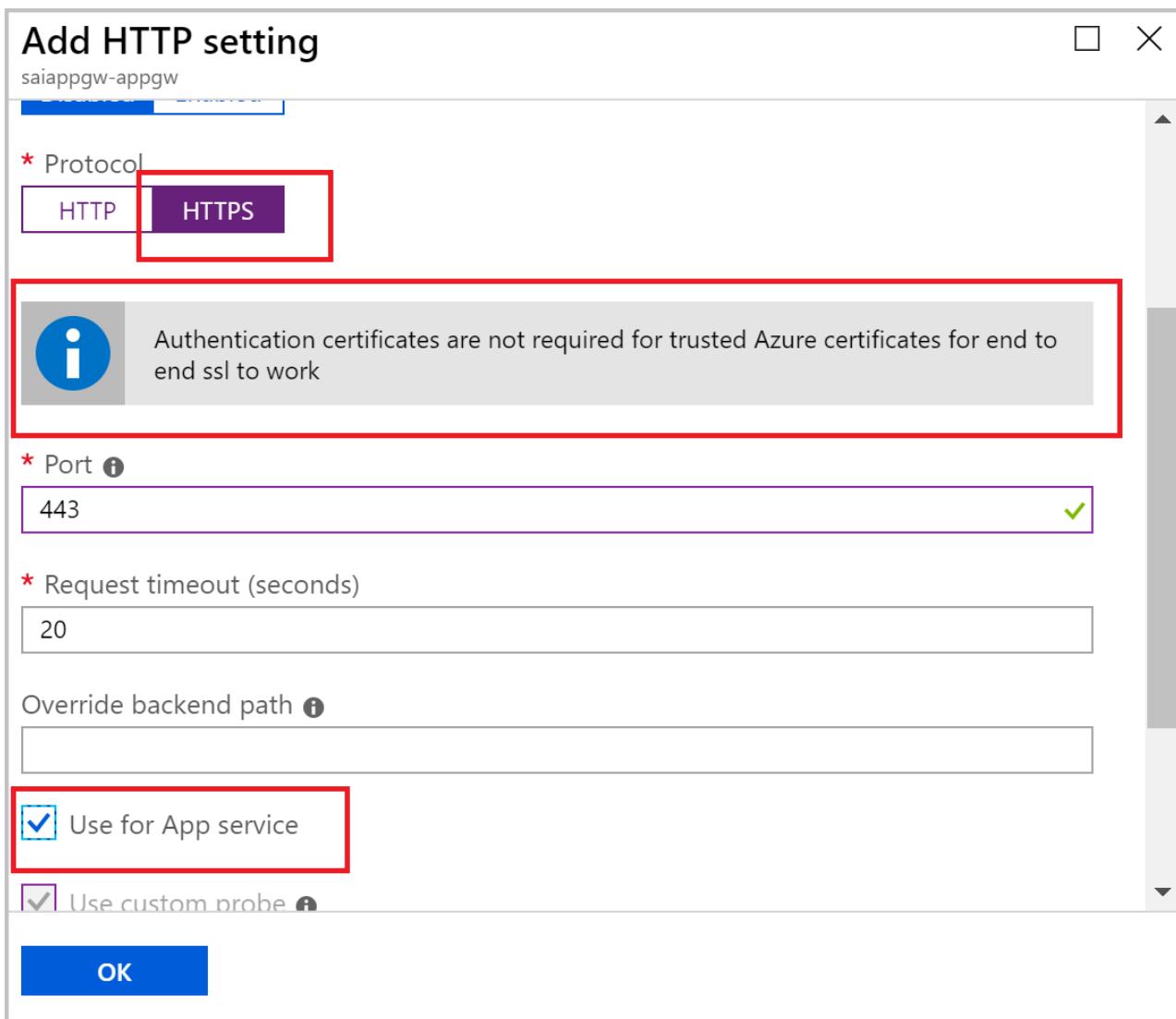


With this capability, customers specify the options in the HTTP settings and custom probes to the appropriate configuration. This setting is then tied to a listener and a back-end pool by using a rule.

Special considerations

SSL termination and end to end SSL with multi-tenant services

Both SSL termination and end to end SSL encryption is supported with multi-tenant services. For SSL termination at the application gateway, SSL certificate continues to be required to be added to the application gateway listener. However, in case of end to end SSL, trusted Azure services such as Azure App service web apps do not require whitelisting the backends in the application gateway. Therefore, there is no need to add any authentication certificates.



Notice that in the above image, there is no requirement to add authentication certificates when App service is selected as backend.

Health probe

Overriding the host header in the **HTTP settings** only affects the request and its routing. It does not impact the health probe behavior. For end to end functionality to work, both the probe and the HTTP settings must be modified to reflect the correct configuration. In addition to providing the ability to specify a host header in the probe configuration, custom probes also support the ability to derive the host header from the currently configured HTTP settings. This configuration can be specified by using the `PickHostNameFromBackendHttpSettings` parameter in the probe configuration.

Redirection to App Service's URL scenario

There can be scenarios where the hostname in the response from the App service may direct the end-user browser to the *.azurewebsites.net hostname instead of the domain associated with the Application Gateway. This issue may happen when:

- You have redirection configured on your App Service. Redirection can be as simple as adding a trailing slash to the request.
- You have Azure AD authentication which causes the redirection.

To resolve such cases, see [Troubleshoot redirection to App service's URL issue](#).

Next steps

Learn how to set up an application gateway with a multi-tenant app such as Azure App service web app as a back-

end pool member by visiting [Configure App Service web apps with Application Gateway](#)

Overview of WebSocket support in Application Gateway

11/15/2019 • 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.

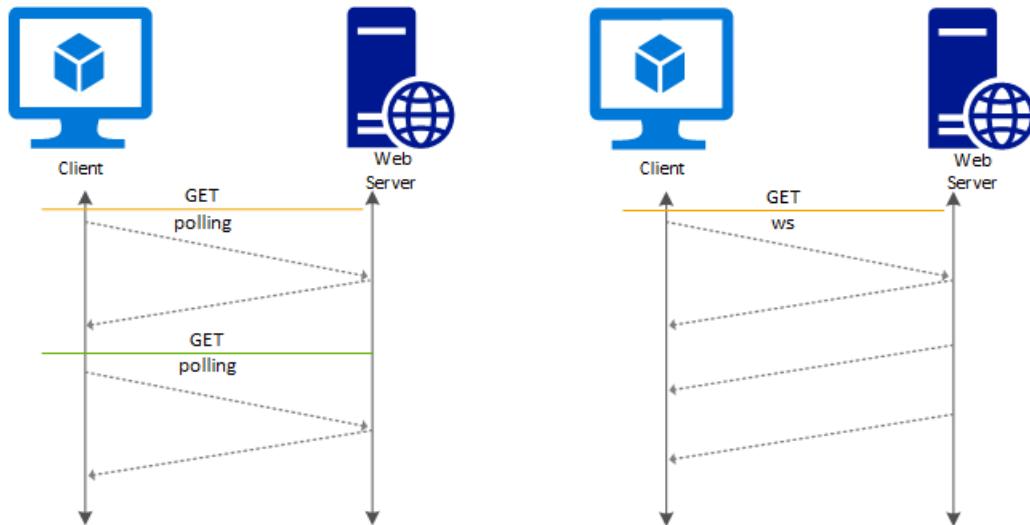
WebSocket protocol standardized in [RFC6455](#) enables a full duplex communication between a server and a client over a long running TCP connection. This feature allows for a more interactive communication between the web server and the client, which can be bidirectional without the need for polling 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.



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/applicationGateways/{applicationGatewayName}/frontendIPConfigurations/DefaultFrontendPublicIP"
            },
            "FrontendPort": {
                "Id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/frontendPorts/appGatewayFrontendPort443"
            },
            "Protocol": "Https",
            "SslCertificate": {
                "Id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/sslCertificates/appGatewaySslCert1"
            },
        }
    },
    {
        "name": "appGatewayHttpListener",
        "properties": {
            "FrontendIPConfiguration": {
                "Id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/frontendIPConfigurations/appGatewayFrontendIP"
            },
            "FrontendPort": {
                "Id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{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/applicationGateways/{applicationGatewayName}/httpListeners/appGatewayHttpsListener')]"
            },
            "backendAddressPool": {
                "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/backendAddressPools/ContosoServerPool')"
            },
            "backendHttpSettings": {
                "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/backendHttpSettingsCollection/appGatewayBackendHttpSettings')]"
            }
        }
    },
    {
        "name": "<ruleName2>",
        "properties": {
            "RuleType": "Basic",
            "httpListener": {
                "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/httpListeners/appGatewayHttpListener')]"
            },
            "backendAddressPool": {
                "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{applicationGatewayName}/backendAddressPools/ContosoServerPool')"
            },
            "backendHttpSettings": {
                "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Network/applicationGateways/{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: dGhlIHhbXbsZSub25jZQ==
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.

Frequently asked questions about Application Gateway

2/27/2020 • 14 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

The following are common questions asked about Azure Application Gateway.

General

What is Application Gateway?

Azure Application Gateway provides an application delivery controller (ADC) as a service. It offers various layer 7 load-balancing capabilities for your applications. This service is highly available, scalable, and fully managed by Azure.

What features does Application Gateway support?

Application Gateway supports autoscaling, SSL offloading, and end-to-end SSL, a web application firewall (WAF), cookie-based session affinity, URL path-based routing, multisite hosting, and other features. For a full list of supported features, see [Introduction to Application Gateway](#).

How do Application Gateway and Azure Load Balancer differ?

Application Gateway is a layer 7 load balancer, which means it works only with web traffic (HTTP, HTTPS, WebSocket, and HTTP/2). It supports capabilities such as SSL termination, cookie-based session affinity, and round robin for load-balancing traffic. Load Balancer load-balances traffic at layer 4 (TCP or UDP).

What protocols does Application Gateway support?

Application Gateway supports HTTP, HTTPS, HTTP/2, and WebSocket.

How does Application Gateway support HTTP/2?

See [HTTP/2 support](#).

What resources are supported as part of a backend pool?

See [supported backend resources](#).

In what regions is Application Gateway available?

Application Gateway is available in all regions of global Azure. It's also available in [Azure China 21Vianet](#) and [Azure Government](#).

Is this deployment dedicated for my subscription, or is it shared across customers?

Application Gateway is a dedicated deployment in your virtual network.

Does Application Gateway support HTTP-to-HTTPS redirection?

Redirection is supported. See [Application Gateway redirect overview](#).

In what order are listeners processed?

See the [order of listener processing](#).

Where do I find the Application Gateway IP and DNS?

If you're using a public IP address as an endpoint, you'll find the IP and DNS information on the public IP address resource. Or find it in the portal, on the overview page for the application gateway. If you're using internal IP addresses, find the information on the overview page.

What are the settings for Keep-Alive timeout and TCP idle timeout?

Keep-Alive timeout governs how long the Application Gateway will wait for a client to send another HTTP request on a persistent connection before reusing it or closing it. *TCP idle timeout* governs how long a TCP connection is kept open in case of no activity.

The *Keep-Alive timeout* in the Application Gateway v1 SKU is 120 seconds and in the v2 SKU it's 75 seconds. The *TCP idle timeout* is a 4-minute default on the frontend virtual IP (VIP) of both v1 and v2 SKU of Application Gateway.

Does the IP or DNS name change over the lifetime of the application gateway?

In Application Gateway V1 SKU, the VIP can change if you stop and start the application gateway. But the DNS name associated with the application gateway doesn't change over the lifetime of the gateway. Because the DNS name doesn't change, you should use a CNAME alias and point it to the DNS address of the application gateway. In Application Gateway V2 SKU, you can set the IP address as static, so IP and DNS name will not change over the lifetime of the application gateway.

Does Application Gateway support static IP?

Yes, the Application Gateway v2 SKU supports static public IP addresses. The v1 SKU supports static internal IPs.

Does Application Gateway support multiple public IPs on the gateway?

An application gateway supports only one public IP address.

How large should I make my subnet for Application Gateway?

See [Application Gateway subnet size considerations](#).

Can I deploy more than one Application Gateway resource to a single subnet?

Yes. In addition to multiple instances of a given Application Gateway deployment, you can provision another unique Application Gateway resource to an existing subnet that contains a different Application Gateway resource.

A single subnet can't support both Standard_v2 and Standard Application Gateway together.

Does Application Gateway support x-forwarded-for headers?

Yes. See [Modifications to a request](#).

How long does it take to deploy an application gateway? Will my application gateway work while it's being updated?

New Application Gateway v1 SKU deployments can take up to 20 minutes to provision. Changes to instance size or count aren't disruptive, and the gateway remains active during this time.

Most deployments that use the v2 SKU take around 6 minutes to provision. However it can take longer depending on the type of deployment. For example, deployments across multiple Availability Zones with many instances can take more than 6 minutes.

Can I use Exchange Server as a backend with Application Gateway?

No. Application Gateway doesn't support email protocols such as SMTP, IMAP, and POP3.

Performance

How does Application Gateway support high availability and scalability?

The Application Gateway v1 SKU supports high-availability scenarios when you've deployed two or more instances. Azure distributes these instances across update and fault domains to ensure that instances don't all fail at the same time. The v1 SKU supports scalability by adding multiple instances of the same gateway to share the load.

The v2 SKU automatically ensures that new instances are spread across fault domains and update domains. If you choose zone redundancy, the newest instances are also spread across availability zones to offer zonal failure resiliency.

How do I achieve a DR scenario across datacenters by using Application Gateway?

Use Traffic Manager to distribute traffic across multiple application gateways in different datacenters.

Does Application Gateway support autoscaling?

Yes, the Application Gateway v2 SKU supports autoscaling. For more information, see [Autoscaling and Zone-redundant Application Gateway](#).

Does manual or automatic scale up or scale down cause downtime?

No. Instances are distributed across upgrade domains and fault domains.

Does Application Gateway support connection draining?

Yes. You can set up connection draining to change members within a backend pool without disruption. For more information, see [connection draining section of Application Gateway](#).

Can I change instance size from medium to large without disruption?

Yes.

Configuration

Is Application Gateway always deployed in a virtual network?

Yes. Application Gateway is always deployed in a virtual network subnet. This subnet can contain only application gateways. For more information, see [virtual network and subnet requirements](#).

Can Application Gateway communicate with instances outside of its virtual network or outside of its subscription?

As long as you have IP connectivity, Application Gateway can communicate with instances outside of the virtual network that it's in. Application Gateway can also communicate with instances outside of the subscription it's in. If you plan to use internal IPs as backend pool members, use [virtual network peering](#) or [Azure VPN Gateway](#).

Can I deploy anything else in the application gateway subnet?

No. But you can deploy other application gateways in the subnet.

Are network security groups supported on the application gateway subnet?

See [Network security groups in the Application Gateway subnet](#).

Does the application gateway subnet support user-defined routes?

See [User-defined routes supported in the Application Gateway subnet](#).

What are the limits on Application Gateway? Can I increase these limits?

See [Application Gateway limits](#).

Can I simultaneously use Application Gateway for both external and internal traffic?

Yes. Application Gateway supports one internal IP and one external IP per application gateway.

Does Application Gateway support virtual network peering?

Yes. Virtual network peering helps load-balance traffic in other virtual networks.

Can I talk to on-premises servers when they're connected by ExpressRoute or VPN tunnels?

Yes, as long as traffic is allowed.

Can one backend pool serve many applications on different ports?

Microservice architecture is supported. To probe on different ports, you need to configure multiple HTTP settings.

Do custom probes support wildcards or regex on response data?

No.

How are routing rules processed in Application Gateway?

See [Order of processing rules](#).

For custom probes, what does the Host field signify?

The Host field specifies the name to send the probe to when you've configured multisite on Application Gateway.

Otherwise use '127.0.0.1'. This value is different from the virtual machine host name. Its format is

<protocol>://<host>:<port><path>.

Can I allow Application Gateway access to only a few source IP addresses?

Yes. See [restrict access to specific source IPs](#).

Can I use the same port for both public-facing and private-facing listeners?

No.

Is there guidance available to migrate from the v1 SKU to the v2 SKU?

Yes. For details see, [Migrate Azure Application Gateway and Web Application Firewall from v1 to v2](#).

Does Application Gateway support IPv6?

Application Gateway v2 does not currently support IPv6. It can operate in a dual stack VNet using only IPv4, but the gateway subnet must be IPv4-only. Application Gateway v1 does not support dual stack VNets.

Configuration - SSL

What certificates does Application Gateway support?

Application Gateway supports self-signed certificates, certificate authority (CA) certificates, Extended Validation (EV) certificates, and wildcard certificates.

What cipher suites does Application Gateway support?

Application Gateway supports the following cipher suites.

- 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

For information on how to customize SSL options, see [Configure SSL policy versions and cipher suites on Application Gateway](#).

Does Application Gateway support reencryption of traffic to the backend?

Yes. Application Gateway supports SSL offload and end-to-end SSL, which reencrypt traffic to the backend.

Can I configure SSL policy to control SSL protocol versions?

Yes. You can configure Application Gateway to deny TLS1.0, TLS1.1, and TLS1.2. By default, SSL 2.0 and 3.0 are already disabled and aren't configurable.

Can I configure cipher suites and policy order?

Yes. In Application Gateway, you can [configure cipher suites](#). To define a custom policy, enable at least one of the following cipher suites.

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256

Application Gateway uses SHA256 to for backend management.

How many SSL certificates does Application Gateway support?

Application Gateway supports up to 100 SSL certificates.

How many authentication certificates for backend reencryption does Application Gateway support?

Application Gateway supports up to 100 authentication certificates.

Does Application Gateway natively integrate with Azure Key Vault?

Yes, the Application Gateway v2 SKU supports Key Vault. For more information, see [SSL termination with Key Vault certificates](#).

How do I configure HTTPS listeners for .com and .net sites?

For multiple domain-based (host-based) routing, you can create multisite listeners, set up listeners that use HTTPS as the protocol, and associate the listeners with the routing rules. For more information, see [Hosting multiple sites by using Application Gateway](#).

Can I use special characters in my .pfx file password?

No, use only alphanumeric characters in your .pfx file password.

Configuration - web application firewall (WAF)

Does the WAF SKU offer all the features available in the Standard SKU?

Yes. WAF supports all the features in the Standard SKU.

How do I monitor WAF?

Monitor WAF through diagnostic logging. For more information, see [Diagnostic logging and metrics for Application Gateway](#).

Does detection mode block traffic?

No. Detection mode only logs traffic that triggers a WAF rule.

Can I customize WAF rules?

Yes. For more information, see [Customize WAF rule groups and rules](#).

What rules are currently available for WAF?

WAF currently supports CRS [2.2.9](#), [3.0](#), and [3.1](#). These rules provide baseline security against most of the top-10 vulnerabilities that Open Web Application Security Project (OWASP) identifies:

- SQL injection protection
- Cross-site scripting protection
- Protection against common web attacks such as command injection, HTTP request smuggling, HTTP response splitting, and remote file inclusion attack
- Protection against HTTP protocol violations
- Protection against HTTP protocol anomalies such as missing host user-agent and accept headers
- Prevention against bots, crawlers, and scanners
- Detection of common application misconfigurations (that is, Apache, IIS, and so on)

For more information, see [OWASP top-10 vulnerabilities](#).

Does WAF support DDoS protection?

Yes. You can enable DDoS protection on the virtual network where the application gateway is deployed. This setting ensures that the Azure DDoS Protection service also protects the application gateway virtual IP (VIP).

Is there guidance available to migrate from the v1 SKU to the v2 SKU?

Yes. For details see, [Migrate Azure Application Gateway and Web Application Firewall from v1 to v2](#).

Configuration - ingress controller for AKS

What is an Ingress Controller?

Kubernetes allows creation of `deployment` and `service` resource to expose a group of pods internally in the cluster. To expose the same service externally, an `Ingress` resource is defined which provides load balancing, SSL termination and name-based virtual hosting. To satisfy this `Ingress` resource, an Ingress Controller is required which listens for any changes to `Ingress` resources and configures the load balancer policies.

The Application Gateway Ingress Controller allows [Azure Application Gateway](#) to be used as the ingress for an [Azure Kubernetes Service](#) also known as an AKS cluster.

Can a single ingress controller instance manage multiple Application Gateways?

Currently, one instance of Ingress Controller can only be associated to one Application Gateway.

Diagnostics and logging

What types of logs does Application Gateway provide?

Application Gateway provides three logs:

- **ApplicationGatewayAccessLog:** The access log contains each request submitted to the application gateway frontend. The data includes the caller's IP, URL requested, response latency, return code, and bytes in and out. The access log is collected every 300 seconds. It contains one record per application gateway.
- **ApplicationGatewayPerformanceLog:** The performance log captures performance information for each application gateway. Information includes the throughput in bytes, total requests served, failed request count, and healthy and unhealthy backend instance count.
- **ApplicationGatewayFirewallLog:** For application gateways that you configure with WAF, the firewall log contains requests that are logged through either detection mode or prevention mode.

For more information, see [Backend health, diagnostics logs, and metrics for Application Gateway](#).

How do I know if my backend pool members are healthy?

Verify health by using the PowerShell cmdlet `Get-AzApplicationGatewayBackendHealth` or the portal. For more information, see [Application Gateway diagnostics](#).

What's the retention policy for the diagnostic logs?

Diagnostic logs flow to the customer's storage account. Customers can set the retention policy based on their preference. Diagnostic logs can also be sent to an event hub or Azure Monitor logs. For more information, see [Application Gateway diagnostics](#).

How do I get audit logs for Application Gateway?

In the portal, on the menu blade of an application gateway, select **Activity Log** to access the audit log.

Can I set alerts with Application Gateway?

Yes. In Application Gateway, alerts are configured on metrics. For more information, see [Application Gateway metrics](#) and [Receive alert notifications](#).

How do I analyze traffic statistics for Application Gateway?

You can view and analyze access logs in several ways. Use Azure Monitor logs, Excel, Power BI, and so on.

You can also use 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, and HTTP status codes. For more information, in GitHub, see the [Readme file in the Resource Manager template folder](#).

What could cause backend health to return an unknown status?

Usually, you see an unknown status when access to the backend is blocked by a network security group (NSG), custom DNS, or user-defined routing (UDR) on the application gateway subnet. For more information, see [Backend health, diagnostics logging, and metrics for Application Gateway](#).

Is there any case where NSG flow logs won't show allowed traffic?

Yes. If your configuration matches following scenario, you won't see allowed traffic in your NSG flow logs:

- You've deployed Application Gateway v2
- You have an NSG on the application gateway subnet
- You've enabled NSG flow logs on that NSG

How do I use Application Gateway V2 with only private frontend IP address?

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

- Private IP and Public IP
- Public IP only

But if you'd like to use Application Gateway V2 with only private IP, you can follow the process below:

1. Create an Application Gateway with both public and private frontend IP address
2. Do not create any listeners for the public frontend IP address. Application Gateway will not listen to any traffic on the public IP address if no listeners are created for it.
3. Create and attach a [Network Security Group](#) for the Application Gateway subnet with the following configuration in the order of priority:
 - a. Allow traffic from Source as **GatewayManager** service tag and Destination as **Any** and Destination port as **65200-65535**. This port range is required for Azure infrastructure communication. These ports are protected (locked down) by certificate authentication. External entities, including the Gateway user administrators, can't initiate changes on those endpoints without appropriate certificates in place
 - b. Allow traffic from Source as **AzureLoadBalancer** service tag and Destination and destination port as **Any**
 - c. Deny all inbound traffic from Source as **Internet** service tag and Destination and destination port as **Any**. Give this rule the *least priority* in the inbound rules
 - d. Keep the default rules like allowing VirtualNetwork inbound so that the access on private IP address is not blocked
 - e. Outbound internet connectivity can't be blocked. Otherwise, you will face issues with logging, metrics, etc.

Sample NSG configuration for private IP only access:

Inbound security rules

Priority	Name	Port	Protocol	Source	Destination	Action
100	Allow_GWM	65200-65535	Any	GatewayManager	Any	Allow
110	Allow_AzureLoadBalancer	Any	Any	AzureLoadBalancer	Any	Allow
4096	DenyAllInbound_Internet	Any	Any	Internet	Any	Deny
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Outbound security rules

Priority	Name	Port	Protocol	Source	Destination	Action
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Does Application Gateway affinity cookie support SameSite attribute?

Yes, the [Chromium browser v80 update](#) introduced a mandate on HTTP cookies without SameSite attribute to be treated as SameSite=Lax. This means that the Application Gateway affinity cookie won't be sent by the browser in a third-party context. To support this scenario, Application Gateway injects another cookie called *ApplicationGatewayAffinityCORS* in addition to the existing *ApplicationGatewayAffinity* cookie. These cookies are similar, but the *ApplicationGatewayAffinityCORS* cookie has two more attributes added to it: *SameSite=None; Secure*. These attributes maintain sticky sessions even for cross-origin requests. See the [cookie based affinity section](#) for more information.

Next steps

To learn more about Application Gateway, see [What is Azure Application Gateway?](#).

Manage web traffic with an application gateway using Azure PowerShell

7/19/2019 • 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.

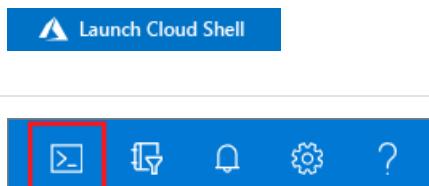
NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 -VMSScaleSetName myVmss

Add-AzVmssExtension -VirtualMachineScaleSet $vmss ` 
-Name "customScript" ` 
-Publisher "Microsoft.Compute" ` 
-Type "CustomScriptExtension" ` 
-TypeHandlerVersion 1.8 ` 
-Setting $publicSettings

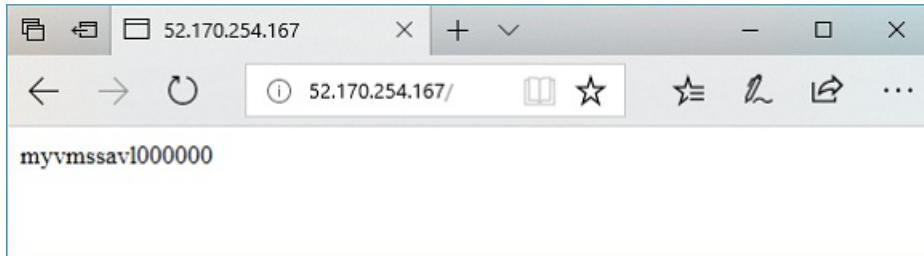
Update-AzVmss ` 
-ResourceGroupName myResourceGroupAG ` 
-Name myVmss ` 
-VirtualMachineScaleSet $vmss

```

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

7/19/2019 • 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

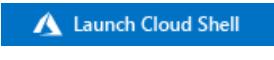
If you prefer, you can complete this procedure using [Azure PowerShell](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

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 CLI locally, this quickstart requires you to run the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

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

2/13/2020 • 5 minutes to read • [Edit Online](#)

Azure Application Gateway 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 Secure Sockets Layer (SSL) termination.

This article guides you through the steps to configure an application gateway with a frontend private IP address using the Azure portal.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>

Create an application gateway

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 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. You create 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 **(US) 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.

The screenshot shows the 'Create virtual network' wizard in the Microsoft Azure portal. The 'Address space' section displays an address range of 10.0.0.0/16, which covers 65536 addresses. The 'Subnets' section lists two subnets: 'myAGSubet' with a range of 10.0.0.0/24 and 'myBackendSubnet' with a range of 10.0.1.0/24. Both subnets currently have 0 addresses assigned.

7. Select **OK** to create the virtual network and subnet.

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 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 **Create 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.

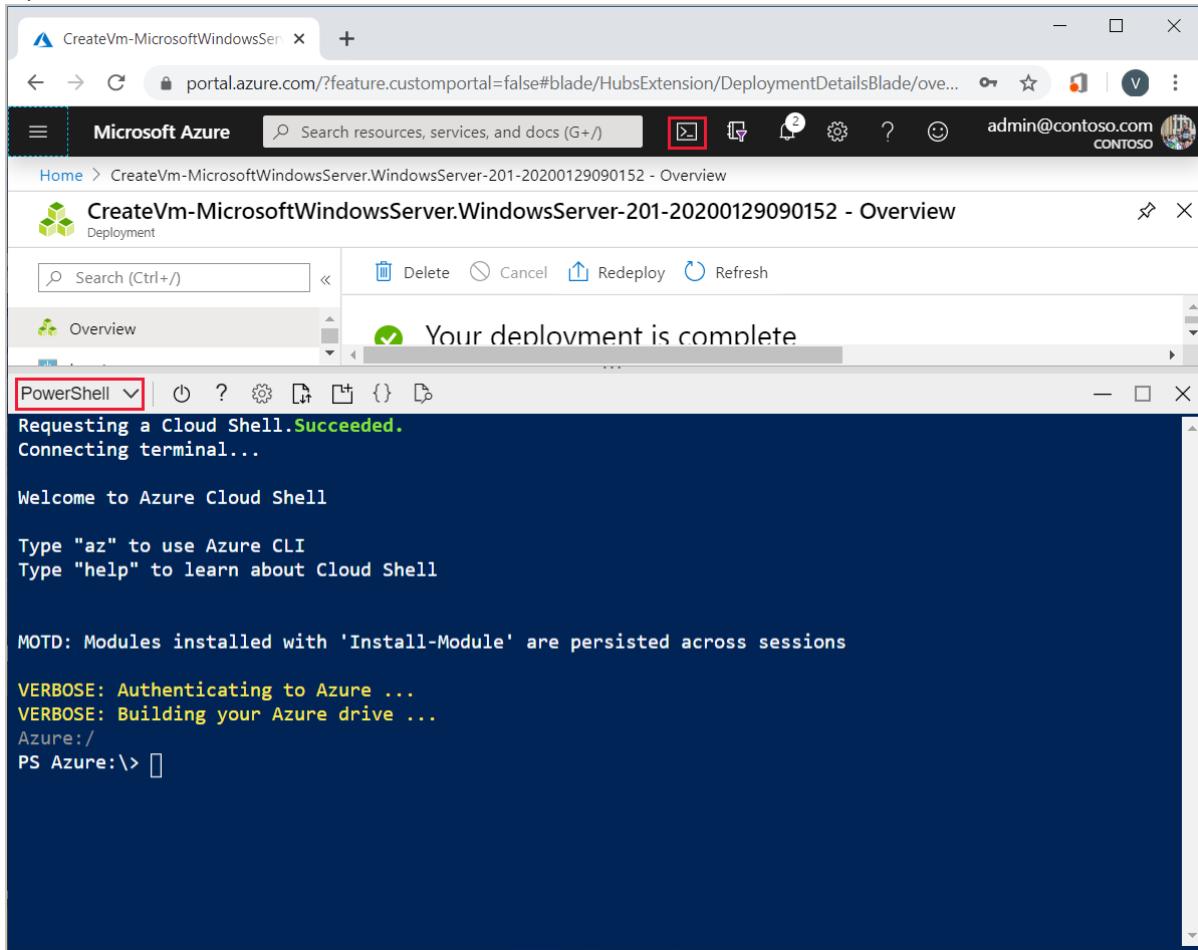
Create a virtual machine

1. Select **Create a resource**.
2. Select **Compute** and then select **Virtual machine**.
3. Enter these values for the virtual machine:
 - select *myResourceGroupAG* for **Resource group**.
 - *myVM* - for **Virtual machine name**.
 - Select **Windows Server 2019 Datacenter** for **Image**.
 - *azureadmin* - for the **Username**.
 - *Azure123456!* for the **Password**.
4. Accept the remaining defaults and select **Next : Disks**.
5. Accept the defaults and select **Next : Networking**.
6. Make sure that **myVNet** is selected for the virtual network and the subnet is **myBackendSubnet**.
7. Accept the remaining defaults, and select **Next : Management**.
8. Select **Off** to disable boot diagnostics.
9. Accept the remaining defaults, and select **Next : Advanced**.
10. Select **Next : Tags**.
11. Select **Next : Review + create**.

12. 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**.



The screenshot shows the Microsoft Azure portal with a deployment named "CreateVm-MicrosoftWindowsServer.WindowsServer-201-20200129090152 - Overview". A message at the top says "Your deployment is complete". Below it, a PowerShell session window is open. The session title is "PowerShell" and it is highlighted with a red box. The session output shows:

```
Requesting a Cloud Shell. Succeeded.  
Connecting terminal...  
  
Welcome to Azure Cloud Shell  
  
Type "az" to use Azure CLI  
Type "help" to learn about Cloud Shell  
  
MOTD: Modules installed with 'Install-Module' are persisted across sessions  
  
VERBOSE: Authenticating to Azure ...  
VERBOSE: Building your Azure drive ...  
Azure:/  
PS Azure:> [ ]
```

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. Enter myVM2 for its 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**. Select **appGatewayBackendPool**.
3. Under **Target type** select **Virtual machine** and under **Target**, select the vNIC associated with myVM.
4. Repeat to add MyVM2.

A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machine scale sets, IP addresses, or a valid Internet hostname.

Name: appGatewayBackendPool

Add backend pool without targets: Yes (selected) / No

Backend targets: 2 items

Target type	Target
Virtual machine	myvm959
Virtual machine	myvm27 (10.0.1.5)
IP address or hostname	

Associated rule: Rule-01

Buttons: Save / Cancel

5. select **Save**.

Test the application gateway

1. Check your frontend IP that got assigned by clicking the **Frontend IP Configurations** page in the portal.

Type	Status	Name	IP address
Public	Not configured	-	-
Private	Configured	appGwPrivateFrontendIp	10.1.0.6

2. Copy the private IP address, and then paste it into the browser address bar in a VM in the same VNet or on-premises that has connectivity to this VNet and try to access the Application Gateway.

Next steps

If you want to monitor the health of your backend, see [Back-end health and diagnostic logs for Application Gateway](#).

Create an application gateway with an internal load balancer (ILB)

11/19/2019 • 7 minutes to read • [Edit Online](#)

Azure Application Gateway 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 Secure Sockets Layer (SSL) termination.

This article walks you through the steps to configure an application gateway with an ILB.

Before you begin

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 orHttps, 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 appgvnet -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-e29defadfffb8
```

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 SSL termination using Azure PowerShell

11/13/2019 • 5 minutes to read • [Edit Online](#)

You can use Azure PowerShell to create an [application gateway](#) with a certificate for [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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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

```

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 -VMSScaleSetName myVmss

Add-AzVmssExtension -VirtualMachineScaleSet $vmss ` 
    -Name "customScript" 
    -Publisher "Microsoft.Compute" 
    -Type "CustomScriptExtension" 
    -TypeHandlerVersion 1.8 
    -Setting $publicSettings

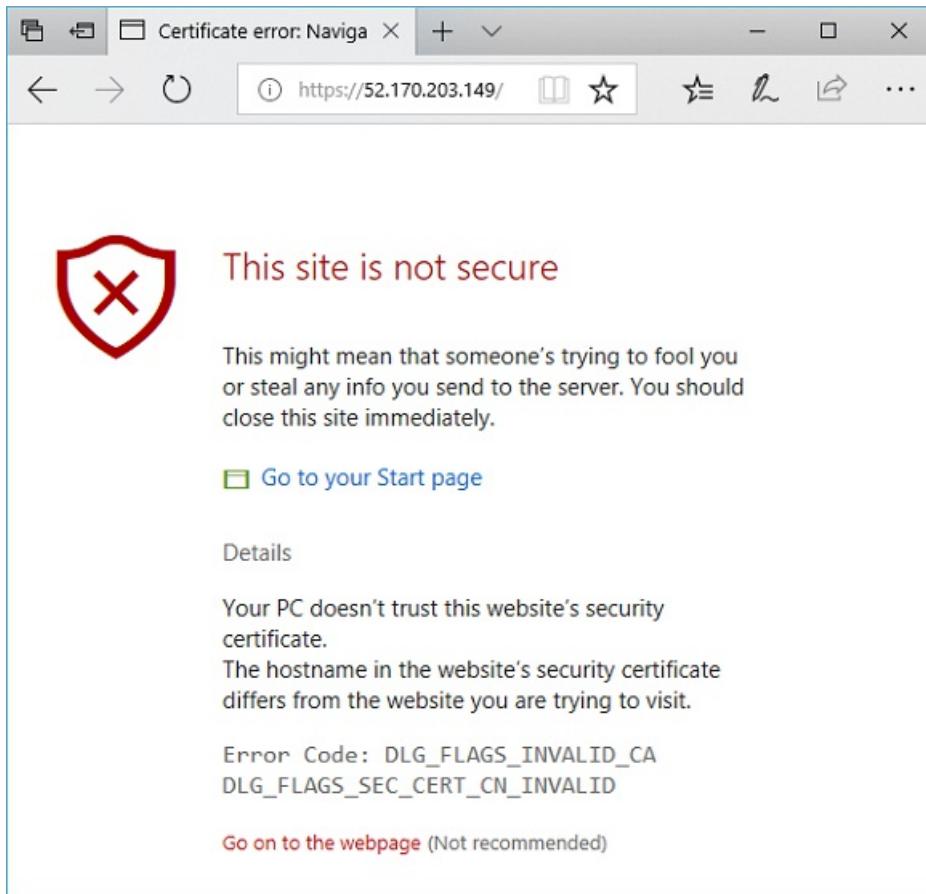
Update-AzVmss ` 
    -ResourceGroupName myResourceGroupAG 
    -Name myVmss 
    -VirtualMachineScaleSet $vmss

```

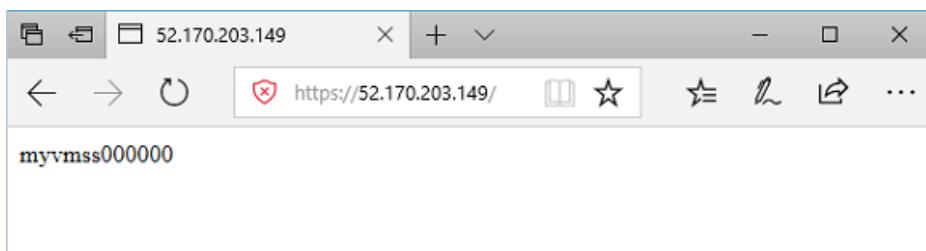
Test the application gateway

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 SSL termination using the Azure CLI

11/13/2019 • 5 minutes to read • [Edit Online](#)

You can use the Azure CLI to create an [application gateway](#) with a certificate for [SSL 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

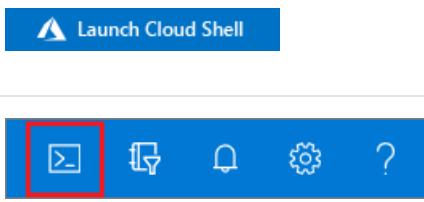
If you prefer, you can complete this procedure using [Azure PowerShell](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

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 CLI locally, this article requires you to run the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

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
```

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 \
--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

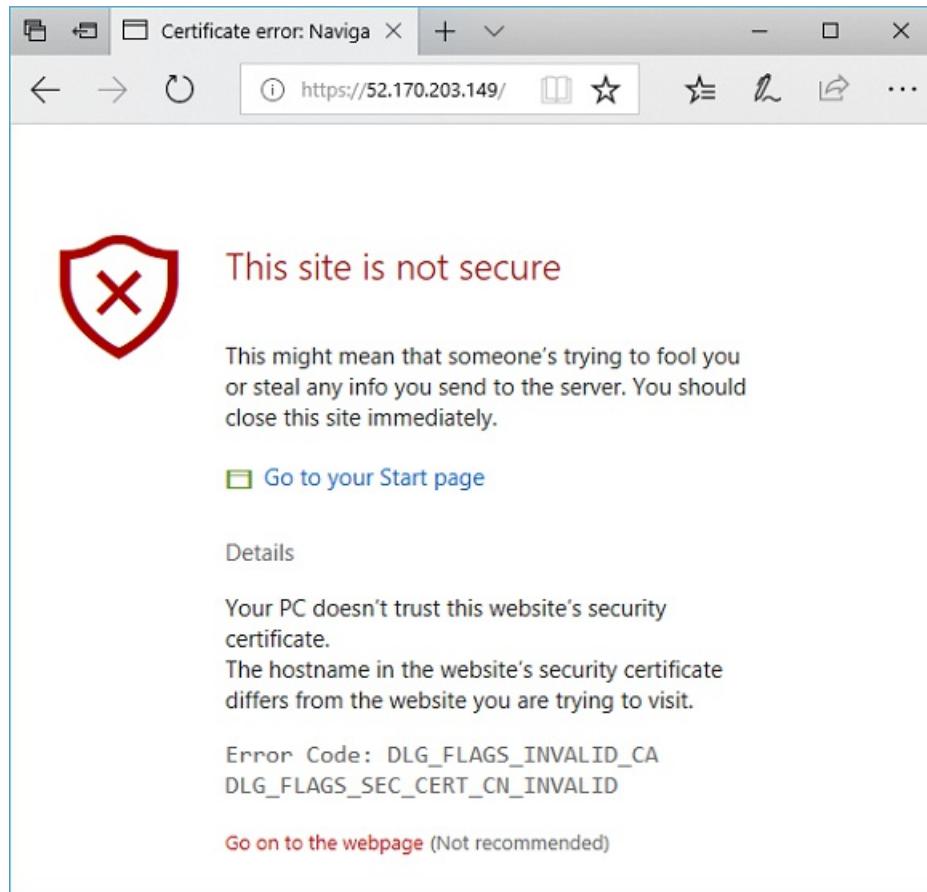
```
az vmss extension set \
--publisher Microsoft.Azure.Extensions \
--version 2.0 \
--name CustomScript \
--resource-group myResourceGroupAG \
--vmss-name myvmss \
--settings '{ \"fileUris\": [\"https://raw.githubusercontent.com/Azure/azure-docs-powershell-samples/master/application-gateway/iis/install_nginx.sh\"],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

Test the application gateway

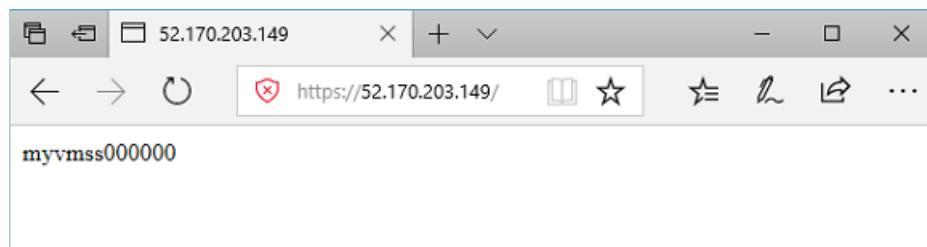
To get the public IP address of the application gateway, you can use [az network public-ip show](#).

```
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 SSL termination with Key Vault certificates by using Azure PowerShell

2/27/2020 • 2 minutes to read • [Edit Online](#)

Azure Key Vault is a platform-managed secret store that you can use to safeguard secrets, keys, and 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 [SSL 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 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 = "TestKeyVaultAppGw"  
$appgwName = "AppGwKVIntegration"
```

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 -EnableSoftDelete
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, "")

```

NOTE

The -EnableSoftDelete flag must be used for SSL termination to function properly.

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 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 SSL termination](#)

Configure end-to-end SSL by using Application Gateway with the portal

11/13/2019 • 4 minutes to read • [Edit Online](#)

This article describes how to use the Azure portal to configure end-to-end 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 SSL 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 SSL protocol specification. The symmetric key is then used to encrypt and decrypt the traffic sent to the gateway.

For end-to-end SSL 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 [SSL termination and end-to-end SSL](#).

Create a new application gateway with end-to-end SSL

To create a new application gateway with end-to-end SSL encryption, you'll need to first enable SSL termination while creating a new application gateway. This action enables SSL 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 SSL encryption for communication between the application gateway and the back-end servers. That accomplishes end-to-end SSL encryption.

Enable SSL termination while creating a new application gateway

To learn more, see [enable SSL 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.

[Home](#) > [appgw1 - HTTP settings](#) > [appGatewayBackendHttpSettings](#)

appGatewayBackendHttpSettings

appgw1

[Save](#) [Discard](#) [Delete](#)

Name
appGatewayBackendHttpSettings

*** Cookie based affinity** [i](#)
[Disabled](#) [Enabled](#)

*** Connection draining** [i](#)
[Disabled](#) [Enabled](#)

*** Protocol**
[HTTP](#) [HTTPS](#)

Use for App service

Backend authentication certificates

Create new Select existing

*** Name**

*** Upload CER certificate** [i](#)

[+ Add certificate](#)

*** Port** [i](#)
443

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.

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 

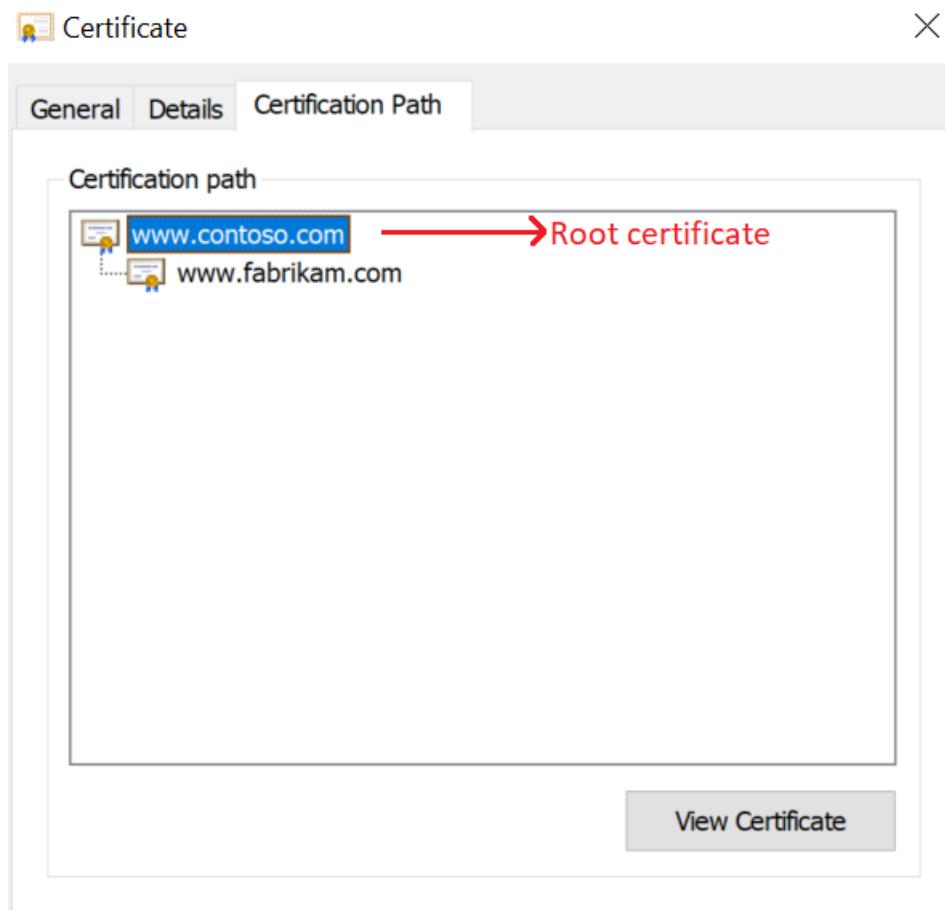
+ Add certificate

* Port 

443

* Request timeout (seconds)

30



8. Select **Save**.

Enable end-to-end SSL for an existing application gateway

To configure an existing application gateway with end-to-end SSL encryption, you must first enable SSL termination in the listener. This action enables SSL 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 SSL encryption for communication between the application gateway and the back-end servers. That accomplishes end-to-end SSL encryption.

You'll need to use a listener with the HTTPS protocol and a certificate for enabling SSL 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 SSL 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 SSL 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 SSL 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.

appGatewayBackendHttpSettings

appgw1

 Save  Discard  Delete

Name

appGatewayBackendHttpSettings

* Cookie based affinity 

Disabled Enabled

* Connection draining 

Disabled Enabled

* Protocol

HTTP HTTPS

Use for App service

Backend authentication certificates

Create new Select existing



* Name

* Upload CER certificate 

Select a file



+ Add certificate

* Port 

443



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.

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**.

Next steps

[Manage web traffic with an application gateway using the Azure CLI](#)

Configure end to end SSL by using Application Gateway with PowerShell

10/11/2019 • 11 minutes to read • [Edit Online](#)

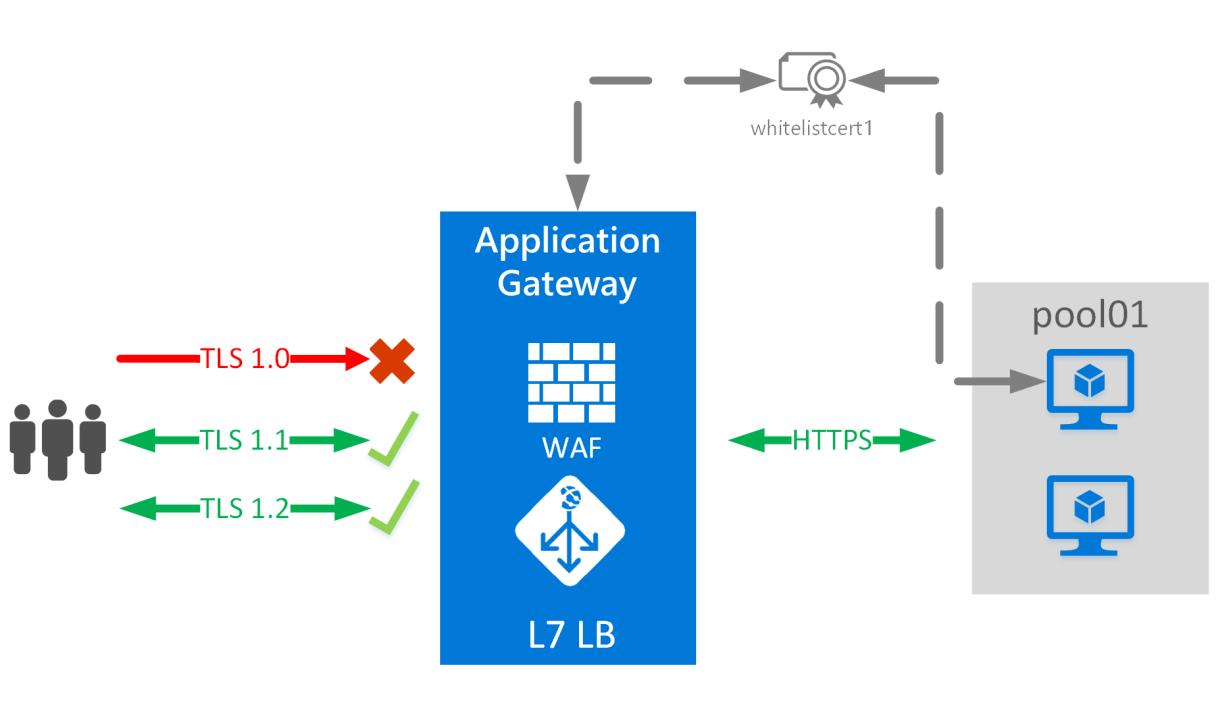
Overview

Azure Application Gateway supports end-to-end encryption of traffic. Application Gateway terminates the 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 SSL 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 SSL options, see the [SSL 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 SSL with PowerShell.

This scenario will:

- Create a resource group named **appgw-rg**.
- Create a virtual network named **appgwnet** with an address space of **10.0.0.0/16**.
- Create two subnets called **appgws subnet** and **appsubnet**.

- Create a small application gateway supporting end-to-end SSL encryption that limits SSL protocol versions and cipher suites.

Before you begin

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

To configure end-to-end SSL 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 SSL 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 SSL 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 'appgwssubnet' -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 'appgwnet' -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 'appgwnet' -ResourceGroupName appgw-rg  
$gwSubnet = Get-AzVirtualNetworkSubnetConfig -Name 'appgws subnet' -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 SSL connection. The certificate needs to be in .pfx format, and the password must be 4 to 12 characters.

6. Create the HTTP listener for the application gateway. Assign the front-end IP configuration, port, and 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 SSL-enabled back-end pool resources.

NOTE

The default probe gets the public key from the *default* SSL 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* SSL 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 SSL binding on the them. If you do not do so, probes fail and the back end is not whitelisted.

```
$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 whitelists 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 SSL 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 SSL policy to be used on the application gateway. Application Gateway supports the ability to set a minimum version for SSL 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 SSL 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 SSL protocol versions on an existing application gateway

The preceding steps took you through creating an application with end-to-end SSL and disabling certain SSL protocol versions. The following example disables certain SSL policies on an existing application gateway.

1. Retrieve the application gateway to update.

```
$gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroupName AdatumAppGatewayRG
```

2. Define an SSL 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 SSL 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-xxxxxxxxxx.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](#).

Create certificates to allow the backend with Azure Application Gateway

11/13/2019 • 3 minutes to read • [Edit Online](#)

To do end to end SSL, 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

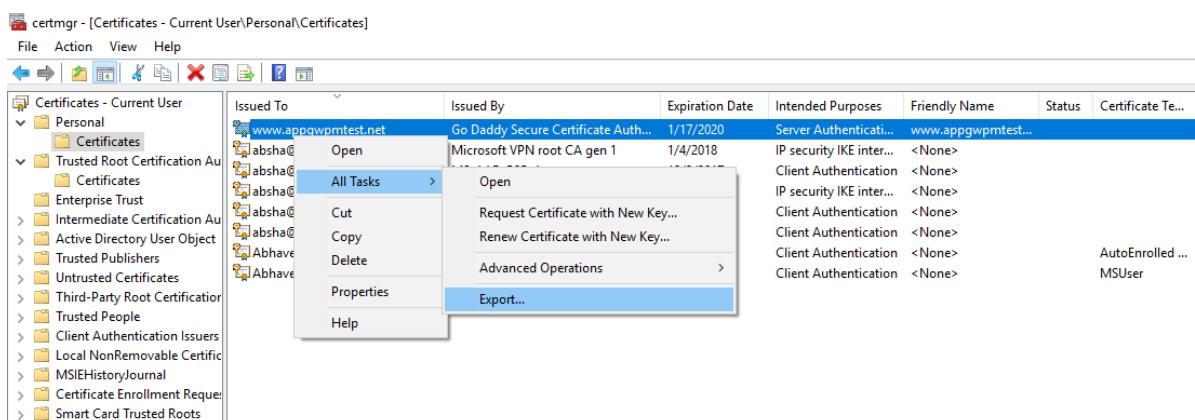
An existing backend certificate is required to generate the authentication certificates or trusted root certificates required for allowing backend instances with Application Gateway. The backend certificate can be the same as the SSL certificate or different for added security. Application Gateway doesn't provide you any mechanism to create or purchase an 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)

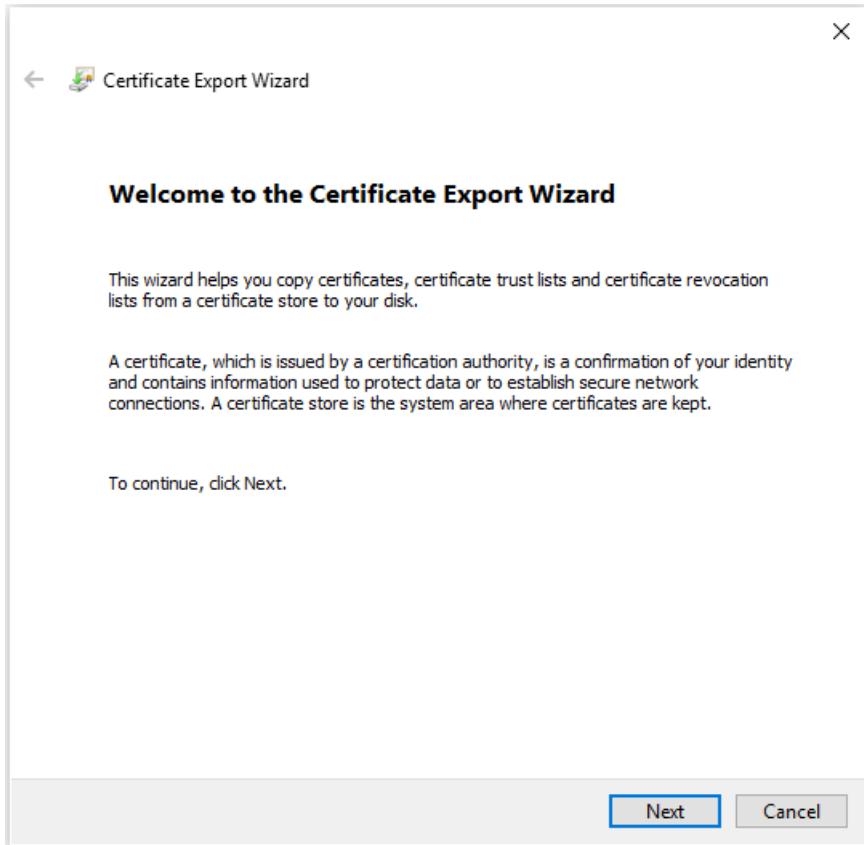
An authentication certificate is required to allow backend instances in Application Gateway v1 SKU. The authentication certificate is the public key of backend server certificates in Base-64 encoded X.509(.CER) format. In this example, you'll use an SSL certificate for the backend certificate and export 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 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:

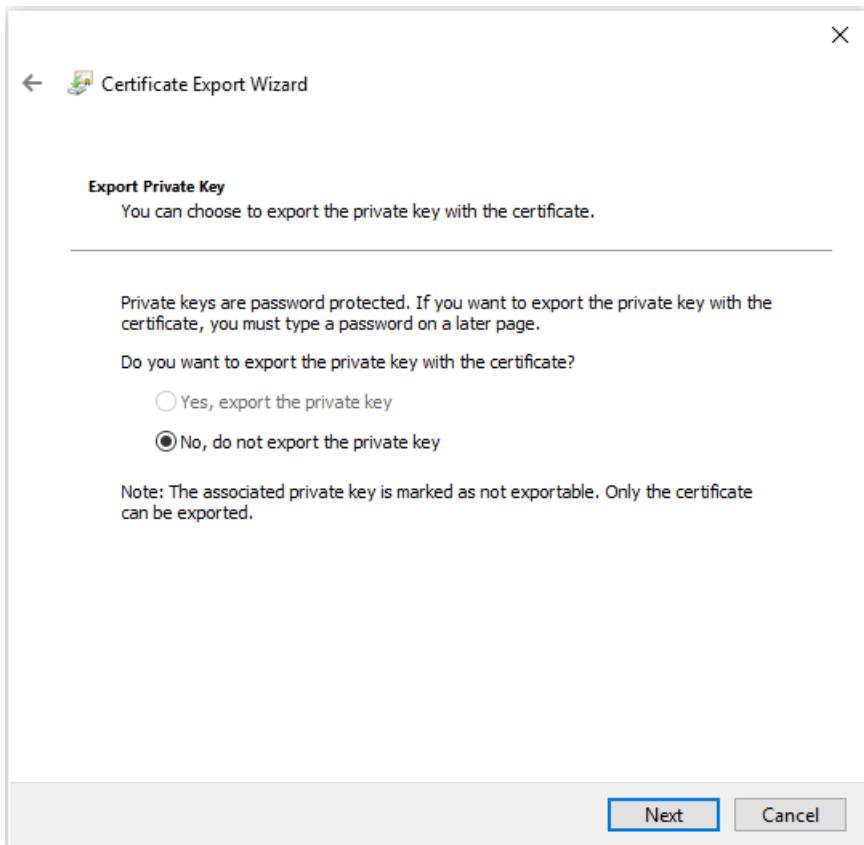
1. To obtain a .cer file from the certificate, open **Manage user certificates**. Locate the certificate, typically in 'Certificates - Current User\Personal\Certificates', and right-click. Click **All Tasks**, and then click **Export**. This opens the **Certificate Export Wizard**. If you can't find 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.



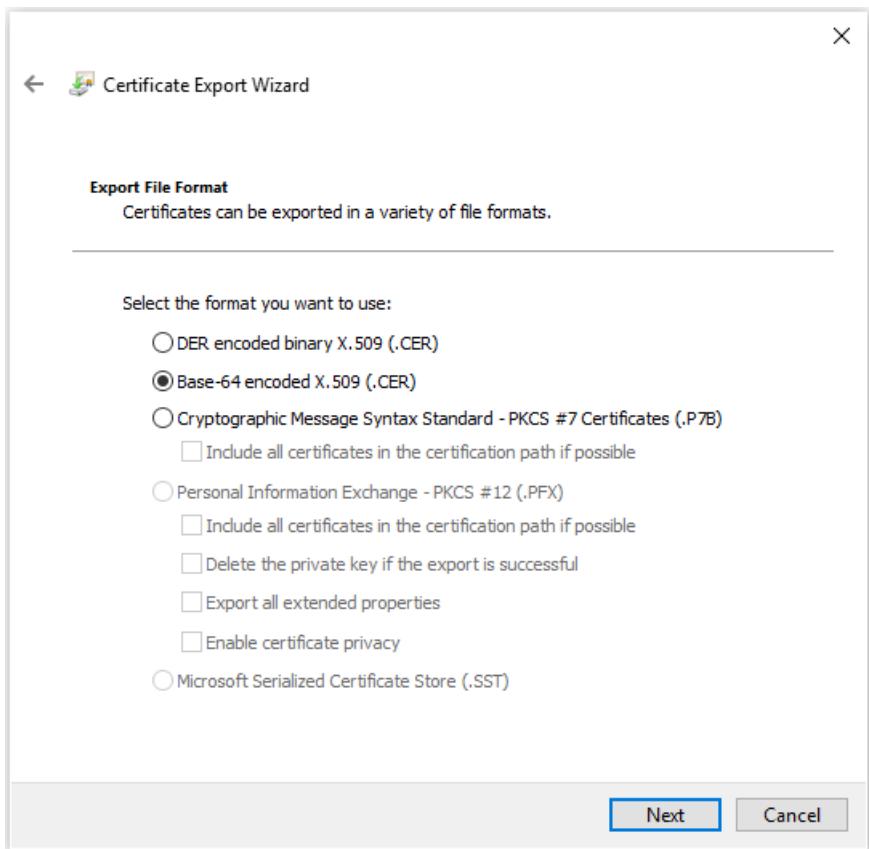
2. In the Wizard, click **Next**.



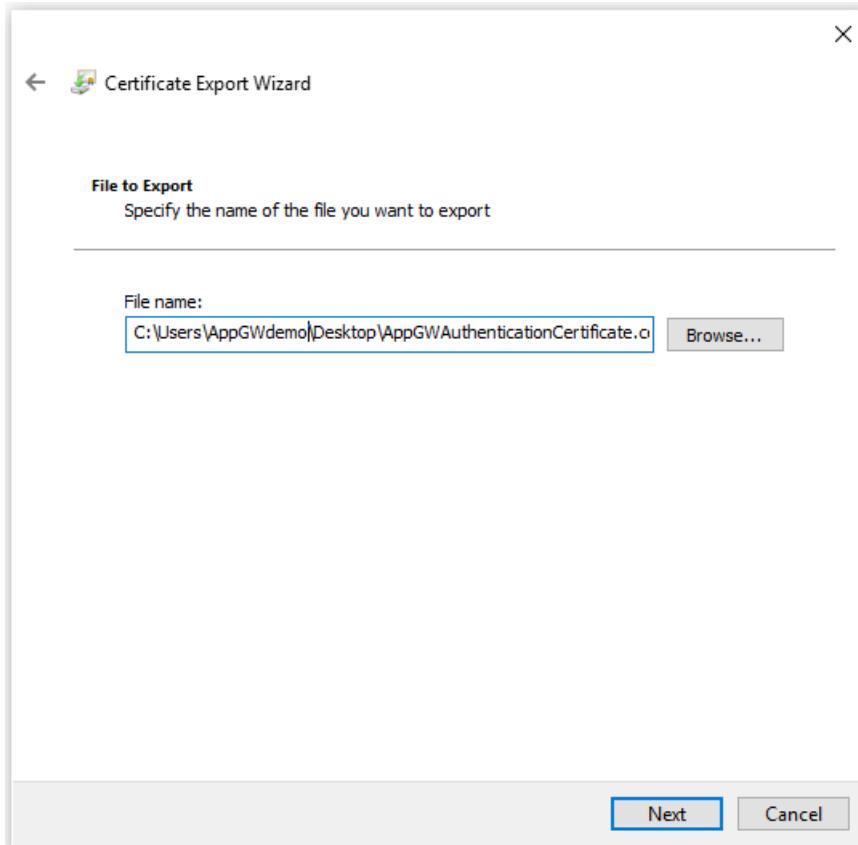
3. Select **No, do not export the private key**, and then click **Next**.



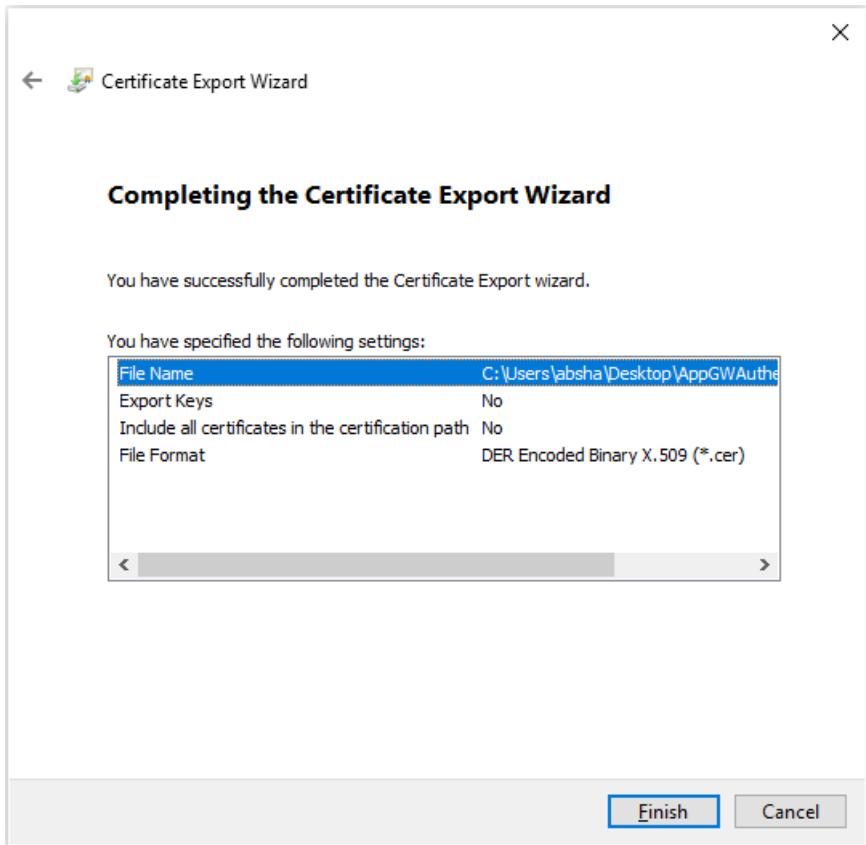
4. On the **Export File Format** page, select **Base-64 encoded X.509 (.CER)**, and then click **Next**.



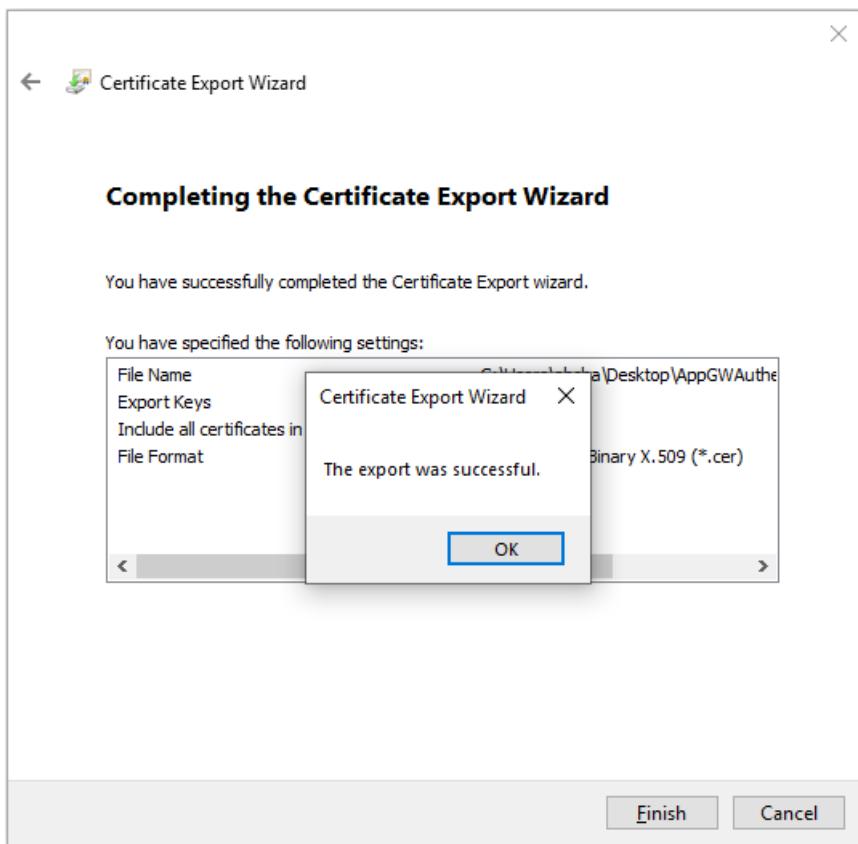
5. For **File to Export**, **Browse** to the location to which you want to export the certificate. For **File name**, name the certificate file. Then, click **Next**.



6. Click **Finish** to export the certificate.



7. Your certificate is successfully exported.

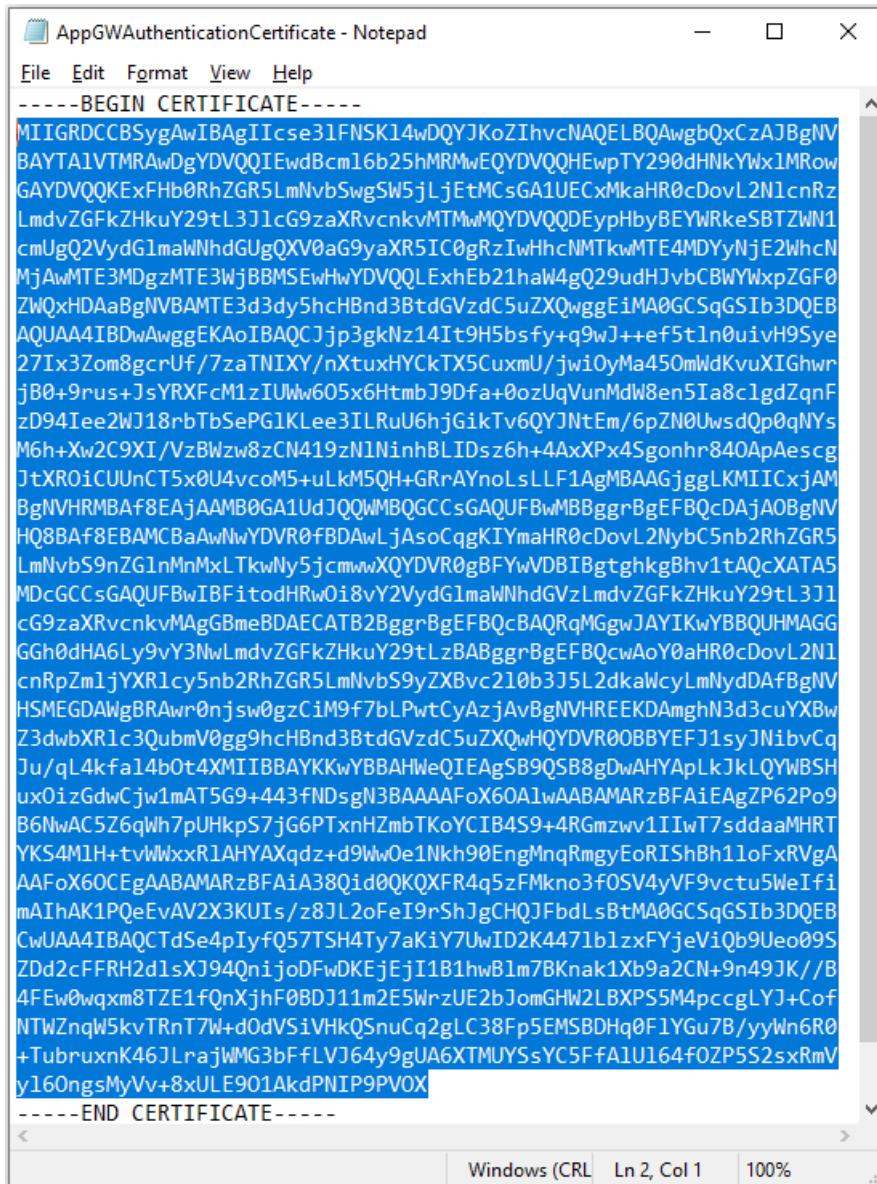


The exported certificate looks similar to this:



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.



The screenshot shows a Notepad window titled "AppGWAAuthenticationCertificate - Notepad". The content of the note pad is a long string of characters representing a Base-64 encoded X.509 (.CER) certificate. The text is divided into two sections by horizontal lines: "-----BEGIN CERTIFICATE-----" at the top and "-----END CERTIFICATE-----" at the bottom. The entire content is highlighted with a blue selection bar. At the bottom of the Notepad window, there is a status bar with the text "Windows (CRL) Ln 2, Col 1 100%".

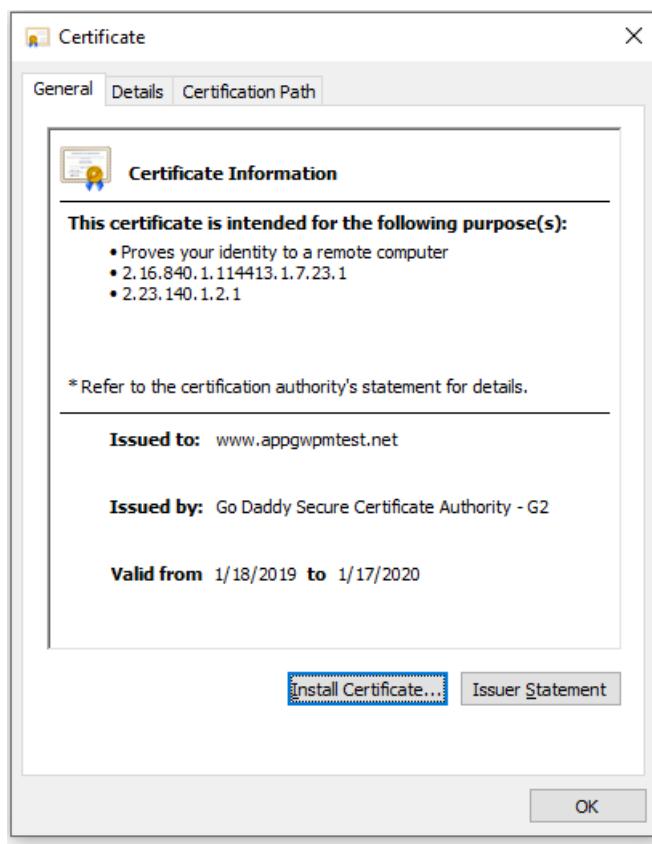
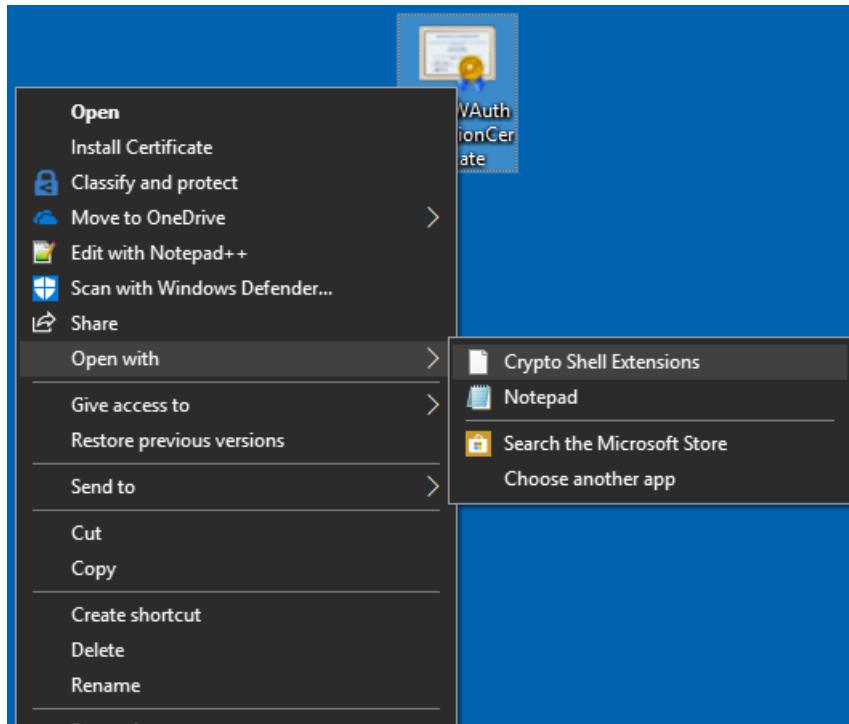
```
-----BEGIN CERTIFICATE-----
MIIGRDCCBSygAwIBAgIIcse31FNSK14wDQYJKoZIhvcNAQELBQAwgbQxCzAJBgNV
BAYTA1VTMRAwDgYDVQQIEwdBcm16b25hMRMwEQYDVQQHEwpTY290dHNkYWx1MRow
GAYDVQQKExFhb0RhZGR5LmNvbSwgSW5jLjEtMCsGA1UECxMkaHR0cDovL2N1cnRz
LmdvZGFkZHkuY29tL3J1cG9zaXRvcnkvMTMwMQYDVQQDEypHbyBEYWRkeSBTZWNN
cmUgQ2VydG1maWNhdGUgQXV0aG9yaXR5IC0gRzIwHhcNMTkwMTE4MDYyNjE2WhcN
MjAwMTE3MDgzMTE3WjBBMSEwHwYDVQQLExhEb21haW4gQ29udHJvbCBWYxpZGF0
ZWQxHDAaBgNVBAMTE3d3dy5hcHBnd3BtdGVzdC5uZXQwggEiMA0GCSqGSIb3DQE
AQUAA4IBDwAwggEKAoIBAQCJjp3gkNz14It9H5bsfy+q9wJ++ef5tln0uivH9Sye
27Ix3Zom8gcrUf/7zaTNIYX/nXtuxHYCkTX5CuxmU/jwi0yMa450mWdKvuXIGhwr
jB0+9rus+JsYRXFcM1zIUlw605x6HtmbJ9Dfa+0ozUqVunMdW8en5Ia8c1gdZqnF
zD94Iee2Wj18rbTbSePG1KLee3ILRuU6hjGikTv6QYJNtEm/6pZN0UwsdQp0qNYs
M6h+Xw2C9XI/VzBWzw8zCN419zN1NinhBLIDsz6h+4AxXPx4Sgonhr840ApAescg
JtXROiCUUnCT5x0U4vcoM5+uLkM5QH+GRrAYnolLLF1AgMBAAGjggLKMIIcxjAM
BgNVHRMBAf8EAjAAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggRgEFBQcDAjAOBgNV
HQ8BAf8EBAMCBaAwNvYDVR0fBDAwLjAsoCqgKIYmaHR0cDovL2Nybc5nb2RhZGR5
LmNvbS9nZG1nMnMxLTkwMy5jcmwwXQYDVR0gBFYwVDBIBgtghkgBhv1tAQcXATA5
MDcGCCsGAQUFBwIBFitodHRwOi8vY2VydG1maWNhdGVzLmdvZGFkZHkuY29tL3J1
cG9zaXRvcnkvMAgGBmeBDAECATB2BggRgEFBQcBAQRqMGgwJAYIKwYBBQUHMAGG
GGh0dHA6Ly9vY3NwLmdvZGFkZHkuY29tLzBABggRgEFBQcwAoY0aHR0cDovL2N1
cnRpZmljYXR1cy5nb2RhZGR5LmNvbS9yZXBvc210b3J5L2dkaWcyLmNydaFBgNV
HSMEGDAwBgBRAwr0njsw0gzCiM9f7bLPwtCyAzjAvBgNVHREEKDAmghN3d3cuYXBw
Z3dwbXR1c3QubmV0gg9hC8Bnd3BtdGVzdC5uZXQwHQYDVR0OBBYEJ1syJNibvCq
Ju/qL4kfal4b0t4XMIIBAYKKwYBBAHwQIEAgSB9QSB8gDwAHYApLkJKLQyWBSH
ux0izGdwCjw1mAT5G9+443fNDsgN3BAAAFAoX60AlwAABAMARzBFAiEAgZP62Po9
B6NwAC5Z6qWh7pUhkpS7jG6PTxnHzmbTKoYCIB4S9+4RGmzwv1IIwT7sddaaMHRT
YKS4M1H+tvWlwxR1AHYAXqdz+d9Ww0e1Nkh90EngMnqRmgYeoRIShBh1loFxRVgA
AAFoX60CEgAABAMARzBFAiA38Qid0QKQXFR4q5zFMkno3fOSV4yVF9vctu5WeIfi
mAiAK1PQeEvAV2X3KUIs/z8JL2oFeI9rShJgCHQJFbdLsBtMA0GCSqGSIb3DQE
CwUAA4IBAQCTdSe4pIyfQ57TS4Ty7aKiY7UwID2K4471b1zxFYjeViQb9Ueo09S
ZDd2cFFRH2d1sXJ94QnijoDFwDKEjEjI1B1hwB1m7BKnak1Xb9a2CN+9n49JK//B
4FEw0wqxm8TZE1fQnXjhF0BDJ11m2E5WrzUE2bJomGHw2LBXP5M4pcchgLYJ+Cof
NTWZnqW5kvTrnT7W+d0dVSiVhkQSnuCq2gLC38Fp5EMSBDHq0F1YGu7B/yyWn6R0
+TuBruxnK46JLrajWMG3bFfLVJ64y9gUA6XTMUYssYC5FFA1U164f0ZP5S2sxRmV
y160ngsMyVv+8xULE901AkdpNIP9PVOX
-----END CERTIFICATE-----
```

Export trusted root certificate (for v2 SKU)

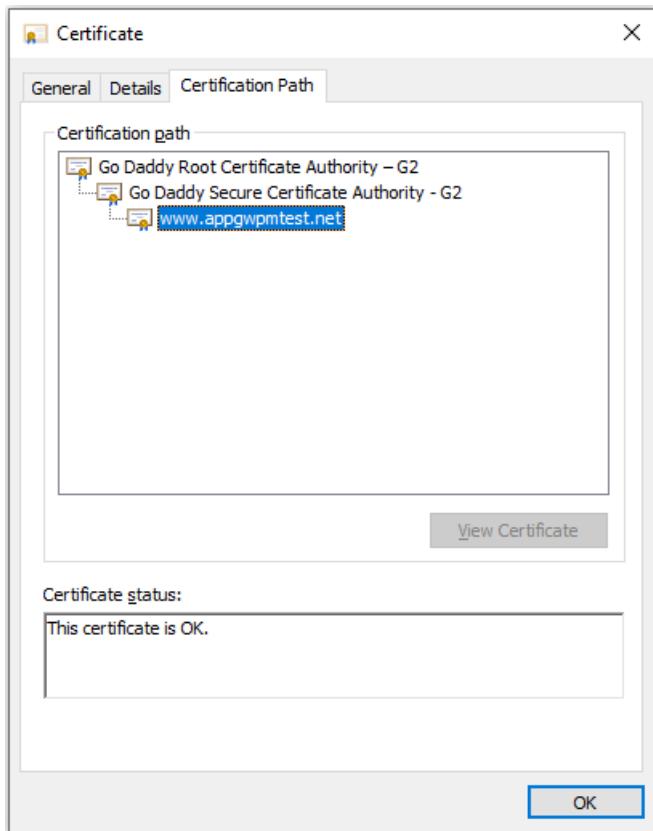
Trusted root certificate is required to whitelist backend instances in application gateway v2 SKU. The root certificate is a Base-64 encoded X.509(.CER) format root certificate from the backend server certificates. In this example, we will use an SSL certificate for the backend certificate, export its 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:

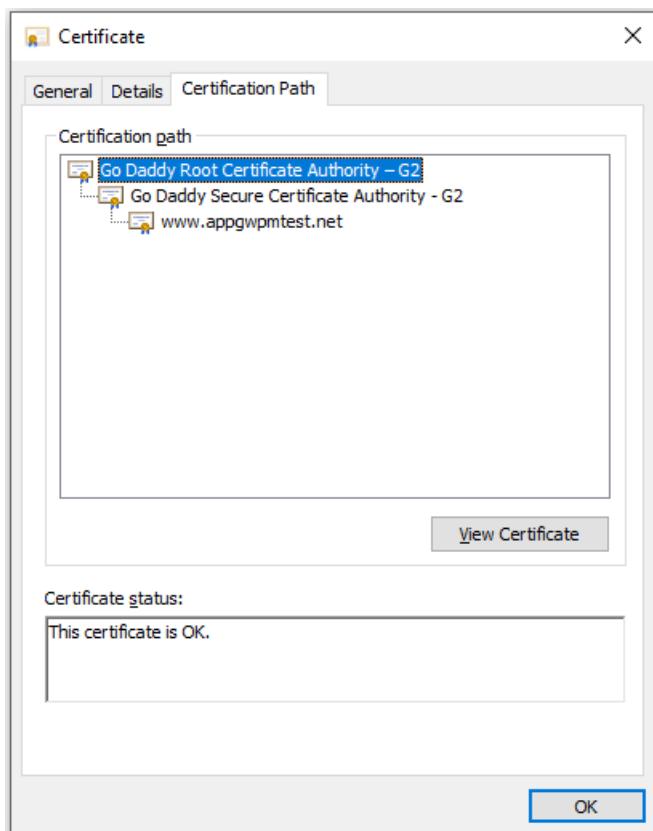
1. Use the steps 1-9 mentioned in the section **Export authentication certificate from a backend certificate (for v1 SKU)** above to export the public key from your backend certificate.
2. Once the public key has been exported, open the file.



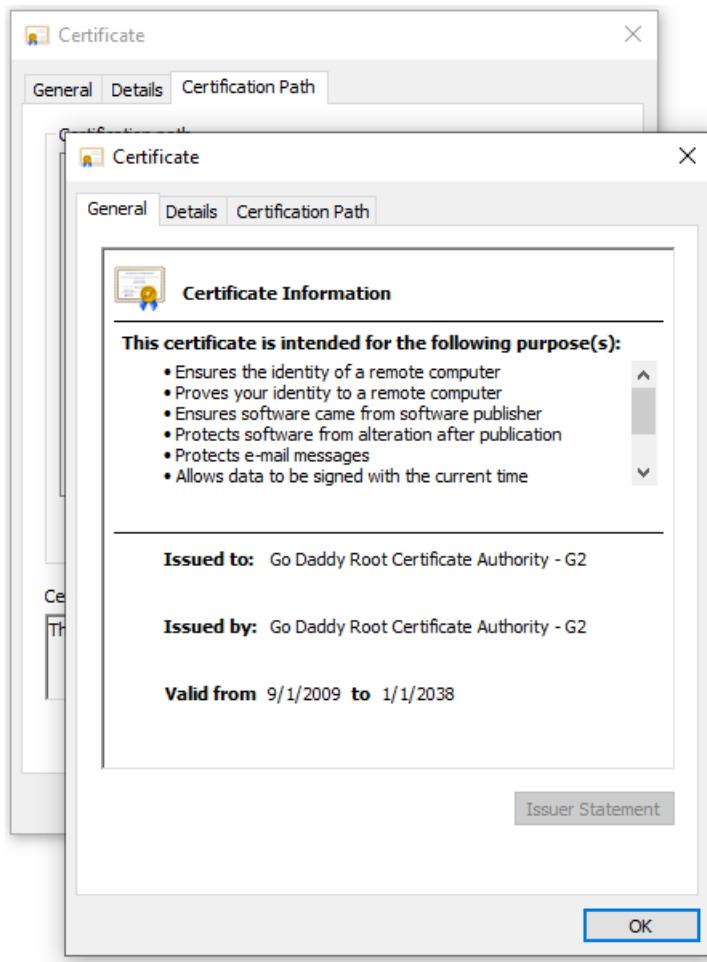
3. Move to the Certification Path view to view the certification authority.



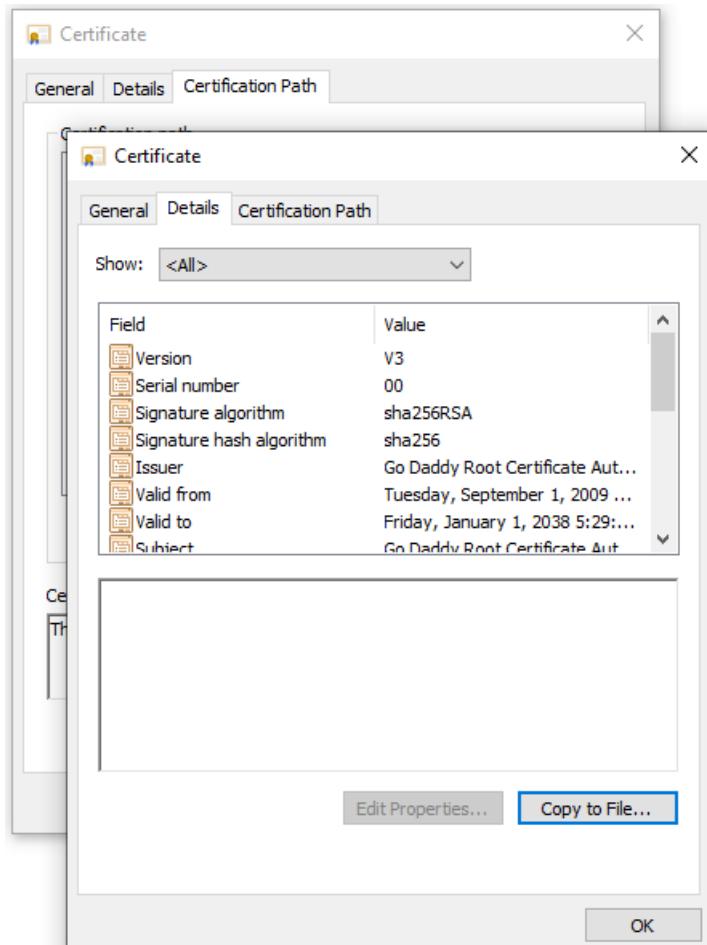
4. Select the root certificate and click on **View Certificate**.



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 whitelist your backend servers for end to end SSL encryption. See [how to configure end to end SSL encryption](#).

Renew Application Gateway certificates

1/19/2020 • 2 minutes to read • [Edit Online](#)

At some point, you'll need to renew your certificates if you configured your application gateway for SSL encryption.

You can renew a certificate associated with a listener using either the Azure portal, Azure PowerShell, or Azure CLI:

Azure portal

To renew a listener certificate from the portal, navigate to your application gateway listeners. Click the listener that has a certificate that needs to be renewed, and then click **Renew or edit selected certificate**.

The screenshot shows the Azure portal interface for managing an Application Gateway listener. The top navigation bar includes 'Home', 'Resource groups', 'myResourceGroupAG', 'myAppGateway - Listeners', and 'AGHTTPSLListener'. Below the navigation is a toolbar with 'Save', 'Discard', and 'Delete' buttons. The main configuration area contains the following fields:

- Name:** AGHTTPSLListener
- * Frontend IP configuration:** myAGFrontendIPConfig
- * Frontend port:** AGFrPort (443)
- Protocol:** HTTPS
- * Certificate:** MyAGCert

Below these fields is a button labeled **Renew or edit selected certificate**, which is highlighted with a red box. Underneath this button are three additional fields:

- * Upload PFX certificate**: A file input field with the placeholder "Select a file".
- * Name:** MyAGCert
- * Password:** An empty password field.

At the bottom left, there is a section for "Associated rule" with a link to "rule1".

Upload your new PFX certificate, give it a name, type the password, and then click **Save**.

Azure PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 SSL Offloading with Azure Application Gateway, see [Configure SSL Offload](#)

Generate an Azure Application Gateway self-signed certificate with a custom root CA

12/10/2019 • 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 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 SSL termination and end to end SSL 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 a password protected key.

```
openssl ecparam -out contoso.key -name prime256v1 -genkey
```

- At the prompt, type a strong password. For example, at least nine characters, using upper case, lower case, numbers, and symbols.

Create a Root Certificate and self-sign it

- Use the following commands to generate the csr and the certificate.

```
openssl req -new -sha256 -key contoso.key -out contoso.csr  
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.

- When prompted, type the password for the root key, and the organizational information for the custom CA such as Country, State, Org, OU, and the fully qualified domain name (this is the domain of the issuer).

```
surajmb@testliboutvml:~$ openssl req -new -sha256 -key contoso.key -out contoso.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]:US  
State or Province Name (full name) [Some-State]:Texas  
Locality Name (eg, city) []:Austin  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MS  
Organizational Unit Name (eg, section) []:DC  
Common Name (e.g. server FQDN or YOUR name) []:www.contoso.com  
Email Address []:user@contoso.com  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:a1b2c3d4  
An optional company name []:a1b2c3d4
```

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`.

- Use the following command to generate the CSR:

```
openssl req -new -sha256 -key fabrikam.key -out fabrikam.csr
```

- When prompted, type the password for the root key, and the organizational information for the custom CA:

Country, 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@testlaboutvm1:~$ 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@testlaboutvm1:~$ 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 SSL settings

In your web server, configure SSL 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 SSL 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>
```

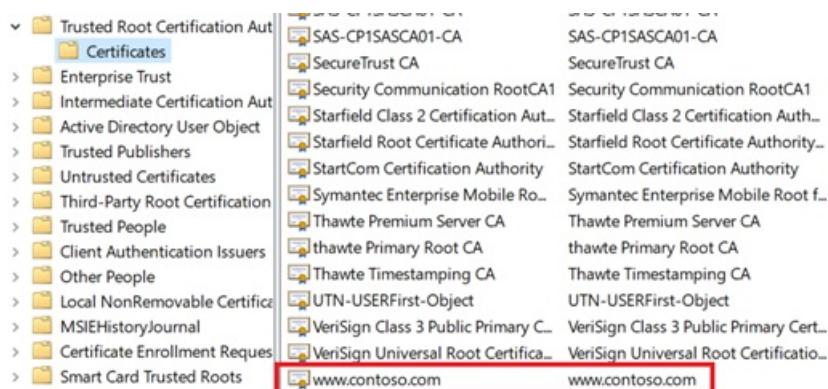
NGINX

The following configuration is an example [NGINX server block](#) with SSL 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
```

```
sura:jmb@testlbtoutvm1:~$ 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.com
verify 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-----
MIICAzCCAggGCCQChcAhn/KbUTDAKBggqhkOPQQDajCBgzELMAkGAs1UEBhMCVVMx
DjANBgNVBAgTBVRleGFzQ8oQYDVRQHDAZBxx0eIw4xCzABgNVBAoMAk1TMQsw
-----END CERTIFICATE-----
```

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.

The screenshot shows the 'appgwv2 - HTTP settings > testbackend' page in the Azure portal. The 'Protocol' dropdown is set to 'HTTPS'. In the 'Trusted Root certificates' section, there is a table with one row for 'contosorootcer'. Below it, there are two radio buttons: 'Create new' (selected) and 'Select existing'. A text input field contains 'TrustedSelfSignedRoot'. Below that is a file input field containing 'contoso.cer'. There is also a '+ Add certificate' button. At the bottom of the form, there are fields for 'Port' (443) and 'Request timeout (seconds)' (20).

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 SSL 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.

52.229.169.132 (testbackendpool)	443 (testbackend)	Healthy	Success
----------------------------------	-------------------	----------------------	---------

Next steps

To learn more about SSL\TLS in Application Gateway, see [Overview of SSL termination and end to end SSL with Application Gateway](#).

Configure SSL policy versions and cipher suites on Application Gateway

11/13/2019 • 4 minutes to read • [Edit Online](#)

Learn how to configure SSL policy versions and cipher suites on Application Gateway. You can select from a list of predefined policies that contain different configurations of SSL policy versions and enabled cipher suites. You also have the ability to define a [custom SSL policy](#) based on your requirements.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Get available SSL 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/147a22e9-2356-4e56-b3de-
    1f5842ae4a3b/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/ApplicationGatewaySslPredefinedPolicy/AppGwSslPolicy20150501
    /subscriptions/147a22e9-2356-4e56-b3de-
    1f5842ae4a3b/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/ApplicationGatewaySslPredefinedPolicy/AppGwSslPolicy20170401
    /subscriptions/147a22e9-2356-4e56-b3de-
    1f5842ae4a3b/resourceGroups//providers/Microsoft.Network/ApplicationGatewayAvailableSslOptions/default/ApplicationGatewaySslPredefinedPolicy/AppGwSslPolicy20170401S

AvailableCipherSuites:
    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

```

List pre-defined SSL Policies

Application gateway comes with three 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 an SSL policy on your application gateway. By default **AppGwSslPolicy20150501** is selected if no specific SSL 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 SSL policy

When configuring a custom SSL 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 SSL 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`

IMPORTANT

`TLS_RSA_WITH_AES_256_CBC_SHA256` must be selected when configuring a custom SSL policy. Application gateway uses this cipher suite for backend management. You can use this in combination with any other suites, but this one must be selected as well.

```

# get an application gateway resource
$gw = Get-AzApplicationGateway -Name AdatumAppGateway -ResourceGroup AdatumAppGatewayRG

# set the SSL 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",
  "TLS_RSA_WITH_AES_128_GCM_SHA256"

# validate the SSL policy locally
Get-AzApplicationGatewaySslPolicy -ApplicationGateway $gw

# update the gateway with validated SSL policy
Set-AzApplicationGateway -ApplicationGateway $gw

```

Create an application gateway with a pre-defined SSL policy

When configuring a Predefined SSL 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 SSL 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 SSL
$fp = New-AzApplicationGatewayFrontendPort -Name frontendport01 -Port 443

# Upload an existing pfx certificate for SSL 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 -ProtocolHttps -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 SSL 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 SSL policy

To set a custom SSL policy, pass the following parameters: **PolicyType**, **MinProtocolVersion**, **CipherSuite**, and **ApplicationGateway**. To set a Predefined SSL 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.

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.  
  
# SSL 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  
  
# SSL Predefined Policy  
# Set-AzApplicationGatewaySslPolicy -PolicyType Predefined -PolicyName "AppGwSslPolicy20170401S" -  
ApplicationGateway $AppGw  
  
# Update AppGW  
# The SSL 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.

Install an Application Gateway Ingress Controller (AGIC) using an existing Application Gateway

11/7/2019 • 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 Advanced Networking](#) enabled
- [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:

- *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
```

- 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 --subscription <subscription-uuid> --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 accessible 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> --sdk-auth | base64 -w0" >>

#####
# Specify if the cluster is 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 Identity 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 --subscription <subscription-uuid> --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
```

- 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.

- 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
```

- View the newly created object:

```
kubectl get AzureIngressProhibitedTargets
```

- 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

11/7/2019 • 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:

 [Launch Cloud Shell](#)

Alternatively, launch Cloud Shell from Azure portal using the following icon:



Your [Azure Cloud Shell](#) already has all necessary tools. Should you choose to use another environment, please 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 (AAD) [service principal object](#). Please record the `appId`, `password`, and `objectId` values - these will be used in the following steps.

1. Create AD service principal ([Read more about RBAC](#)):

```
az ad sp create-for-rbac --skip-assignment -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 **RBAC** enabled cluster, set the `aksEnabledRBAC` field to `true`

Deploy Components

This step will add the following components to your subscription:

- [Azure Kubernetes Service](#)
- [Application Gateway v2](#)
- [Virtual Network with 2 subnets](#)
- [Public IP Address](#)
- [Managed Identity](#), which will be used by [AAD 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`. This may 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 group deployment 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 group deployment 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 are 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 will use to connect to our new Kubernetes cluster. [Cloud Shell](#) has [kubectl](#) already installed. We will 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 AAD Pod Identity

Azure Active Directory Pod Identity provides token-based access to [Azure Resource Manager \(ARM\)](#).

[AAD 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 AAD Pod Identity to your cluster:

- *RBAC enabled* AKS cluster

```
```bash
kubectl create -f https://raw.githubusercontent.com/Azure/aad-pod-identity/master/deploy/infra/deployment-
rbac.yaml
```

```

- *RBAC disabled* AKS cluster

```
```bash
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 will leverage it to install the

[application-gateway-kubernetes-ingress](#) package:

1. Install [Helm](#) and run the following to add [application-gateway-kubernetes-ingress](#) helm package:

- *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
```

- *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 accessible 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> --sdk-auth | base64 -w0" >>

#####
# Specify if the cluster is 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 name space, which AGIC should watch. This could 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. See below for more information on Identity
- `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 [Create an Identity](#) 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.

Enable Cookie based affinity with an Application Gateway

11/7/2019 • 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

11/7/2019 • 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: `f1-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 accessible 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> -sdk-auth | base64 -w0" ></>

#####
# Specify if the cluster is 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

11/7/2019 • 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

11/7/2019 • 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](#)
- [HttpGet Action](#)

### 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` are not supported.

## Without `readinessProbe` Or `livenessProbe`

If the above probes are not provided, then Ingress Controller make an assumption that the service is reachable on `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 not 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

11/7/2019 • 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 are 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 will need a 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 is not accessible from outside the AKS cluster. We will 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 will 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 this 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 additional paths into this ingress and redirect those paths to other services:

```
```yaml
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

11/7/2019 • 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 we begin the upgrade procedure, ensure that you have 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 |
| rc1           | default   |                          |        |                         |           |

The Helm chart installation from the sample response above is named `odd-billygoat`. We will use this name for the rest of the commands. Your actual deployment name will most likely differ.

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

Should the Helm deployment fail, you can rollback 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             |

From the sample output of the `helm history` command it looks like the last successful deployment of our `odd-billygoat` was revision `1`

2. Rollback to the last successful revision:

```
helm rollback odd-billygoat 1
```

# Use certificates with LetsEncrypt.org on Application Gateway for AKS clusters

11/12/2019 • 3 minutes to read • [Edit Online](#)

This section configures your AKS to leverage [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 install \
 --name cert-manager \
 --namespace cert-manager \
 --version v0.8.0 \
 jetstack/cert-manager
```

## 2. ClusterIssuer Resource

Create a `ClusterIssuer` resource. It is required by `cert-manager` to represent the `Lets Encrypt` certificate authority where the signed certificates will be obtained.

By 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](https://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 cert authority. The staging certificate is issued by `CN=Fake LE Intermediate X1`. This is an indication that the system worked as expected and you are ready for your production certificate.

#### 4. Production Certificate

Once your staging certificate is setup successfully you can switch to a production ACME server:

- Replace the staging annotation on your Ingress resource with:

`certmanager.k8s.io/cluster-issuer: letsencrypt-prod`

- 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 is using to configure the Application Gateway.

# Expose a WebSocket server to Application Gateway

11/7/2019 • 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:

```

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)

1/23/2020 • 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
applicationGatewayId
```

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 principle 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://<application-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

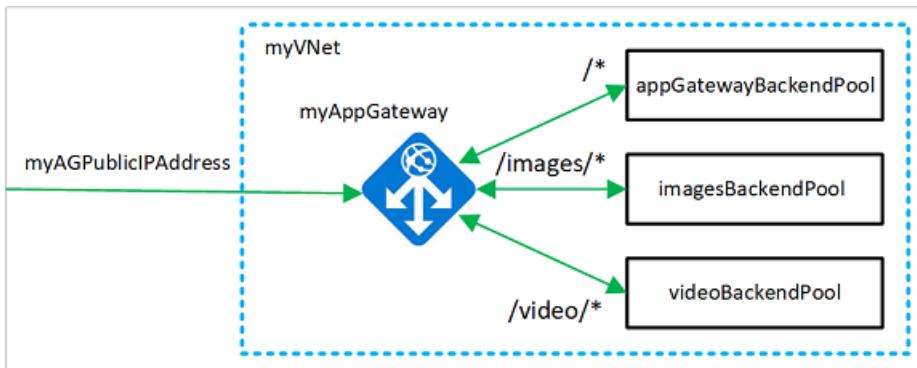
10/31/2019 • 9 minutes to read • [Edit Online](#)

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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.            |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |  |

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 [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[0].Id `
-BackendAddressPoolName $poolId `
-Protocol Tcp `
-Port 80 `
-LoadBalancerSku Standard `
-Primary>true
```

```

-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

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

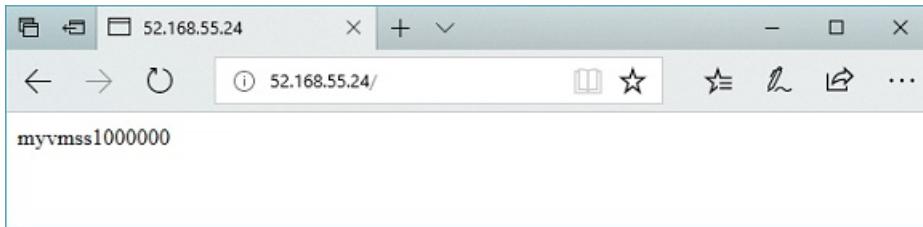
 Update-AzVmss `
 -ResourceGroupName myResourceGroupAG `
 -Name myVmss$i `
 -VirtualMachineScaleSet $vmss
}

```

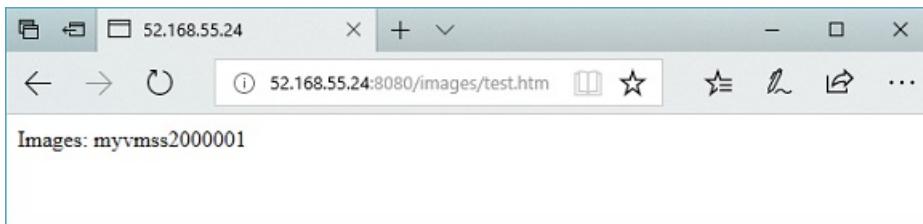
## Test the application gateway

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`.

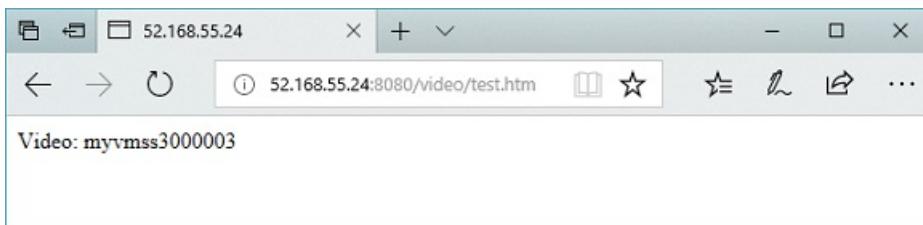
```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```



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
```

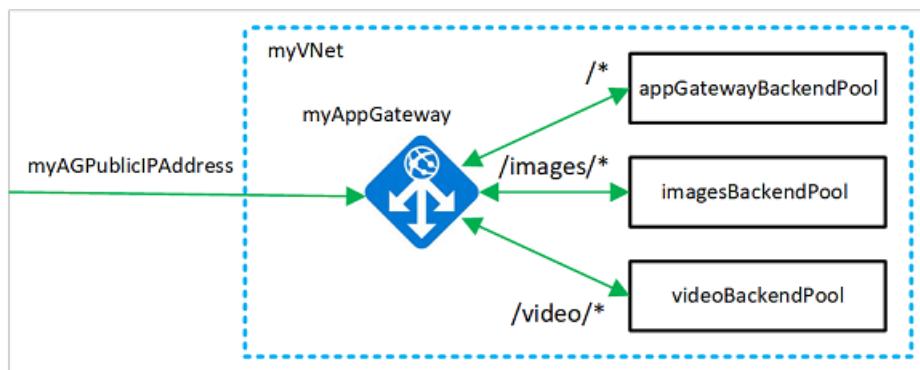
## Next steps

[Redirect web traffic based on the URL](#)

# Route web traffic based on the URL using the Azure CLI

7/31/2019 • 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 a [free account](#) before you begin.

## 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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.            | <div style="border: 1px solid #ccc; padding: 5px; display: flex; align-items: center;"><span style="margin-right: 10px;">Azure CLI</span><span style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 3px; margin-right: 10px;">Copy</span><span style="border: 2px solid #0078D4; color: #0078D4; padding: 2px 5px; border-radius: 3px; font-weight: bold;">Try It</span></div> |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. | <a href="#" style="border: 1px solid #0078D4; color: white; padding: 5px 10px; border-radius: 3px; font-weight: bold;">Launch Cloud Shell</a>                                                                                                                                                                                                                                             |

| OPTION                                                                                                        | EXAMPLE/LINK                                                                       |
|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> . |  |

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 CLI locally, this article requires you to run the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

## 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
```

```
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
```

```
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

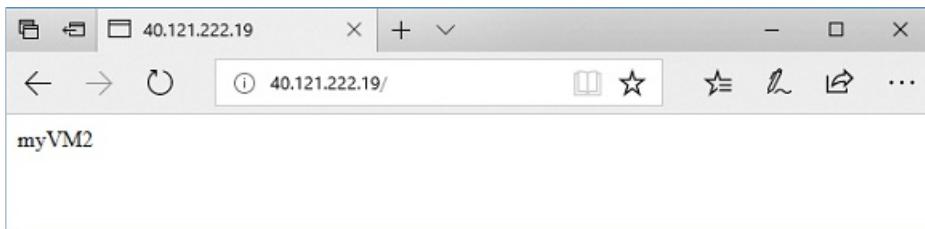
To get the public IP address of the application gateway, use `az network public-ip show`. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://40.121.222.19`,

`http://40.121.222.19:8080/images/test.htm`, or `http://40.121.222.19:8080/video/test.htm`.

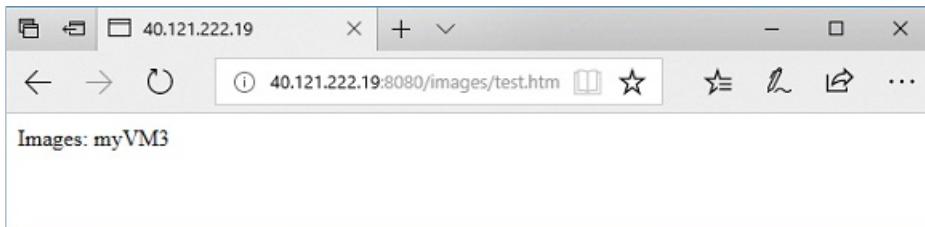
```

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

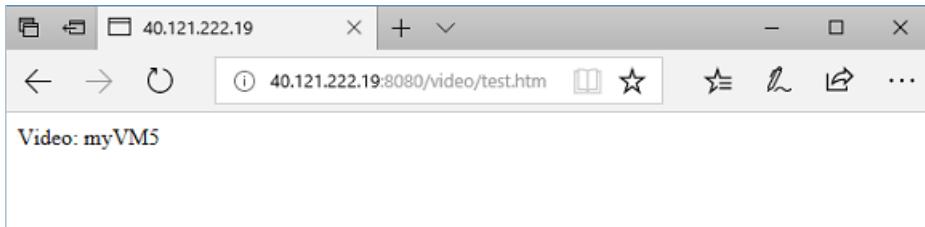
```



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](#)

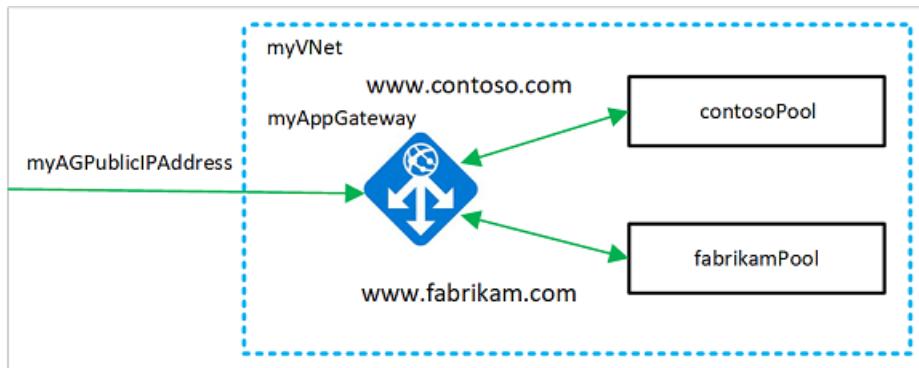
# Create an application gateway that hosts multiple web sites using Azure PowerShell

11/13/2019 • 6 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <p>Select <b>Try It</b> in the upper-right corner of a code block.<br/>           Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.</p> |  |
| <p>Go to <a href="https://shell.azure.com">https://shell.azure.com</a>, or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.</p>   |  |
| <p>Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a>.</p>                                               |  |

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](#).

```
$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
```

## 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
}

```

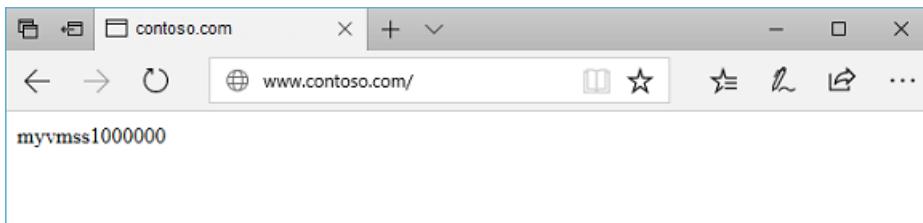
## Create CNAME record in your domain

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.

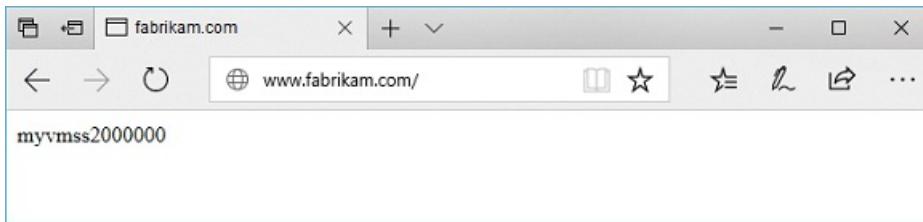
```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```

## 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](#)

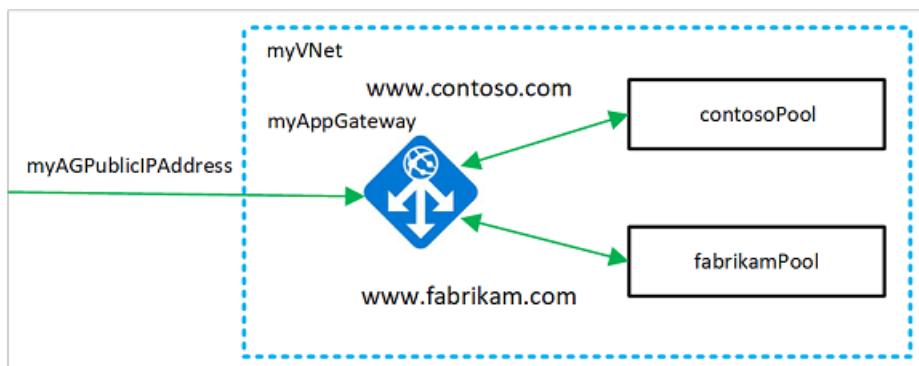
# Create an application gateway that hosts multiple web sites using the Azure CLI

11/13/2019 • 6 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 a [free account](#) before you begin.

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |

| OPTION                                                                                                        | EXAMPLE/LINK |
|---------------------------------------------------------------------------------------------------------------|--------------|
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> . |              |

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 CLI locally, this article requires that you're running the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

## 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 backend listeners

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 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. 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
```

## 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.

```

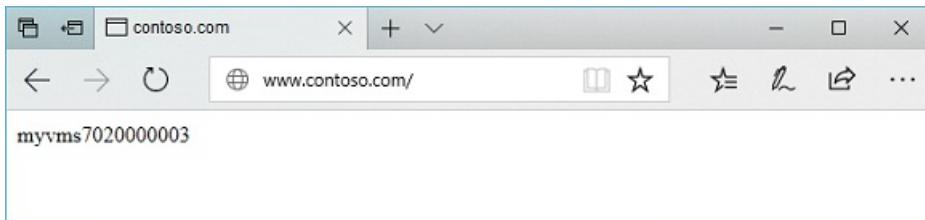
az network public-ip show \
 --resource-group myResourceGroupAG \
 --name myAGPublicIPAddress \
 --query [dnsSettings.fqdn] \
 --output tsv

```

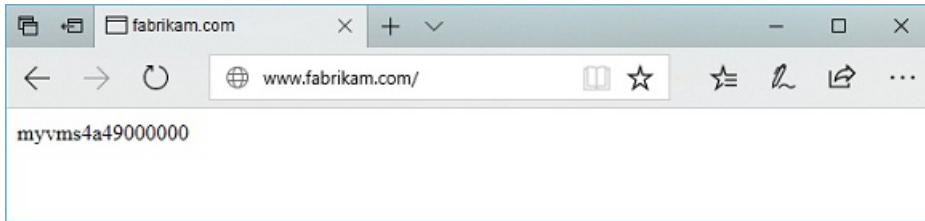
The use of A-records isn't recommended because the VIP may change when the application gateway restarts.

## 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

11/13/2019 • 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.

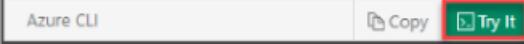
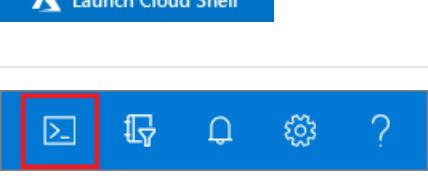
## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |                                                                                      |

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

In this article, you learned how to:

- Set up the network
- Create a listener and redirection rule
- Create an application gateway

# Create an application gateway with external redirection using the Azure CLI

11/12/2019 • 4 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 a [free account](#) before you begin.

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |  |

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 CLI locally, this quickstart requires that you are running the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

## 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](http://www.consoto.org) to the listener for [www.contoso.com](http://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

In this tutorial, you learned how to:

- Set up the network
- Create a listener and redirection rule
- Create an application gateway

# Create an application gateway with HTTP to HTTPS redirection using the Azure portal

1/23/2020 • 6 minutes to read • [Edit Online](#)

You can use the Azure portal to create an [application gateway](#) with a certificate for 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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
```

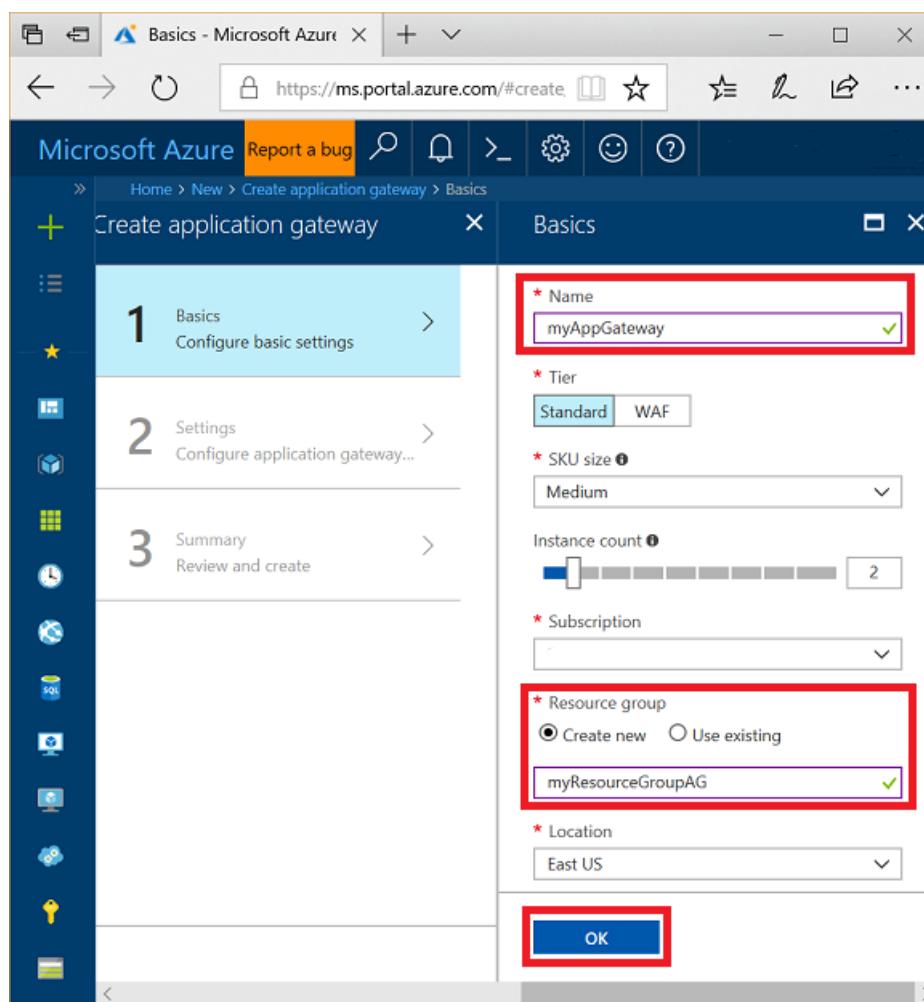
Use the thumbprint to create the pfx file:

```
$pwd = ConvertTo-SecureString -String "Azure123456!" -Force -AsPlainText
Export-PfxCertificate `-
-cer cert:\localMachine\my\E1E81C23B3AD33F9B4D1717B20AB65DBB91AC630 `-
-FilePath c:\appgwcert.pfx `-
-Password $pwd
```

## Create an application gateway

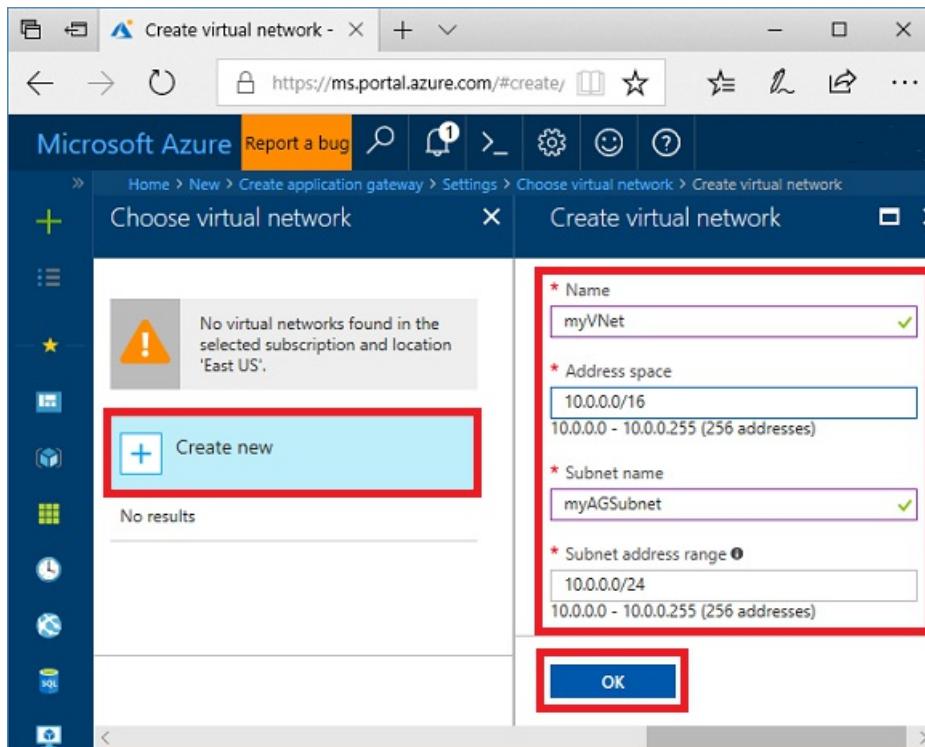
A virtual network is needed for communication between the resources that you create. Two subnets are created in this example: one for the application gateway, and the other for the backend servers. You can create a virtual network at the same time that you create the application gateway.

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**.

The screenshot shows the Microsoft Azure portal interface. The title bar says "Subnets - Microsoft Azur". The address bar shows the URL "https://ms.portal.azure.com/#resource/subscript...". The main content area is titled "myVNet - Subnets" under "Virtual network". On the left, there's a sidebar with various icons and links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Address space, Connected devices, Subnets, DDoS protection). The "Subnets" link is highlighted with a red box. At the top right, there are two buttons: "+ Subnet" and "+ Gateway subnet", also highlighted with red boxes. Below them is a table titled "Search subnets" with one row: "myAGSubnet" (NAME), "10.0.0.0/24" (ADDRESS RANGE), and "251" (AVAILABLE ADDR...).

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

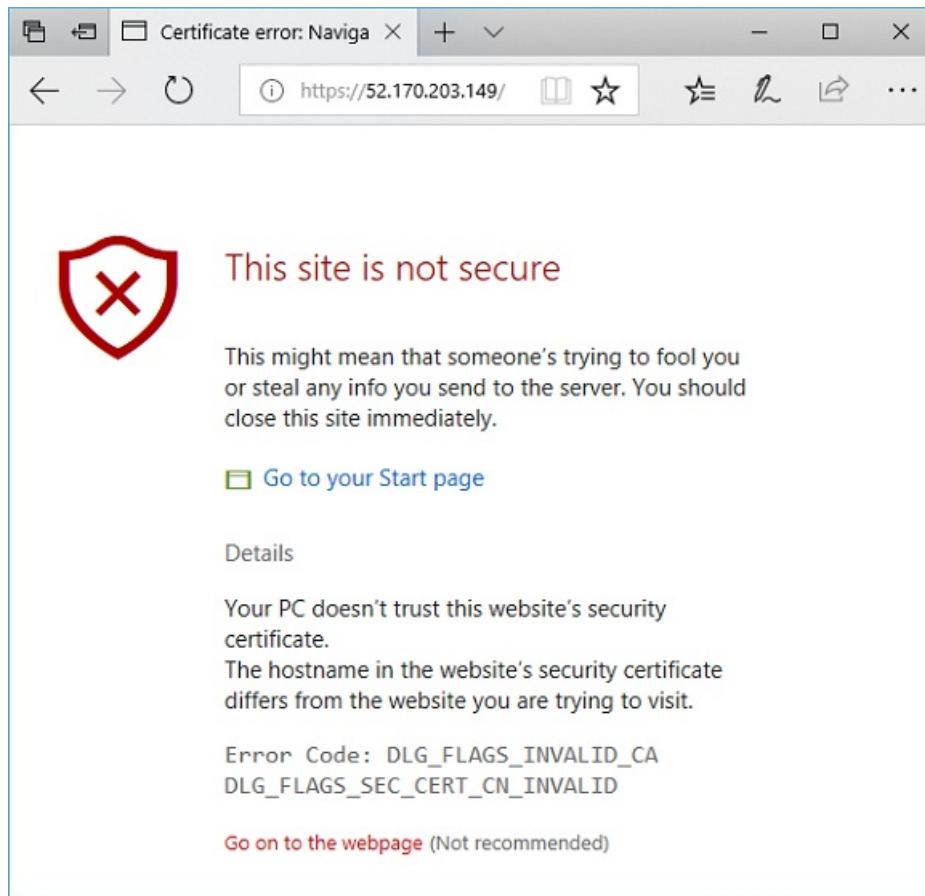
After changing the instances with IIS , you must once again upgrade the scale set with this change.

1. Select the **myvmss** scale set.
2. Under **Settings**, select **Instances**.
3. Select both instances, and then select **Upgrade**.
4. Select **Yes** to confirm.

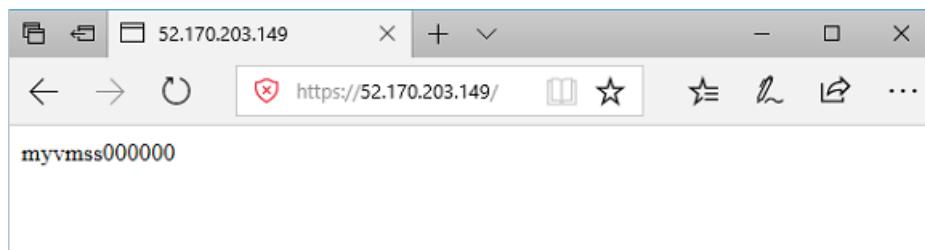
## Test the application gateway

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

2/14/2020 • 6 minutes to read • [Edit Online](#)

You can use the Azure PowerShell to create an [application gateway](#) with a certificate for 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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

```

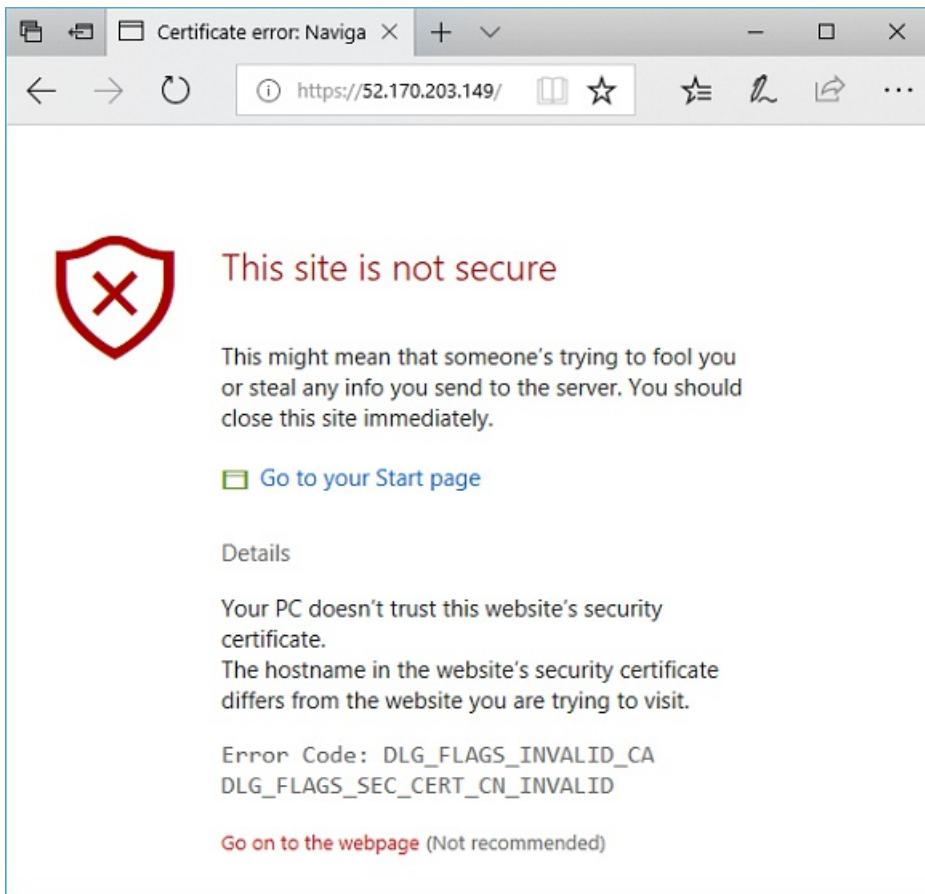
$publicSettings = @{
 "fileUris" = (,"https://raw.githubusercontent.com/Azure/azure-docs-powershell-samples/master/application-gateway/iis/appgatewayurl.ps1");
 "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File appgatewayurl.ps1" }
$vmss = Get-AzVmss -ResourceGroupName myResourceGroupAG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

```

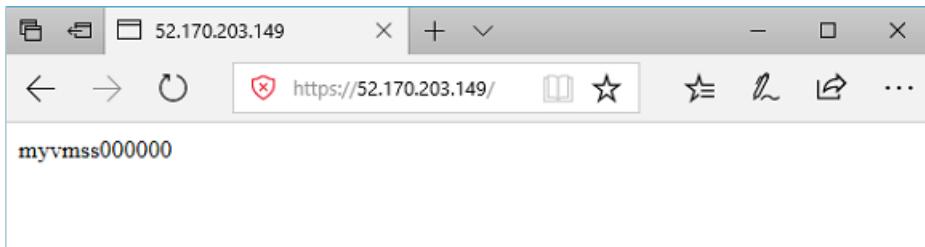
## Test the application gateway

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

In this tutorial, you learned 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

# Create an application gateway with HTTP to HTTPS redirection using the Azure CLI

11/14/2019 • 5 minutes to read • [Edit Online](#)

You can use the Azure CLI to create an [application gateway](#) with a certificate for 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.

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |  |

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 CLI locally, this quickstart requires that you are running the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

## 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
```

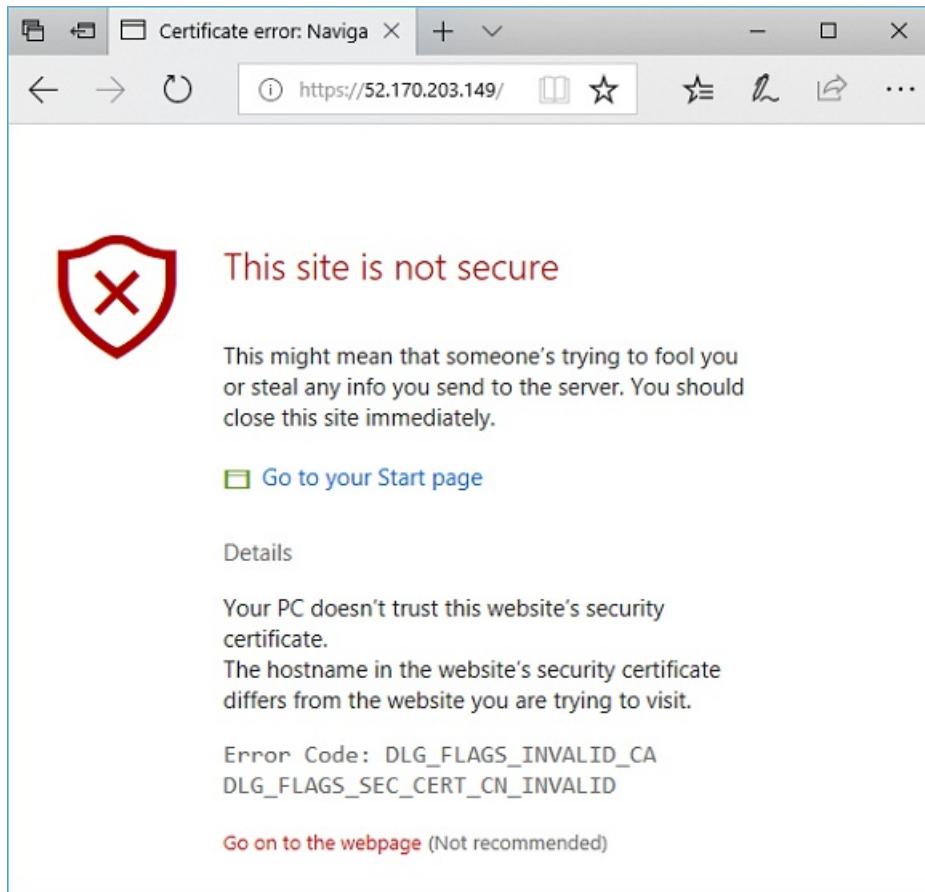
## Install NGINX

```
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" }'
```

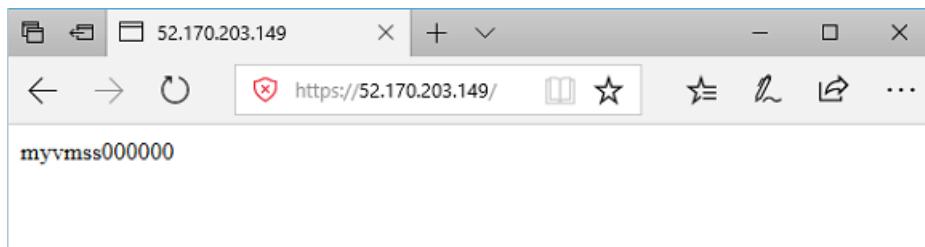
## Test the application gateway

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

In this tutorial, you learned 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

# Create an application gateway with internal redirection using Azure PowerShell

11/13/2019 • 7 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 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 a [free account](#) before you begin.

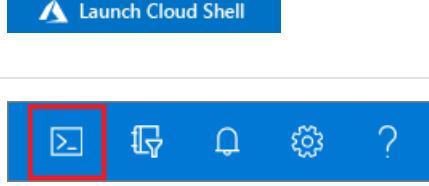
## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |                                                                                      |

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 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 tutorial, you create two listeners for your two domains. In this example, listeners are created for the domains of [www.contoso.com](http://www.contoso.com) and [www.contoso.org](http://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 -VMSScaleSetName 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

```

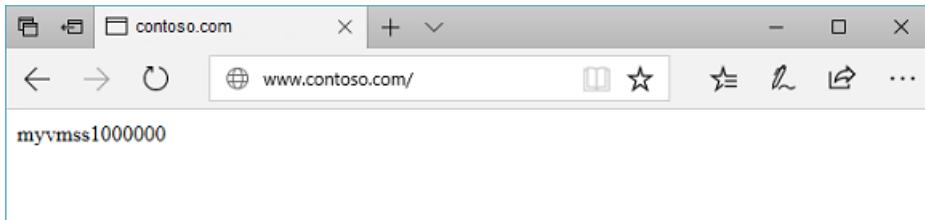
## Create CNAME record in your domain

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](https://www.contoso.com).

## Next steps

In this article, 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 pools
- Create a CNAME record in your domain

# Create an application gateway with internal redirection using the Azure CLI

11/13/2019 • 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:

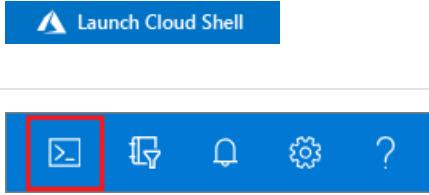
- Set up the network
- Create an application gateway
- Add listeners and redirection rule
- Create a virtual machine scale set with the backend pool
- Create a CNAME record in your domain

If you don't have an Azure subscription, create a [free account](#) before you begin.

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.               |  |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .                                             |                                                                                      |

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 CLI locally, this quickstart requires that you are running the Azure CLI version 2.0.4 or later. To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

## 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\" }'
```

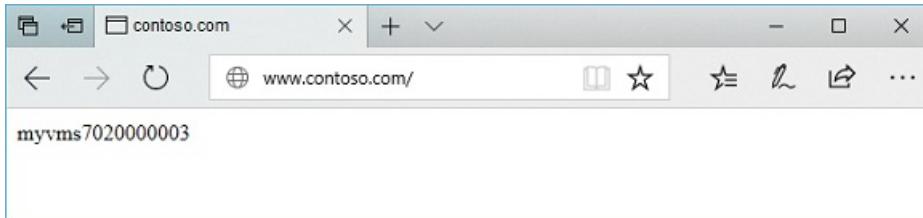
## Create CNAME record in your domain

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](http://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

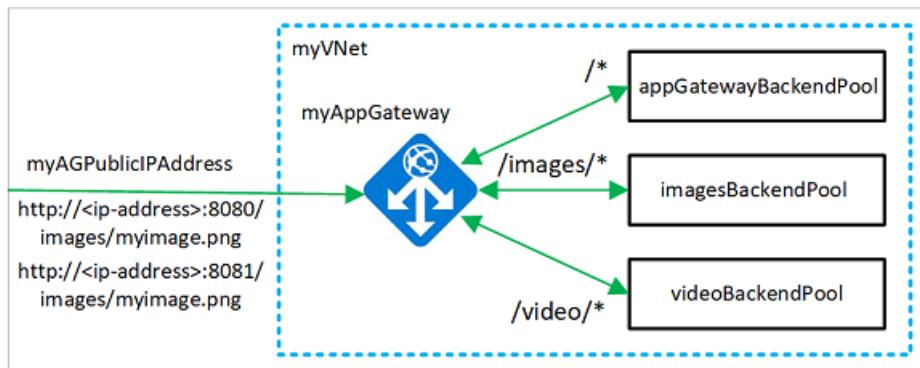
11/13/2019 • 9 minutes to read • [Edit Online](#)

You can use Azure PowerShell to configure [URL-based routing rules](#) when you create an [application gateway](#). In this 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 CLI](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <p>Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.</p>              |  |
| <p>Go to <a href="https://shell.azure.com">https://shell.azure.com</a>, or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.</p> |  |
| <p>Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a>.</p>                                             |  |

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 `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 tutorial, 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.

```

$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 $vmssConfig `
 -AdminUsername azureuser `
 -AdminPassword "Azure123456!" `
 -ComputerNamePrefix myvmss$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 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
}

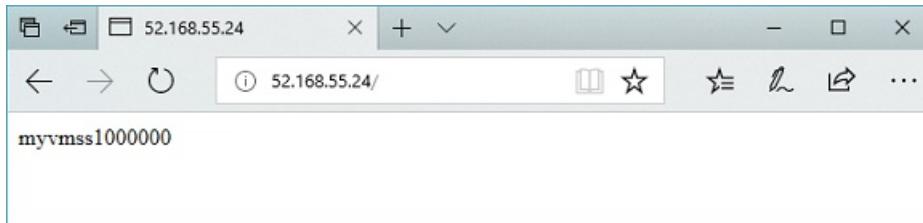
```

## Test the application gateway

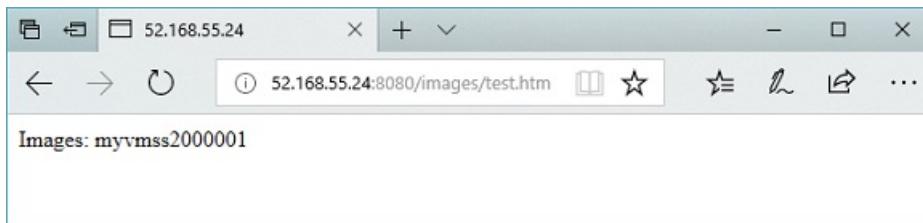
You can use [Get-AzPublicIPAddress](#) to get the public IP address of the application gateway. Copy the public IP address, and then paste it into the address bar of your browser. Such as, `http://52.168.55.24`,

`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`.

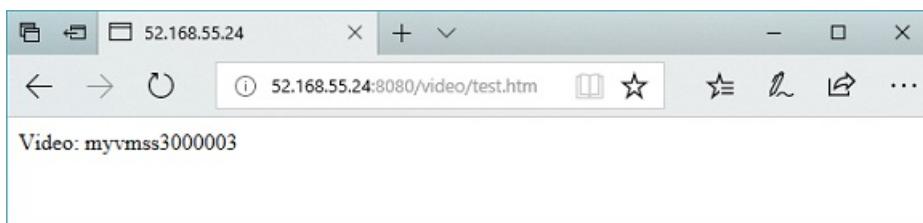
```
Get-AzPublicIPAddress -ResourceGroupName myResourceGroupAG -Name myAGPublicIPAddress
```



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

11/12/2019 • 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**:

The screenshot shows the 'TestHeaderRewrite - Rewrites' page in the Azure portal. On the left sidebar, under 'Settings', the 'Rewrites' option is selected and highlighted with a red box. At the top center, there is a button labeled '+ Rewrite set' also highlighted with a red box. The main area displays a table with three columns: 'REWRITE SETS', 'REWRITES', and 'RULES APPLIED'. A placeholder message 'Start with creating a rewrite set' is visible in the 'REWRITE SETS' column.

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**.

Home > TestHeaderRewrite- Rewrites > Create rewrite set

## Create rewrite set

**① Name and Association**    **② Rewrite rule configuration**

To rewrite HTTP(S) headers, you need to create rewrite sets and associate them with routing rules. On this tab, you can provide the name to the rewrite set and associate it with the routing rules in your application gateway. On the next tab, you can configure the rewrite set by adding one or more rewrite rules to it. [Learn more about rewrite sets.](#)

\* Name: RewriteLocationHeader

**Associated routing rules:**

Select the routing rules to associate to this rewrite set. You can't select routing rules that already have an associated rewrite set.

| ROUTING RULES   PATHS | TYPE                                      |
|-----------------------|-------------------------------------------|
| rule1                 | Basic rule                                |
| PathRule1             | Path-based rule (Default rewrite setting) |
| api                   | Path-based rule                           |
| rule2                 | Basic rule                                |
| demorule              | Basic rule                                |
| rule3                 | Basic rule                                |

Previous    **Next**

5. Create a rewrite rule:

- Select **Add rewrite rule**.

Home > TestHeaderRewrite- Rewrites > Create rewrite set

## Create rewrite set

**① Name and Association**    **② Rewrite rule configuration**

**+ Add rewrite rule**

**REWRITE RULES (RULE SEQUENCE)**

Click add rewrite rule button.

**+ Add condition**    **+ Add action**    **>Delete rewrite rule**

\* Rewrite rule name: Rewrite rule name

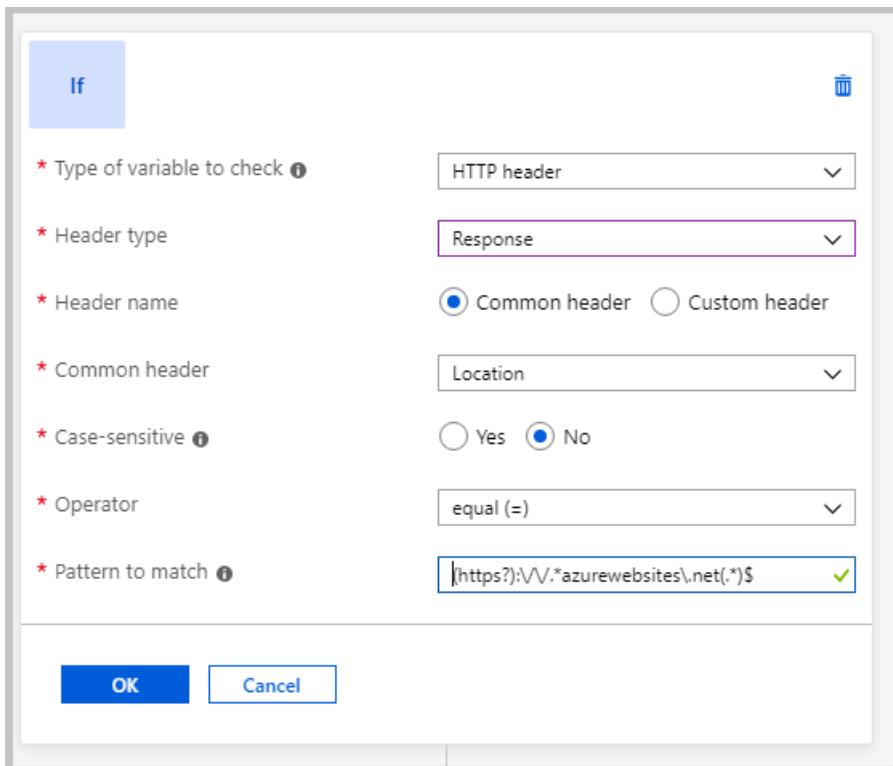
\* Rule sequence: 100

- 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:

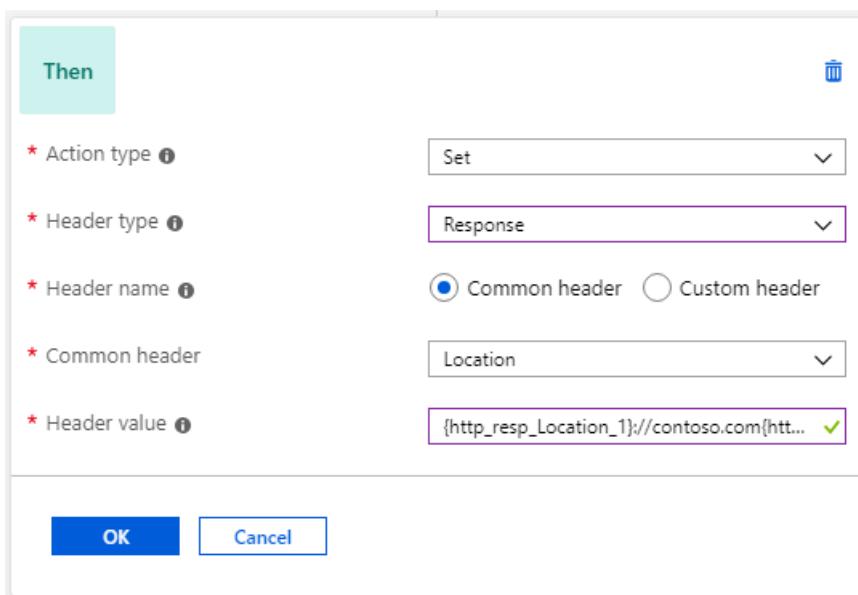
- Select **Add condition** and then select the box containing the **If** instructions to expand it.

- 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:

The screenshot shows the 'Create rewrite set' dialog in the Azure portal. The 'Rewrite rule configuration' tab is active. A new rule named 'LocationHeader' is being created with a sequence of 50. The 'Then' section is configured to set the 'Location' header to the value of the 'http\_resp\_Location\_1' response header. The 'OK' button is highlighted.

9. The Rewrite set view will open. Verify that the rewrite set you created is in the list of rewrite sets:

| REWRITE SETS          | REWRITES | RULES APPLIED |
|-----------------------|----------|---------------|
| LocationHeaderRewrite | 1        | ▶ 1           |
| sai_set-1             | 1        | ▶ 2           |

## 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

4/30/2019 • 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 -RequestHeaderConfiguration
$requestHeaderConfiguration -ResponseHeaderConfiguration $responseHeaderConfiguration
$condition = New-AzApplicationGatewayRewriteRuleCondition -Variable "http_resp_Location" -Pattern "
(https?://\/*azurewebsites\.net(.*)$)" -IgnoreCase
$rewriteRule = New-AzApplicationGatewayRewriteRule -Name LocationHeader -ActionSet $actionSet
$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

```
Add-AzApplicationGatewayRewriteRuleSet -ApplicationGateway $appgw -Name LocationHeaderRewrite -RewriteRule
$rewriteRuleSet.RewriteRules
Set-AzApplicationGatewayRequestRoutingRule -ApplicationGateway $appgw -Name rule1 -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.RewriteRuleSets= $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

11/18/2019 • 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](#)

# Configure App Service with Application Gateway

11/13/2019 • 5 minutes to read • [Edit Online](#)

Since app service is a multi-tenant service instead of a dedicated deployment, it uses host header in the incoming request to resolve the request to the correct app service endpoint. Usually, the DNS name of the application, which in turn is the DNS name associated with the application gateway fronting the app service, is different from the domain name of the backend app service. Therefore, the host header in the original request received by the application gateway is not the same as the host name of the backend service. Because of this, unless the host header in the request from the application gateway to the backend is changed to the host name of the backend service, the multi-tenant backends are not able to resolve the request to the correct endpoint.

Application Gateway provides a switch called `Pick host name from backend address` which overrides the host header in the request with the host name of the back-end when the request is routed from the Application Gateway to the backend. This capability enables support for multi-tenant back ends such as Azure app service and API management.

In this article, you learn how to:

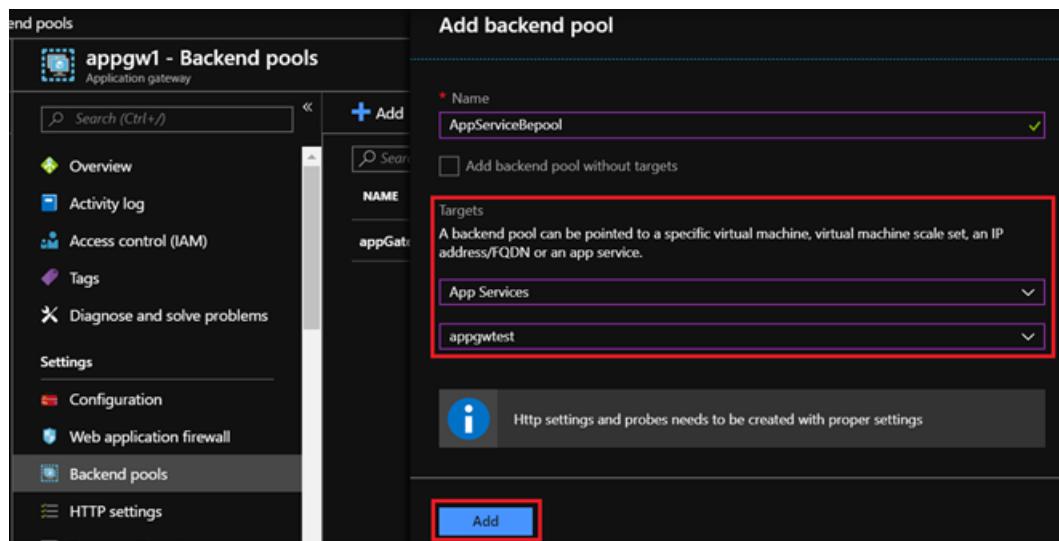
- Create a backend pool and add an App Service to it
- Create HTTP Settings and Custom Probe with "Pick Hostname" switches enabled

## Prerequisites

- Application gateway: If you don't have an existing application gateway, see how to [create an application gateway](#).
- App service: If you don't have an existing App service, see [App service documentation](#).

## Add App service as backend pool

1. In the Azure portal, open the configuration view of your application gateway.
2. Under **Backend pools**, click on **Add** to create a new backend pool.
3. Provide a suitable name to the backend pool.
4. Under **Targets**, click on the dropdown and choose **App Services** as the option.
5. A dropdown immediately below the **Targets** dropdown will appear which will contain a list of your App Services. From this dropdown, choose the App Service you want to add as a backend pool member and click **Add**.



#### NOTE

The dropdown will only populate 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.

## Create HTTP settings for App service

- Under **HTTP Settings**, click **Add** to create a new HTTP Setting.
- Input a name for the HTTP Setting and you can enable or disable Cookie Based Affinity as per your requirement.
- Choose the protocol as HTTP or HTTPS as per your use case.

#### NOTE

If you select HTTPS, you do not need to upload any authentication certificate or trusted root certificate to whitelist the app service backend since app service is a trusted Azure service.

- Check the box for **Use for App Service**. Note that the switches

`Create a probe with pick host name from backend address` and `Pick host name from backend address` will automatically get enabled. `Pick host name from backend address` will override the host header in the request with the host name of the back-end when the request is routed from the Application Gateway to the backend.

`Create a probe with pick host name from backend address` will automatically create a health probe and associate it to this HTTP Setting. You do not need to create any other health probe for this HTTP setting. You can check that a new probe with the name has been added in the list of Health probes and it already has the switch `Pick host name from backend http settings enabled`.

If you already have one or more HTTP Settings which are being used for App service and if those HTTP settings use the same protocol as the one you are using in the one you are creating, then instead of the `Create a probe with pick host name from backend address` switch, you will get a dropdown to select one of the custom probes. This is because since there already exists an HTTP Setting with app service, therefore, there would also exist a health probe which has the switch

`Pick host name from backend http settings enabled`. Choose that custom probe from the dropdown.

5. Click **OK** to create the HTTP setting.

Home > appgw1 - HTTP settings > Add HTTP setting

## Add HTTP setting

appgw1

\* Name  ✓

\* Cookie based affinity  Enabled

\* Connection draining ⓘ  Enabled

\* Protocol  HTTPS

\* Port ⓘ

\* Request timeout

Override backend path

Use for App service

Create a probe with pick host name from backend address

Pick host name from backend address

**OK**

Home > appgw1 - Health probes > AppServiceHTTPSetting1e79f5fd-341b-465e-920a-9f93e5c8f007

## AppServiceHTTPSetting1e79f5fd-341b-465e-920a-9f93e5c8f007

appgw1

Save Discard Delete

Name  
AppServiceHTTPSetting1e79f5fd-341b-465e-920a-9f93e5c8f007

\* Protocol  
HTTP HTTPS

Pick host name from backend http settings

\* Path  
/

\* Interval (seconds)  
30

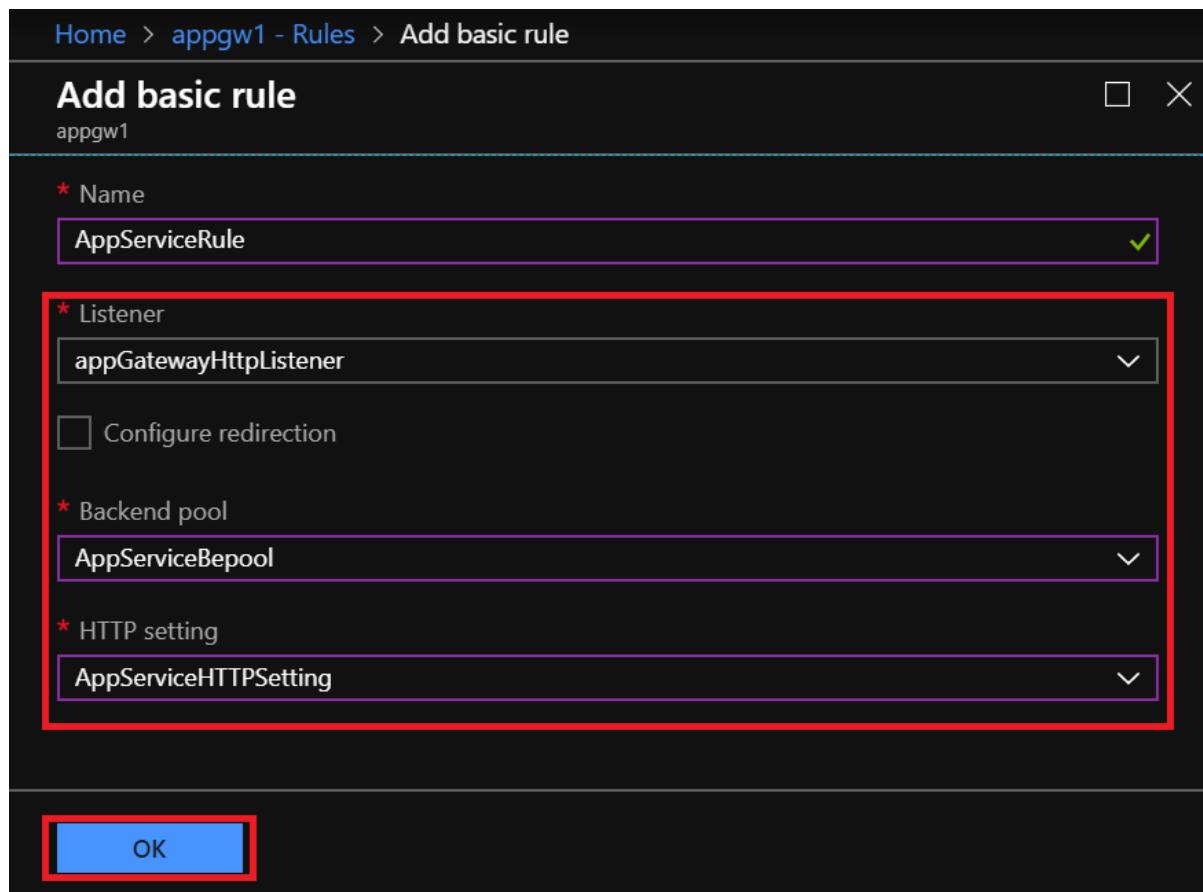
\* Timeout (seconds)  
30

\* Unhealthy threshold  
3

Use probe matching conditions ⓘ

## Create Rule to tie the Listener, Backend Pool and HTTP Setting

1. Under **Rules**, click **Basic** to create a new Basic rule.
2. Provide a suitable name and select the listener which will be accepting the incoming requests for the App service.
3. In the **Backend pool** dropdown, choose the backend pool you created above.
4. In the **HTTP setting** dropdown, choose the HTTP setting you created above.
5. Click **OK** to save this rule.



## Additional configuration in case of redirection to app service's relative path

When the app service sends a redirection response to the client to redirect to its relative path (For example, a redirect from contoso.azurewebsites.net/path1 to contoso.azurewebsites.net/path2), it uses the same hostname in the location header of its response as the one in the request it received 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.

If in your use case, there are scenarios where the App service will need to send a redirection response to the client, perform the [additional steps to rewrite the location header](#).

## 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.

One way you can restrict access to your web apps is to 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.

## Next steps

To learn more about the App service and other multi-tenant support with application gateway, see [multi-tenant service support with application gateway](#).

# Configure App Service with Application Gateway using PowerShell

11/14/2019 • 5 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 first example shows you how to configure an existing application gateway to use a web app as a back-end pool member. The second example shows you how to create a new application gateway with a web app as a back-end pool member.

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Configure a web app behind an existing application gateway

The following example adds a web app as a back-end pool member to an existing application gateway. Both the switch `-PickHostNameFromBackendHttpSettings` on the Probe configuration and `-PickHostNameFromBackendAddress` on the back-end http settings must be provided in order for web apps to work.

```
FQDN of the web app
$webappFQDN = "<enter your webapp FQDN i.e mywebsite.azurewebsites.net>"

Retrieve the resource group
$rg = Get-AzResourceGroup -Name 'your resource group name'

Retrieve an existing application gateway
$gw = Get-AzApplicationGateway -Name 'your application gateway name' -ResourceGroupName $rg.ResourceGroupName

Define the status codes to match for the probe
$match=New-AzApplicationGatewayProbeHealthResponseMatch -StatusCode 200-399

Add a new probe to the application gateway
Add-AzApplicationGatewayProbeConfig -name webappprobe2 -ApplicationGateway $gw -Protocol Http -Path / -Interval 30 -Timeout 120 -UnhealthyThreshold 3 -PickHostNameFromBackendHttpSettings -Match $match

Retrieve the newly added probe
$probe = Get-AzApplicationGatewayProbeConfig -name webappprobe2 -ApplicationGateway $gw

Configure an existing backend http settings
Set-AzApplicationGatewayBackendHttpSettings -Name appGatewayBackendHttpSettings -ApplicationGateway $gw -PickHostNameFromBackendAddress -Port 80 -Protocol http -CookieBasedAffinity Disabled -RequestTimeout 30 -Probe $probe

Add the web app to the backend pool
Set-AzApplicationGatewayBackendAddressPool -Name appGatewayBackendPool -ApplicationGateway $gw -BackendFqdns $webappFQDN

Update the application gateway
Set-AzApplicationGateway -ApplicationGateway $gw
```

# Configure a web application behind a new application gateway

This scenario deploys a web app with the asp.net getting started website and an application gateway.

```
Defines a variable for a dotnet get started web app repository location
$gitrepo="https://github.com/Azure-Samples/app-service-web-dotnet-get-started.git"

Unique web app name
$webappname="mywebapp$(Get-Random)"

Creates a resource group
$rg = New-AzResourceGroup -Name ContosoRG -Location Eastus

Create an App Service plan in Free tier.
New-AzAppServicePlan -Name $webappname -Location EastUs -ResourceGroupName $rg.ResourceGroupName -Tier Free

Creates a web app
$webapp = New-AzWebApp -ResourceGroupName $rg.ResourceGroupName -Name $webappname -Location EastUs -AppServicePlan $webappname

Configure GitHub deployment from your GitHub repo and deploy once to web app.
$PropertiesObject = @{
 repoUrl = "$gitrepo";
 branch = "master";
 isManualIntegration = "true";
}
Set-AzResource -PropertyObject $PropertiesObject -ResourceGroupName $rg.ResourceGroupName -ResourceType Microsoft.Web/sites/sourcecontrols -ResourceName $webappname/web -ApiVersion 2015-08-01 -Force

Creates a subnet for the application gateway
$subnet = New-AzVirtualNetworkSubnetConfig -Name subnet01 -AddressPrefix 10.0.0.0/24

Creates a vnet for the application gateway
$vnet = New-AzVirtualNetwork -Name appgwvnet -ResourceGroupName $rg.ResourceGroupName -Location EastUs -AddressPrefix 10.0.0.0/16 -Subnet $subnet

Retrieve the subnet object for use later
$subnet=$vnet.Subnets[0]

Create a public IP address
$publicip = New-AzPublicIpAddress -ResourceGroupName $rg.ResourceGroupName -name publicIP01 -location EastUs -AllocationMethod Dynamic

Create a new IP configuration
$gipconfig = New-AzApplicationGatewayIPConfiguration -Name gatewayIP01 -Subnet $subnet

Create a backend pool with the hostname of the web app
$pool = New-AzApplicationGatewayBackendAddressPool -Name appGatewayBackendPool -BackendFqdns $webapp.HostNames

Define the status codes to match for the probe
$match = New-AzApplicationGatewayProbeHealthResponseMatch -StatusCode 200-399

Create a probe with the PickHostNameFromBackendHttpSettings switch for web apps
$probeconfig = New-AzApplicationGatewayProbeConfig -name webappprobe -Protocol Http -Path / -Interval 30 -Timeout 120 -UnhealthyThreshold 3 -PickHostNameFromBackendHttpSettings -Match $match

Define the backend http settings
$poolSetting = New-AzApplicationGatewayBackendHttpSettings -Name appGatewayBackendHttpSettings -Port 80 -Protocol Http -CookieBasedAffinity Disabled -RequestTimeout 120 -PickHostNameFromBackendAddress -Probe $probeconfig

Create a new front-end port
$fp = New-AzApplicationGatewayFrontendPort -Name frontendport01 -Port 80

Create a new front end IP configuration
$fipconfig = New-AzApplicationGatewayFrontendIPConfig -Name fipconfig01 -PublicIPAddress $publicip
```

```

Create a new listener using the front-end ip configuration and port created earlier
$listener = New-AzApplicationGatewayHttpListener -Name listener01 -Protocol Http -FrontendIPConfiguration
$fipconfig -FrontendPort $fp

Create a new rule
$rule = New-AzApplicationGatewayRequestRoutingRule -Name rule01 -RuleType Basic -BackendHttpSettings
$poolSetting -HttpListener $listener -BackendAddressPool $pool

Define the application gateway SKU to use
$sku = New-AzApplicationGatewaySku -Name Standard_Small -Tier Standard -Capacity 2

Create the application gateway
$appgw = New-AzApplicationGateway -Name ContosoAppGateway -ResourceGroupName $rg.ResourceGroupName -Location
EastUs -BackendAddressPools $pool -BackendHttpSettingsCollection $poolSetting -Probes $probeconfig -
FrontendIpConfigurations $fipconfig -GatewayIpConfigurations $gipconfig -FrontendPorts $fp -HttpListeners
$listener -RequestRoutingRules $rule -Sku $sku

```

## 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. To create the alias, retrieve the details of the application gateway and its associated IP/DNS name using the PublicIPAddress element attached to the application gateway. This can be done with Azure DNS or other DNS providers, by creating a CNAME record that points to the [public IP address](#). The use of A-records is not recommended since the VIP may change on restart of application gateway.

```
Get-AzPublicIpAddress -ResourceGroupName ContosoRG -Name publicIP01
```

```

Name : publicIP01
ResourceGroupName : ContosoRG
Location : eastus
Id :
/subscriptions/<subscription_id>/resourceGroups/ContosoRG/providers/Microsoft.Network/publicIPAddresses/publicI
P01
Etag : W/"00000d5b-54ed-4907-bae8-99bd5766d0e5"
ResourceGuid : 00000000-0000-0000-0000-000000000000
ProvisioningState : Succeeded
Tags :
PublicIpAllocationMethod : Dynamic
IpAddress : xx.xx.xxxx.xx
PublicIpAddressVersion : IPv4
IdleTimeoutInMinutes : 4
IpConfiguration : {
 "Id": "
/subscriptions/<subscription_id>/resourceGroups/ContosoRG/providers/Microsoft.Network/applicationGateways/Cont
osoAppGateway/frontendIP
 Configurations/foreground1"
}
DnsSettings : {
 "Fqdn": "00000000-0000-xxxx-xxxx-xxxxxxxxxxxx.cloudapp.net"
}

```

## 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.

One way you can restrict access to your web apps is to 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.

## Next steps

Learn how to configure redirection by visiting: [Configure redirection on Application Gateway with PowerShell](#).

# Create a custom probe for Application Gateway by using the portal

11/13/2019 • 7 minutes to read • [Edit Online](#)

In this article, you add a custom health probe to an existing application gateway through the Azure portal. Using the health probes, Azure Application Gateway monitors the health of the resources in the back-end pool.

## Before you begin

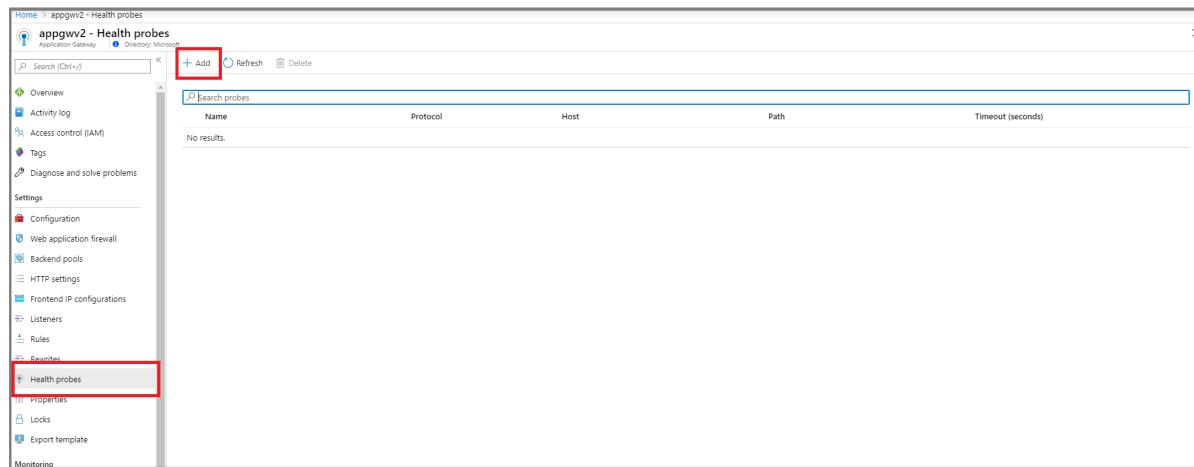
If you do not already have an application gateway, visit [Create an Application Gateway](#) to create an application gateway to work with.

## 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.



The screenshot shows the Azure portal interface for managing an Application Gateway named 'appgwv2'. On the left, there's a navigation sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration, Backend pools, HTTP settings, Frontend IP configurations, Listeners, Rules, Requests, Properties, Locks, Export template, and Monitoring. Under the 'Requests' section, 'Health probes' is highlighted with a red box. At the top center, there's a search bar labeled 'Search probes' and a table header with columns: Name, Protocol, Host, Path, and Timeout (seconds). Below the table, it says 'No results.'

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. |

| Setting                                          | Value             | Details                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Protocol</b>                                  | HTTP or HTTPS     | The protocol that the health probe uses.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Host</b>                                      | 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'.                                                                                                                       |
| <b>Pick host name from backend HTTP settings</b> | Yes or No         | Sets the <i>host</i> 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 to. Specially required in case of multi-tenant backends such as Azure app service. <a href="#">Learn more</a>                                                                                                                                                                                          |
| <b>Path</b>                                      | / or another 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 '/'                                                                                                                                                                                                                                                                                                                                         |
| <b>Interval (secs)</b>                           | 30                | How often the probe is run to check for health. It is not recommended to set the lower than 30 seconds.                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Timeout (secs)</b>                            | 30                | The amount of time the probe waits before timing out. If a valid response is not 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. Note that the timeout value should not 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. |
| <b>Unhealthy threshold</b>                       | 3                 | Number of consecutive failed attempts to be considered unhealthy. The threshold can be set to 1 or more.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Use probe matching conditions</b>             | 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. <a href="#">Learn more</a>                                                                                                                                                                                                                                                                     |

| SETTING              | VALUE                   | DETAILS                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>HTTP Settings</b> | selection from dropdown | <p>Probe will get associated with the HTTP setting(s) 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 setting(s) which are not associated with any other custom probe.</p> <p>Note that only those HTTP setting(s) are available for association which have the same protocol as the protocol chosen in this probe configuration and have the same state for the <i>Pick Host Name From Backend HTTP setting</i> switch.</p> |

### IMPORTANT

The probe will monitor health of the backend only when it is associated with one or more HTTP Setting(s). It will monitor back-end resources of those back-end pools which are associated to the HTTP setting(s) to which this probe is associated with. The probe request will be sent to `http://(host name):(port from httpsetting)/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 Setting(s) used for this probe.

The screenshot shows the Azure portal interface for managing Application Gateways. On the left, the navigation menu includes sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Configuration, Web application firewall, Backend pools, HTTP settings, Frontend IP configurations, Listeners, Rules, Rewrites), Health probes (selected), Properties, Locks, Export template, Monitoring (Alerts, Metrics, Diagnostic settings, Logs, Backend health, Connection troubleshoot), and Support + troubleshooting (Resource health, New support request).

The main content area displays the 'Add health probe' dialog for the 'appgw2 - Health probes' application gateway. The dialog fields are as follows:

- Name \***: customProbe
- Protocol \***: HTTP (selected)
- Host \***: 127.0.0.1
- Pick host name from backend HTTP settings**: Yes (selected)
- Path \* (radio button)**: / (selected)
- Interval (seconds) \***: 30
- Timeout (seconds) \***: 30
- Unhealthy threshold \***: 3
- Use probe matching conditions**: No (selected)
- HTTP settings (dropdown)**: 2 selected

At the bottom of the dialog, there is a checkbox:  I want to test the backend health before adding the health probe. Below the checkbox are two buttons: **Test** (highlighted with a red box) and **Cancel**.

- 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 is not recommended. This is because the Application Gateway removes those backend resources 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 502 error.

| Backend pool          | HTTP setting                  | Status    | Details                                                                                                                                                                                                                              |
|-----------------------|-------------------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| appGatewayBackendPool | appGatewayBackendHttpSettings | Healthy   | Success. Received 200 status code                                                                                                                                                                                                    |
| testappserv           | testIhttp                     | Unhealthy | Received invalid status code: 404 in the backend's HTTP response. Per the health probe configuration, 200-399 is the acceptable status code. Either modify probe configuration or resolve backend issues. <a href="#">Learn more</a> |

- 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

- Sign in to the [Azure portal](#). If you don't already have an account, you can sign up for a [free one-month trial](#)
- 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.
- 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                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>                                      | customProbe       | This value is a friendly name given to the probe that is accessible in the portal.                                                                                                                                                                                                                                                                          |
| <b>Protocol</b>                                  | HTTP or HTTPS     | The protocol that the health probe uses.                                                                                                                                                                                                                                                                                                                    |
| <b>Host</b>                                      | 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'. |
| <b>Pick host name from backend HTTP settings</b> | Yes or No         | Sets the <i>host</i> 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 to. Specially required in case of multi-tenant backends such as Azure app service. <a href="#">Learn more</a>                                                                    |
| <b>Path</b>                                      | / or another 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 '/'                                                                                                                                                                                                                   |

| SETTING                              | VALUE     | DETAILS                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Interval (secs)</b>               | 30        | How often the probe is run to check for health. It is not recommended to set the lower than 30 seconds.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Timeout (secs)</b>                | 30        | The amount of time the probe waits before timing out. If a valid response is not 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. Note that the time-out value should not 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. |
| <b>Unhealthy threshold</b>           | 3         | Number of consecutive failed attempts to be considered unhealthy. The threshold can be set to 1 or more.                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Use probe matching conditions</b> | 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. <a href="#">Learn more</a>                                                                                                                                                                                                                                                                      |

#### IMPORTANT

The host name is not the same as server name. This value is the name of the virtual host running on the application server. The probe is sent to `http://(host name):(port from httpsetting)/urlPath`

#### Add probe to the gateway

Now that the probe has been created, it is 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.

| NAME                          | PORT | PROTOCOL | COOKIE BASED AFFINITY | CUSTOM PROBE |
|-------------------------------|------|----------|-----------------------|--------------|
| appGatewayBackendHttpSettings | 80   | HTTP     | Disabled              | -            |

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

1/14/2020 • 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 orHttps is case-sensitive.

A new configuration item <Probe> is added to configure custom probes.

The configuration parameters are:

| PARAMETER                 | DESCRIPTION                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>               | Reference name for custom probe.                                                                                                                                                                                                                                                                        |
| <b>Protocol</b>           | Protocol used (possible values are HTTP or HTTPS).                                                                                                                                                                                                                                                      |
| <b>Host and Path</b>      | 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. |
| <b>Interval</b>           | Configures the probe interval checks in seconds.                                                                                                                                                                                                                                                        |
| <b>Timeout</b>            | Defines the probe time-out for an HTTP response check.                                                                                                                                                                                                                                                  |
| <b>UnhealthyThreshold</b> | The number of failed HTTP responses needed to flag the back-end instance as <i>unhealthy</i> .                                                                                                                                                                                                          |

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 Secure Sockets Layer (SSL) offload, see [Configure an application gateway for SSL 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

11/13/2019 • 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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
<b>Gateway IP configuration</b>	An IP configuration for an application gateway.
<b>Backend pool</b>	A pool of IP addresses, FQDN's, or NICs that are to the application servers that host the web application
<b>Health probe</b>	A custom probe used to monitor the health of the backend pool members
<b>HTTP settings</b>	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
<b>Frontend port</b>	The port that the application gateway listens for traffic on
<b>Listener</b>	A combination of a protocol, frontend IP configuration, and frontend port. This is what listens for incoming requests.

COMPONENT	DESCRIPTION
<b>Rule</b>	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
$getgw = 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 SSL offloading by visiting: [Configure SSL Offload](#)

# Back-end server certificate is not whitelisted for an application gateway using an Internal Load Balancer with an App Service Environment

9/19/2019 • 2 minutes to read • [Edit Online](#)

This article troubleshoots the following issue: A certificate isn't whitelisted 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 SSL 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 – Back-end server certificate is not whitelisted 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 a 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 is not 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 in case of 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 are 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 whitelisting. It won't be used for SSL communication.) You can retrieve the ILB certificate by accessing the ILB with its IP address from your browser on HTTPS then exporting the 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

11/13/2019 • 5 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:

- The app service URL is exposed in the browser when there's a redirection.
- The app service ARRAffinity cookie domain is set to the app service host name, example.azurewebsites.net, instead of the original host.

When a back-end application sends a redirection response, you might want to redirect the client to a different URL than the one specified by the back-end application. You might want to do this when an app service is hosted behind an application gateway and requires the client to do a redirection to its relative path. An example is a redirect from contoso.azurewebsites.net/path1 to contoso.azurewebsites.net/path2.

When the app service sends a redirection response, it uses the same host name in the location header of its response as the one in the request it receives from the application gateway. For example, the client makes the request directly to contoso.azurewebsites.net/path2 instead of going through the application gateway contoso.com/path2. You don't want to bypass the application gateway.

This issue might happen for the following main reasons:

- You have redirection configured on your app service. Redirection can be as simple as adding a trailing slash to the request.
- You have Azure Active Directory authentication, which causes the redirection.

Also, when you use app services behind an application gateway, the domain name associated with the application gateway (example.com) is different from the domain name of the app service (say, example.azurewebsites.net). The domain value for the ARRAffinity cookie set by the app service carries the example.azurewebsites.net domain name, which isn't desirable. The original host name, example.com, should be the domain name value in the cookie.

## Sample configuration

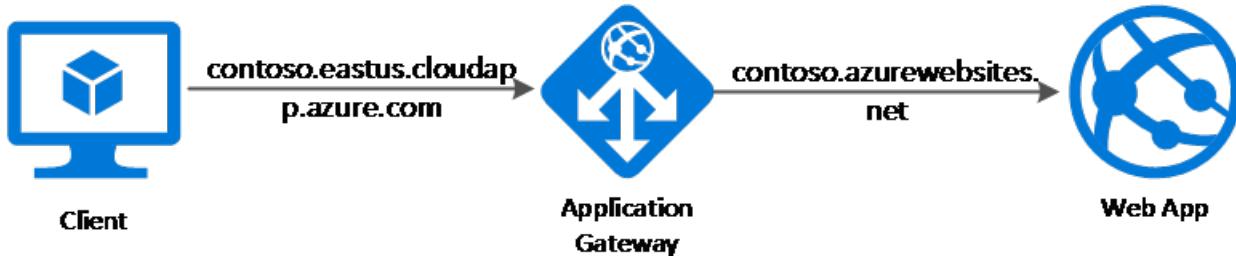
- HTTP listener: Basic or multi-site
- Back-end address pool: App Service
- HTTP settings: **Pick Hostname from Backend Address** enabled
- Probe: **Pick Hostname from HTTP Settings** enabled

## 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 original request from the client has the application gateway's domain name, contoso.com, as the host name. You need to configure the application gateway to change the host name in the original request to the app service's

host name when it routes the request to the app service back end. Use the switch **Pick Hostname from Backend Address** in the application gateway's HTTP setting configuration. Use the switch **Pick Hostname from Backend HTTP Settings** in the health probe configuration.



When the app service does a redirection, it uses the overridden host name `contoso.azurewebsites.net` in the location header instead of the original host name `contoso.com`, unless configured otherwise. Check the following example request and response headers.

```
Request headers to Application Gateway:

Request URL: http://www.contoso.com/path

Request Method: GET

Host: www.contoso.com

Response headers:

Status Code: 301 Moved Permanently

Location: http://contoso.azurewebsites.net/path/

Server: Microsoft-IIS/10.0

Set-Cookie: ARRAffinity=b5b1b14066f35b3e4533a1974cacfb9d96bf1960b6518aa2c2e2619700e4010;Path=/;HttpOnly;Domain=contoso.azurewebsites.net

X-Powered-By: ASP.NET
```

In the previous example, notice that the response header has a status code of 301 for redirection. The location header has the app service's host name instead of the original host name `www.contoso.com`.

## Solution: Rewrite the location header

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. If you use v1 SKU, we recommend that you [migrate from v1 to v2](#). You want to use rewrite and other [advanced capabilities](#) that are available with v2 SKU.

## Alternate solution: Use a custom domain name

If you use v1 SKU, you can't rewrite the location header. This capability is only available for v2 SKU. To resolve the

redirection issue, pass the same host name that the application gateway receives to the app service as well, instead of doing a host override.

The app service now does the redirection (if any) on the same original host header that points to the application gateway and not its own.

You must own a custom domain and follow this process:

- Register the domain to the custom domain list of the app service. You must have a CNAME in your custom domain that points to the app service's FQDN. For more information, see [Map an existing custom DNS name to Azure App Service](#).

HOSTNAMES ASSIGNED TO SITE	SSL BINDING
www.contoso.com	Add binding
cephalin-vscode.azurewebsites.net	...

- Your app service is ready to accept the host name `www.contoso.com`. Change your CNAME entry in DNS to point it back to the application gateway's FQDN, for example, `appgw.eastus.cloudapp.azure.com`.
- Make sure that your domain `www.contoso.com` resolves to the application gateway's FQDN when you do a DNS query.
- Set your custom probe to disable **Pick Hostname from Backend HTTP Settings**. In the Azure portal, clear the check box in the probe settings. In PowerShell, don't use the `-PickHostNameFromBackendHttpSettings` switch in the `Set-AzApplicationGatewayProbeConfig` command. In the host name field of the probe, enter your app service's FQDN, example.azurewebsites.net. The probe requests sent from the application gateway carry this FQDN in the host header.

#### NOTE

For the next step, make sure that your custom probe isn't associated to your back-end HTTP settings. Your HTTP settings still have the **Pick Hostname from Backend Address** switch enabled at this point.

- Set your application gateway's HTTP settings to disable **Pick Hostname from Backend Address**. In the Azure portal, clear the check box. In PowerShell, don't use the `-PickHostNameFromBackendAddress` switch in the `Set-AzApplicationGatewayBackendHttpSettings` command.
- Associate the custom probe back to the back-end HTTP settings, and verify that the back end is healthy.
- The application gateway should now forward the same host name, `www.contoso.com`, to the app service. The redirection happens on the same host name. Check the following example request and response headers.

To implement the previous steps using PowerShell for an existing setup, use the sample PowerShell script that follows. Note how we haven't used the **-PickHostname** switches in the probe and HTTP settings configuration.

```
$gw=Get-AzApplicationGateway -Name AppGw1 -ResourceGroupName AppGwRG
Set-AzApplicationGatewayProbeConfig -ApplicationGateway $gw -Name AppServiceProbe -Protocol Http -HostName "example.azurewebsites.net" -Path "/" -Interval 30 -Timeout 30 -UnhealthyThreshold 3
$probe=Get-AzApplicationGatewayProbeConfig -Name AppServiceProbe -ApplicationGateway $gw
Set-AzApplicationGatewayBackendHttpSettings -Name appgwhttpsettings -ApplicationGateway $gw -Port 80 -Protocol Http -CookieBasedAffinity Disabled -Probe $probe -RequestTimeout 30
Set-AzApplicationGateway -ApplicationGateway $gw
```

```
Request headers to Application Gateway:

Request URL: http://www.contoso.com/path

Request Method: GET

Host: www.contoso.com

Response headers:

Status Code: 301 Moved Permanently

Location: http://www.contoso.com/path/

Server: Microsoft-IIS/10.0

Set-Cookie:
ARRAffinity=b5b1b14066f35b3e4533a1974cacfb9d969bf1960b6518aa2c2e2619700e4010;Path=/;HttpOnly;Domain=www.contoso.com

X-Powered-By: ASP.NET
```

## Next steps

If the preceding steps didn't resolve the issue, open a [support ticket](#).

# Troubleshoot Azure Application Gateway session affinity issues

11/13/2019 • 7 minutes to read • [Edit Online](#)

Learn how to diagnose and resolve session affinity issues with Azure Application Gateway.

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Overview

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.

## Possible problem causes

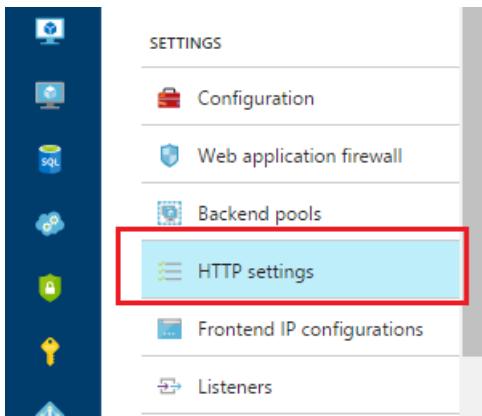
The problem in maintaining cookie-based session affinity may happen due to the following main reasons:

- “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. Click **appGatewayBackendHttpSettings** on the right side to check whether you have selected **Enabled** for Cookie based affinity.

**appGatewayBackendHttpSettings**  
mikeappgateway

Name  
appGatewayBackendHttpSettings

\* Cookie based affinity  
 Enabled  Disabled

\* Request timeout  
30

\* Protocol  
 HTTP  HTTPS

\* Port  
80

Use custom probe ?

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 ARRAffinity details. If you don't find the ARRAffinity details, such as "**ARRAffinity=ARRAffinityValue**" within the cookie set, that means the client is not replying with the ARRA cookie, which is provided by the Application Gateway. For example:

382	303	HTTPS			887	text/html; c...	iexplor...	
383	200	HTTPS			67	application/...	iexplor...	
384	404	HTTPS			148	text/xml; c...	iexplor...	
385	404	HTTPS			148	text/xml; c...	iexplor...	
386	404	HTTPS			148	text/xml; c...	iexplor...	
387	200	HTTPS			263	no-stor...	application/...	
388	303	HTTPS			100	text/html; c...	iexplor...	
389	303	HTTPS			106	text/html; c...	iexplor...	
390	303	HTTPS			111	text/html; c...	iexplor...	
391	200	HTTPS	/en-US/app/launcher/home		1,278	no-stor...	text/html; c...	iexplor...
392	304	HTTPS			0	no-cac...	application/...	iexplor...
393	200	HTTPS			321	no-cac...	application/...	iexplor...
394	304	HTTPS			0	no-cac...	application/...	iexplor...
395	200	HTTPS			442	no-cac...	application/...	iexplor...
396	304	HTTPS			0	no-cac...	application/...	iexplor...
397	304	HTTPS			0	public; ...	application/...	iexplor...
398	304	HTTPS			0	no-cac...	application/...	iexplor...
399	304	HTTPS			0	public; ...	application/...	iexplor...
400	304	HTTPS			0	no-cac...	application/...	iexplor...
401	304	HTTPS			0	no-cac...	application/...	iexplor...
402	200	HTTPS			1,196	no-stor...	application/...	iexplor...
403	200	HTTPS			3,875	no-stor...	application/...	iexplor...
404	200	HTTPS			2,422	no-stor...	application/...	iexplor...
405	200	HTTPS			3,673	no-stor...	application/...	iexplor...
406	200	HTTPS			660	no-stor...	application/...	iexplor...
407	200	HTTPS			4,052	no-stor...	application/...	iexplor...
408	403	HTTPS			89	no-stor...	application/...	iexplor...
409	200	HTTPS			1,913	no-stor...	application/...	iexplor...
410	200	HTTPS			1,195	no-stor...	application/...	iexplor...
411	200	HTTPS			1,151	no-stor...	application/...	iexplor...
412	200	HTTPS			2,421	no-stor...	application/...	iexplor...
413	403	HTTPS			155	no-stor...	application/...	iexplor...
414	303	HTTPS			887	text/html; c...	iexplor...	
415	200	HTTPS			67	application/...	iexplor...	
416	404	HTTPS			148	text/xml; c...	iexplor...	

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 through the Azure portal

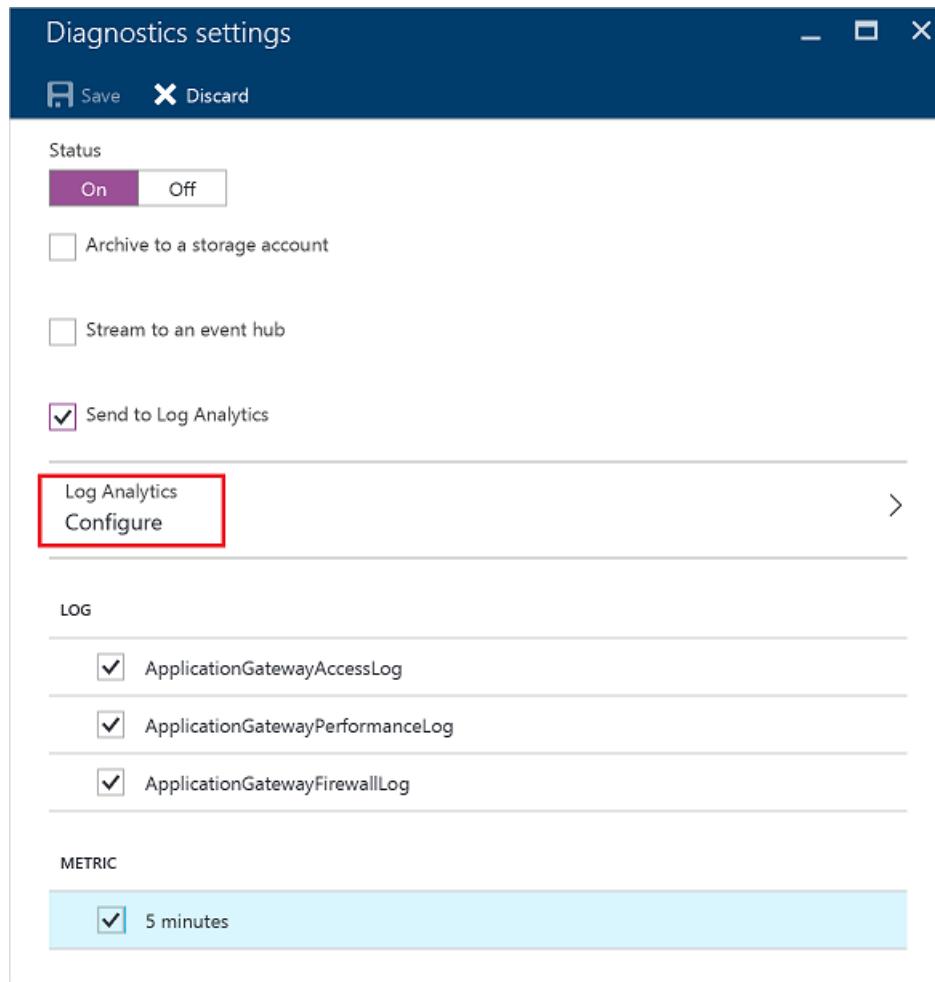
1. In the [Azure portal](#), find your resource and then click **Diagnostic logs**.

For Application Gateway, three logs are available: Access log, Performance log, Firewall log

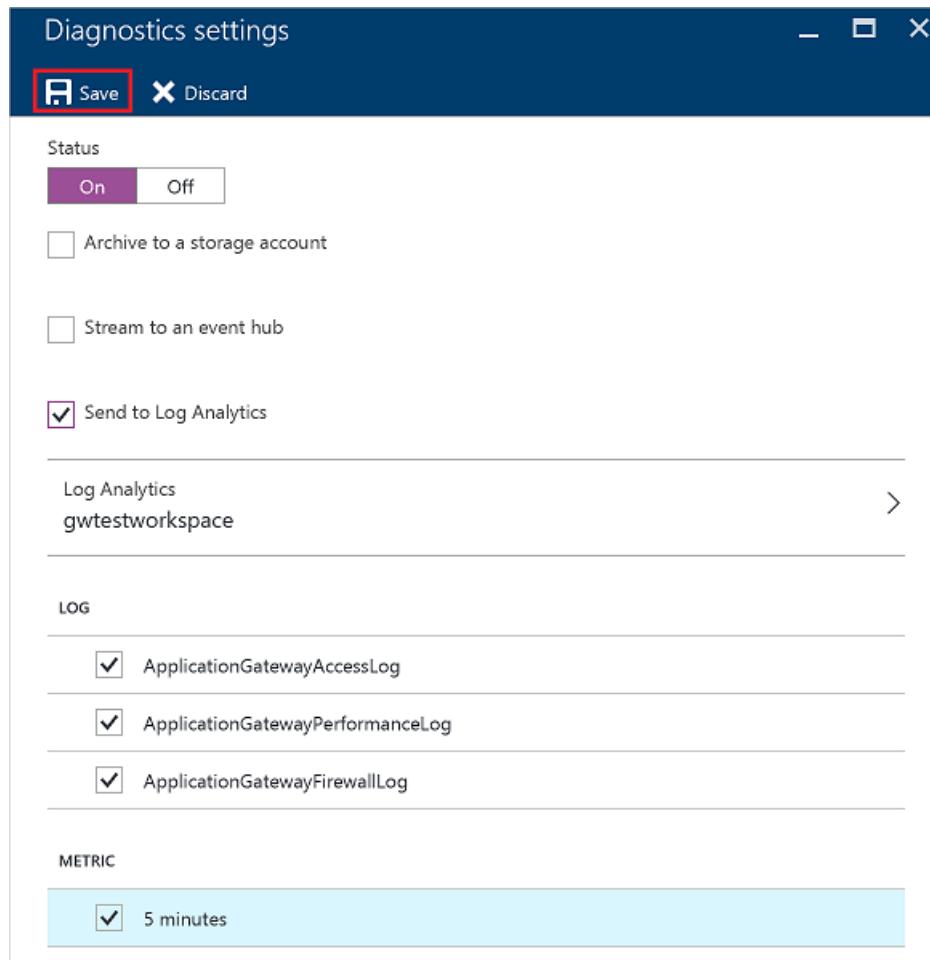
2. To start to collect data, click **Turn on diagnostics**.

The screenshot shows the 'Diagnostics logs' blade for an Application Gateway named 'AdatumAppGateway'. The left sidebar lists various monitoring and configuration options. The main area shows the 'Turn on diagnostics' section, which lists the logs to be collected: ApplicationGatewayAccessLog, ApplicationGatewayPerformanceLog, and ApplicationGatewayFirewallLog.

3. The **Diagnostics settings** blade provides the settings for the diagnostic logs. In this example, Log Analytics stores the logs. Click **Configure** under **Log Analytics** to set your workspace. You can also use event hubs and a storage account to save the diagnostic logs.

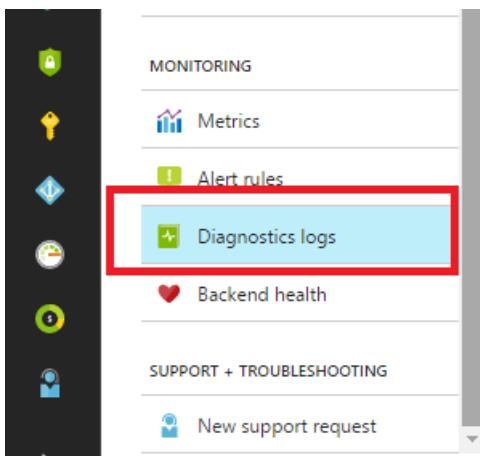


4. Confirm the settings and then click **Save**.



## View and analyze the Application Gateway access logs

1. In the Azure portal under the Application Gateway resource view, select **Diagnostics logs** in the **MONITORING** section.



2. On the right side, select “**ApplicationGatewayAccessLog**” in the drop-down list under **Log categories**.

The screenshot shows the 'Log categories' dropdown menu. It includes a checkbox for 'Select all' and three specific log categories: 'ApplicationGatewayAccessLog' (which is checked and highlighted with a blue dashed border), 'ApplicationGatewayPerformanceLog', and 'ApplicationGatewayFirewallLog'. To the right of the dropdown are 'Timespan' and 'LAST UPDATED' filters.

3. In the Application Gateway Access Log list, click the log you want to analyze and export, and then export the JSON file.
4. Convert the JSON file that you exported in step 3 to CSV file and view them in Excel, Power BI, or any other data-visualization tool.
5. Check the following data:
  - **ClientIP** – This is the client IP address from the connecting client.
  - **ClientPort** - This is the source port from the connecting client for the request.
  - **RequestQuery** – This indicates the destination server that the request is received.
  - **Server-Routed**: Back-end pool instance that the request is received.
  - **X-AzureApplicationGateway-LOG-ID**: Correlation ID used for the request. It can be used to troubleshoot traffic issues on the back-end servers. For example: X-AzureApplicationGateway-CACHE-HIT=0&SERVER-ROUTED=10.0.2.4.
    - **SERVER-STATUS**: HTTP response code that Application Gateway received from the back end.

The screenshot shows a single JSON log entry from the Application Gateway Access Log. The log contains fields such as resourceId, operationName, time, category, properties, userAgent, and timeTaken. The 'clientIP' and 'clientPort' fields are highlighted in yellow, and the 'X-AzureApplicationGateway-CACHE-HIT=0&SERVER-ROUTED=10.0.2.4' field is also highlighted in yellow.

```
{"resourceId": "00000000-0000-0000-0000-000000000000", "operationName": "ApplicationGatewayAccess", "time": "2017-01-26T21:58:54Z", "category": "ApplicationGatewayAccessLog", "properties": {"instanceId": "ApplicationGatewayRole_IN_0", "clientIP": "10.0.2.4", "clientPort": "443", "X-AzureApplicationGateway-CACHE-HIT=0&SERVER-ROUTED=10.0.2.4", "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36", "timeTaken": 0}
```

If you see two items are coming from the same ClientIP and Client Port, and they are sent to the same back-end server, that means the Application Gateway configured correctly.

If you see two items are coming from the same ClientIP and Client Port, and they are sent to the different back-end servers, that means the request is bouncing between backend servers, select "**Application is using cookie-based affinity but requests still bouncing between back-end servers**" at the bottom to troubleshoot it.

### 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.

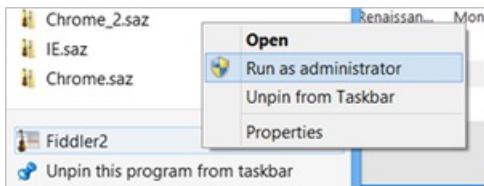
Use the web debugger of your choice. In this sample we will use Fiddler to capture and analyze http or https traffics, follow the instructions:

1. Download the Fiddler tool at <https://www.telerik.com/download/fiddler>.

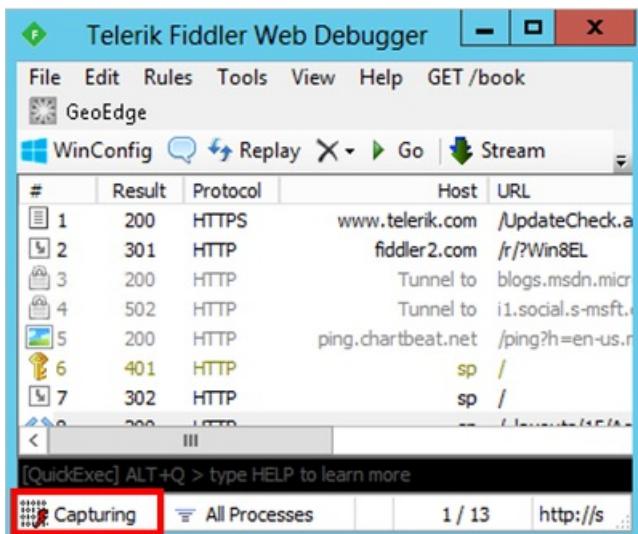
#### NOTE

Choose Fiddler4 if the capturing computer has .NET 4 installed. Otherwise, choose Fiddler2.

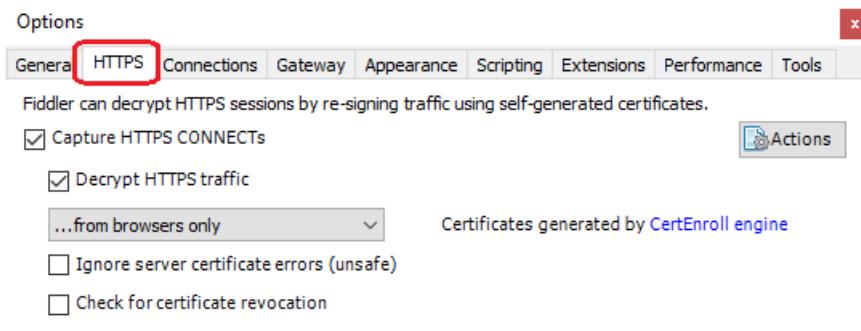
2. Right click the setup executable, and run as administrator to install.



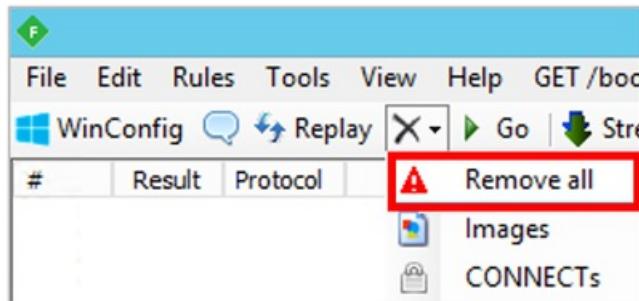
3. When you open Fiddler, it should automatically start capturing traffic (notice the Capturing at lower-left-hand corner). Press F12 to start or stop traffic capture.



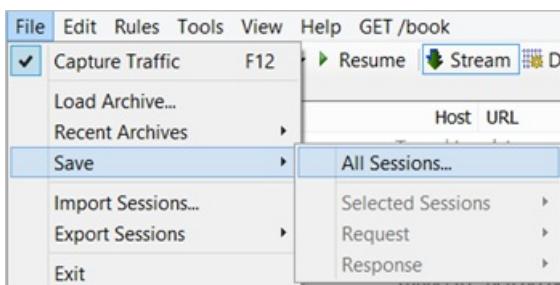
4. Most likely, you will be interested in decrypted HTTPS traffic, and you can enable HTTPS decryption by selecting **Tools > Fiddler Options**, and check the box " **Decrypt HTTPS traffic**".



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: ARRAffinity=ARRAffinityValue.**" If there's no cookie, session affinity isn't set, or the Application Gateway isn't applying cookie back to the client.

#### NOTE

This ARRAffinity value is the cookie-id, that the Application Gateway sets for the client to be sent to a particular back-end server.

The screenshot shows a WinConfig Network Miniserver interface. The session log pane displays several requests and responses. The selected request is a GET / HTTP/1.1 from a client. The 'Cookies' tab in the bottom navigation bar is active, showing a single cookie named 'ARRAffinity' with a value of 'ARRAffinity=81d22c0de530cc453a5d85d2fc1801290151080071a0296c3ab01c23692a46f8'. The cookie is highlighted in yellow.

- Example B:** The next session log followed by the previous one is the client responding back to the Application Gateway, which has set the ARRAffinity cookie-id. If the ARRAffinity 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 ARRAffinity cookie is changing.

This screenshot shows the continuation of the session. A new request is selected, which is a GET / HTTP/1.1. The 'Cookies' tab is still active, and the same ARRAffinity cookie is present, indicating the client is sending the cookie back to the Application Gateway.

#### NOTE

For the same communication session, the cookie should not 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

11/15/2019 • 6 minutes to read • [Edit Online](#)

Learn how to troubleshoot bad gateway (502) errors received when using Azure Application Gateway.

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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	<code>http://127.0.0.1/</code>	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

- Ensure that a default site is configured and is listening at 127.0.0.1.
- If BackendHttpSetting specifies a port other than 80, the default site should be configured to listen at that port.
- The call to `http://127.0.0.1:port` should return an HTTP result code of 200. This should be returned within the 30-second timeout period.
- Ensure that the port configured is open and that there are no firewall rules or Azure Network Security Groups,

which block 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. If the application gateway does not receive a response from back-end application in this interval, 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](#).

# Troubleshoot common questions or issues with Ingress Controller

11/7/2019 • 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](https://shell.azure.com) or by clicking the link:

 [Launch Cloud Shell](#)

## 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 setup 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: 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
```

Copy and paste all lines at once from the script above into a [Azure Cloud Shell](#). Please ensure the entire command is copied - starting with `cat` and including the last `EOF`.

The screenshot shows the Azure Cloud Shell interface. At the top, there are navigation icons (back, forward, refresh) and a URL bar displaying <https://shell.azure.com>. Below the URL bar is the title "Azure Cloud Shell". The main area is a terminal window with a dark background and white text. It displays the command history and output of a user named "delyan@Azure". The user is executing a series of commands to apply Kubernetes resources from a file. The output shows the creation of a Pod, a Service, and an Ingress resource, all with the name "test-agic-app". The deployment is successful, as indicated by the green text "Succeeded". The terminal ends with a prompt "[ ]".

```
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.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	NODE	READINESS	GATES
ingress-azure-66747895bb-ttppds	1/1	Running	0	18m	10.0.0.31	aks-agentpool-35064155-1	<none>	<none>	<none>	
mic-67cc97d599-4qnf1	1/1	Running	1	3d	10.0.0.18	aks-agentpool-35064155-1	<none>	<none>	<none>	
mic-67cc97d599-ptp81	1/1	Running	0	3d	10.0.0.13	aks-agentpool-35064155-1	<none>	<none>	<none>	
nmi-d9jqz	1/1	Running	0	3d	10.0.0.4	aks-agentpool-35064155-1	<none>	<none>	<none>	
test-agic-app-pod	1/1	Running	0	34s	10.0.0.33	aks-agentpool-35064155-1	<none>	<none>	<none>	

Get the list of services: `kubectl get services -o wide`. We expect to see a service named 'test-agic-app-service'.

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'.

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 have 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:

The screenshot shows the Azure portal interface for managing an Application Gateway. The left sidebar has navigation links: Home, applicationgateway4a7b - Listeners, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Settings. The main area is titled 'applicationgateway4a7b - Listeners' under 'Application Gateway'. It includes a search bar, a 'Basic' button, a 'Multi-site' button, a 'Save' button, and a 'Discard' button. Below these are tabs for 'Search listeners', 'Name', 'Protocol', and 'Port'. Two listeners are listed: 'fl-80' (Protocol: HTTP, Port: 80) and 'fl-test.agic.contoso.com-80' (Protocol: HTTP, Port: 80). The second listener is highlighted with a red box.

Name	Protocol	Port
fl-80	HTTP	80
fl-test.agic.contoso.com-80	HTTP	80

SSL Policy

- Routing Rule:

Home > applicationgateway4a7b - Listeners > fl-test.agic.contoso.com-80 > Rules > rr-test.agic.contoso.com-80

## rr-test.agic.contoso.com-80

applicationgateway4a7b

[Edit](#) [Delete](#)

Name  
rr-test.agic.contoso.com-80

Type  
Basic

Listener  
fl-test.agic.contoso.com-80

Backend pool  
pool-default-test-agic-app-service-80-bp-80

HTTP setting  
bp-default-test-agic-app-service-80-80-test-agic-app-ingress

- 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`

Home > applicationgateway4a7b - Listeners > fl-test.agic.contoso.com-80 > Rules > rr-test.agic.contoso.com-80 > Edit backend pool

## Edit backend pool

A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machine scale sets, IP addresses, or a valid Internet hostname.

Name \*  
pool-default-test-agic-app-service-80-bp-80

Add backend pool without targets

Backend targets  
1 item

Target type	Target	⋮
IP address or hostname	10.0.0.92	⋮
IP address or hostname		

Associated rule  
rr-test.agic.contoso.com-80

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' <public-ip-address-from-previous-command>`

```

delyan@Azure:~$ kubectl get ingress
NAME HOSTS ADDRESS PORTS AGE
test-agic-app-ingress test.agic.contoso.com 52.143.73.84 80 20m
delyan@Azure:~$ curl -I -H 'Host: test.agic.contoso.com' 52.143.73.84
HTTP/1.1 200 OK
Date: Fri, 27 Sep 2019 23:13:07 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Server: Kestrel

delyan@Azure:~$

```

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 must be in place for AGIC to function as expected:

- AKS must have one or more healthy **pods**. Verify this 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

```

- One or more **services**, referencing the pods above via matching `selector` labels. Verify this 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>

```

- Ingress**, annotated with `kubernetes.io/ingress.class: azure/application-gateway`, referencing the service above Verify this 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>

```

- 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 is not healthy (`STATUS` column from the command above is not `Running`):
  - get logs to understand why: `kubectl logs <pod-name>`
  - 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:
  - in your terminal via `kubectl get events --sort-by=.metadata.creationTimestamp`
  - in your browser using the [Kubernetes Web UI \(Dashboard\)](#)

## Logging Levels

AGIC has 3 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 9 levels of logging for the `kubectl` tool. In this repository we are utilizing 3 of these, 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 re-install
- 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 accessible 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> --sdk-auth | base64 -w0" >>

#####
Specify if the cluster is 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

7/8/2019 • 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.

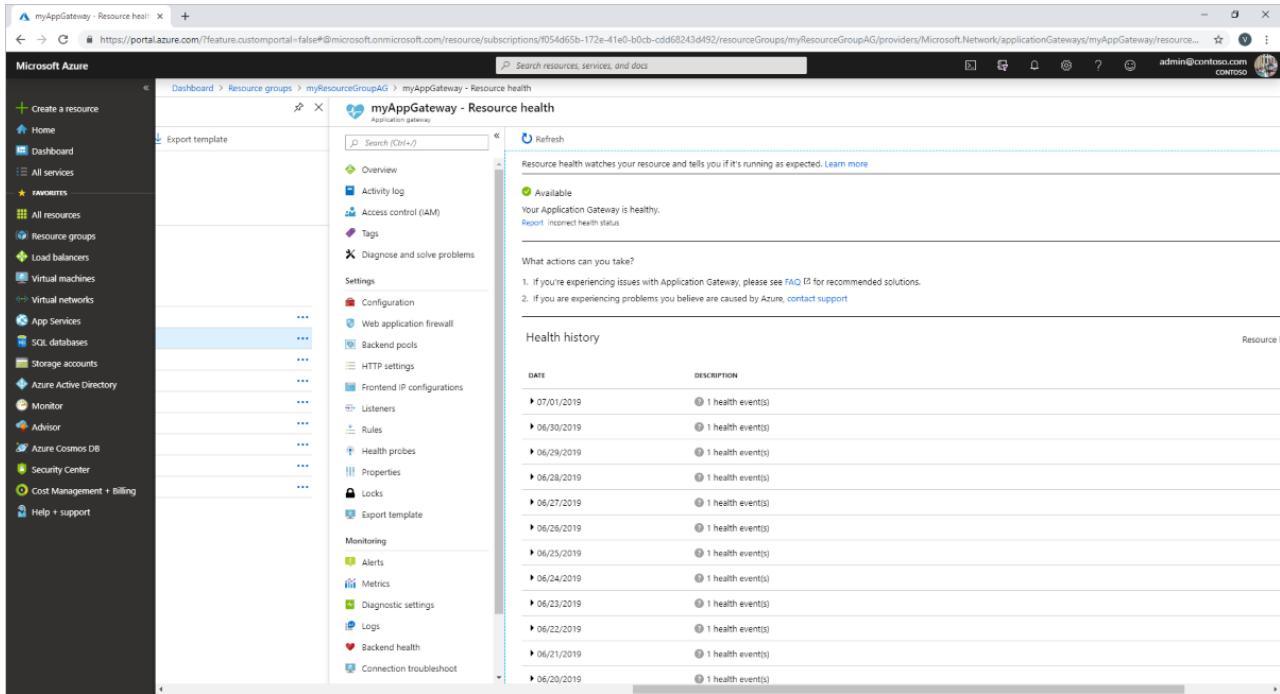
For Application Gateway, Resource Health relies on signals emitted by the gateway to assess whether it's healthy or not. If the gateway is unhealthy, Resource Health analyzes additional information to determine the source of the problem. It also identifies actions that Microsoft is taking or what you 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.



Date	Description
07/01/2019	1 health event(s)
06/30/2019	1 health event(s)
06/29/2019	1 health event(s)
06/28/2019	1 health event(s)
06/27/2019	1 health event(s)
06/26/2019	1 health event(s)
06/25/2019	1 health event(s)
06/24/2019	1 health event(s)
06/23/2019	1 health event(s)
06/22/2019	1 health event(s)
06/21/2019	1 health event(s)
06/20/2019	1 health event(s)

## 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.

The screenshot shows the Azure Application Gateway Resource Health page. On the left, a sidebar menu includes: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Configuration, Web application firewall, Backend pools, HTTP settings, Frontend IP configurations, Listeners, Rules, Rewrites, Health probes, Properties, Locks, Export template), Monitoring (Alerts, Metrics, Diagnostic settings, Backend health, Connection troubleshoot), Support + troubleshooting (Resource health, New support request), and a Search bar. The 'Resource health' item under 'Support + troubleshooting' is highlighted with a blue background. The main content area has a header 'Resource health watches your resource and tells you if it's running as expected. [Learn more](#)'. It displays two events:

- Unavailable**: Your Application Gateway is unavailable. [Report incorrect health status](#).  
What actions can you take?
  - If you're experiencing issues with Application Gateway, please see [FAQ](#) for recommended solutions.
  - If you are experiencing problems you believe are caused by Azure, [contact support](#)

Health history (Resource health events over the last 2 weeks)

DATE	DESCRIPTION
06/06/2019	<b>Unavailable</b> At Thursday, June 6, 2019, 1:44:11 PM PDT, the Azure monitoring system received the following information regarding your Application Gateway: Your Application Gateway is unavailable. <b>Recommended Steps</b> <ul style="list-style-type: none"><li>If you're experiencing issues with Application Gateway, please see <a href="#">FAQ</a> for recommended solutions.</li><li>If you are experiencing problems you believe are caused by Azure, <a href="#">contact support</a></li></ul> <a href="#">Download as PDF</a> <a href="#">Was this helpful? Yes   No</a>

**Degraded**: Performance degraded  
At Thursday, June 6, 2019, 1:32:11 PM PDT, the Azure monitoring system received the following information regarding your Application Gateway:  
Performance of your Application Gateway is degraded. 50% of all instances are healthy and serving traffic.  
**Recommended Steps**
  - If you're experiencing issues with Application Gateway, please see [FAQ](#) for recommended solutions.
  - If you are experiencing problems you believe are caused by Azure, [contact support](#)[Download as PDF](#) [Was this helpful? Yes | No](#)

## 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.

The screenshot shows the Azure portal interface for managing an Application Gateway. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Configuration, Web application firewall, Backend pools, HTTP settings, Frontend IP configurations, Listeners, Rules, Health probes, Properties, Locks, Export template), Monitoring (Alerts, Metrics, Diagnostic settings, Backend health, Connection troubleshoot), Support + troubleshooting (New support request), and Resource health (selected). The main content area displays a timeline of health events from May 22, 2019, to May 06, 2019. A single event is listed under the date May 06, 2019, with a status of "Unknown". The event details state: "At Wednesday, May 22, 2019, 5:00:00 PM PDT, the Azure monitoring system received the following information regarding your Application Gateway: The health status for this Application Gateway is unknown." Below this, a section titled "Recommended steps" provides five items of advice. At the bottom right of the event card, there is a link to "Download as PDF" and a "Was this helpful? Yes | No" feedback button.

Date	Status
06/06/2019	1 health event(s)
06/05/2019	1 health event(s)
06/04/2019	1 health event(s)
06/03/2019	1 health event(s)
06/02/2019	1 health event(s)
06/01/2019	1 health event(s)
05/31/2019	1 health event(s)
05/30/2019	1 health event(s)
05/29/2019	1 health event(s)
05/28/2019	1 health event(s)
05/27/2019	1 health event(s)
05/26/2019	1 health event(s)
05/25/2019	1 health event(s)
05/24/2019	1 health event(s)
05/23/2019	1 health event(s)
05/22/2019	1 health event(s)

## Degraded

The **Degraded** health status indicates your gateway has detected a loss in performance, although it's still available for usage.

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

---

Settings

Configuration

Web application firewall

Backend pools

HTTP settings

Frontend IP configurations

Listeners

Rules

Rewrites

Health probes

Properties

Locks

Export template

---

Monitoring

Alerts

Metrics

Diagnostic settings

Backend health

Connection troubleshoot

---

Support + troubleshooting

Resource health

New support request

Refresh

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

---

⚠ Degraded

Performance of your Application Gateway is degraded. 50% of all instances are healthy and serving traffic.

[Report incorrect health status](#)

---

What actions can you take?

1. If you're experiencing issues with Application Gateway, please see [FAQ](#) for recommended solutions.
2. If you are experiencing problems you believe are caused by Azure, [contact support](#)

---

Health history

Resource health events over the last 2 weeks

DATE	DESCRIPTION
▶ 06/06/2019	⚠ 1 health event(s)
06/05/2019	✓ Available
06/04/2019	✓ Available
▶ 06/03/2019	⚠ 1 health event(s)
▶ 06/02/2019	⚠ 1 health event(s)
▶ 06/01/2019	⚠ 1 health event(s)
▶ 05/31/2019	⚠ 1 health event(s)
▶ 05/30/2019	⚠ 1 health event(s)
▶ 05/29/2019	⚠ 1 health event(s)
▶ 05/28/2019	⚠ 1 health event(s)
▶ 05/27/2019	⚠ 1 health event(s)
▶ 05/26/2019	⚠ 1 health event(s)
▶ 05/25/2019	⚠ 1 health event(s)
▶ 05/24/2019	⚠ 1 health event(s)
▶ 05/23/2019	⚠ 1 health event(s)
▶ 05/22/2019	⚠ 1 health event(s)

## 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 Logs

11/13/2019 • 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:

The screenshot shows the Azure Log Analytics interface with a query results table. The table has columns: TimeGenerated [UTC], clientIp\_s, ruleSetType\_s, ruleSetVersion\_s, ruleId\_s, Message, action\_s, site\_s, and details\_message\_s. The table contains 10 rows of data, mostly from July 8, 2019, at 40:51:48. Most entries show 'Matched' actions and 'Global' sites, with various messages related to IP address validation and header anomalies. One entry shows 'Blocked' action and 'Global' site, with a message about access denied due to code 403.

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

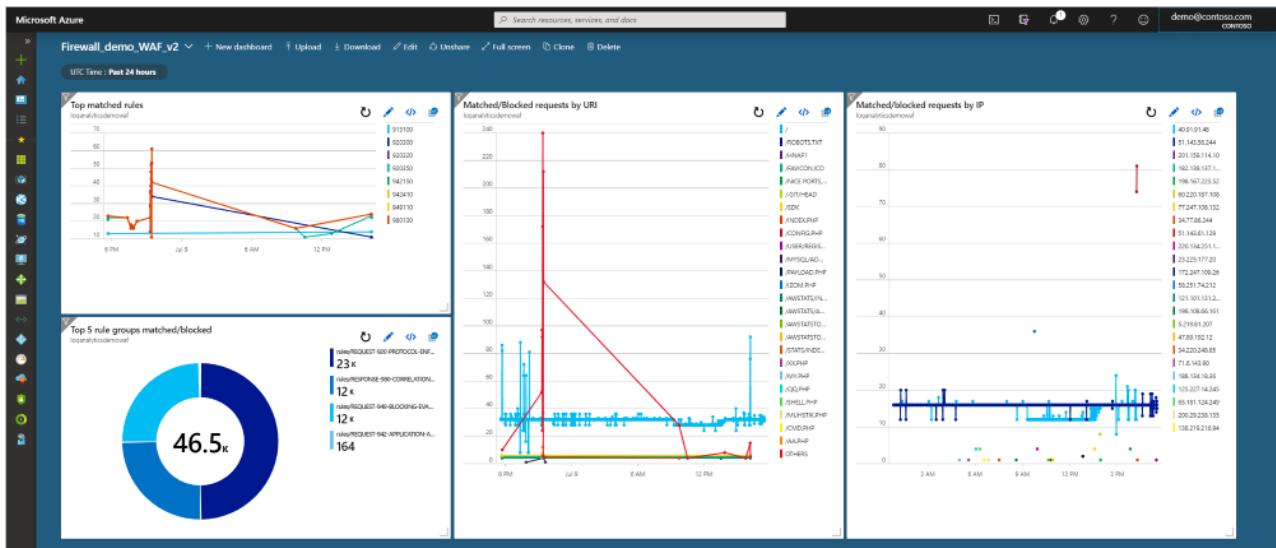
```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.NETWORK" and Category == "ApplicationGatewayFirewallLog"
| summarize count() by ruleId_s, bin(TimeGenerated, 1m)
| where count_ > 10
| render timechart
```

## 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

10/7/2019 • 19 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:

- 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:

```

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/backendAddressPools/appGatewayBackendPool"
 },
 "BackendHttpSettingsCollection": [
 {
 "BackendHttpSettings": {
 "TrustedRootCertificates": [],
 "Id": "/subscriptions/536d30b8-665b-40fc-bd7e-68c65f816365/resourceGroups/rgOne/providers/Microsoft.Network/applicationGateways/appgw1/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, 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 and 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 Domain Name System (DNS) (custom or Azure default) and 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, and 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.

#### TCP connect error

**Message:** Application Gateway could not connect to the backend. Please 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 an SSL handshake is done (if SSL 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 range 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 at this point. 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 have an option to 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](#).

#### **Backend server certificate invalid CA**

**Message:** The server certificate used by the backend is not signed by a well-known Certificate Authority (CA). Whitelist 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 an 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 just 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 an 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 1-11 in the preceding method 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)\ndepth=0 OU = Domain Control Validated, CN = *.example.com\\n\nverify error:num=20:unable to get local issuer certificate\\n\nverify return:1\\n\ndepth=0 OU = Domain Control Validated, CN = *.example.com\\n\nverify error:num=21:unable to verify the first certificate\\n\nverify return:1\\n\n\\-\\-\\-\nCertificate chain\\n\n0 s:/OU=Domain Control Validated/CN=*.example.com\\n\n\ti:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./OU=http://certs.godaddy.com/repository//CN=Go Daddy\nSecure Certificate Authority - G2\\n\n\t-----BEGIN CERTIFICATE-----\\n\nxx\\n\n\t-----END CERTIFICATE-----\\n
```

#### 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 SSL certificate. This is Standard\_v2 and WAF\_v2 SKU behavior. The Standard and WAF SKU's Server Name Indication (SNI) is set as the FQDN in the backend pool address.

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 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 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 Open SSL 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 **Any** or **Internet**.
  - 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 server (or 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](#).

## Next steps

Learn more about [Application Gateway diagnostics and logging](#).

# Migrate Azure Application Gateway and Web Application Firewall from v1 to v2

2/5/2020 • 9 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.
- To migrate an SSL configuration, you must specify all the 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.

## 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:

```
Install-Script -Name AzureAppGWMigration
```

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>
-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. The script creates it for you if it doesn't exist.
- **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.

- **sslCertificates: [PSApplicationGatewaySslCertificate]: Optional.** A comma-separated list of PSApplicationGatewaySslCertificate objects that you create to represent the SSL certs from your v1 gateway must be uploaded to the new v2 gateway. For each of your 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 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/myv1appgate
way `
 -subnetAddressRange 10.0.0.0/24 `
 -appgwname "MynewV2gw" `
 -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 send an email to appgwmigrationsup@microsoft.com, open a support case with Azure Support, or do both.

## Next steps

[Learn about Application Gateway v2](#)

# Annotations for Application Gateway Ingress Controller

11/7/2019 • 3 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
<code>appgw.ingress.kubernetes.io/backend-path-prefix</code>	<code>string</code>	<code>nil</code>	
<code>appgw.ingress.kubernetes.io/ssl-redirect</code>	<code>bool</code>	<code>false</code>	
<code>appgw.ingress.kubernetes.io/connection-draining</code>	<code>bool</code>	<code>false</code>	
<code>appgw.ingress.kubernetes.io/connection-draining-timeout</code>	<code>int32</code> (seconds)	<code>30</code>	
<code>appgw.ingress.kubernetes.io/cookie-based-affinity</code>	<code>bool</code>	<code>false</code>	
<code>appgw.ingress.kubernetes.io/request-timeout</code>	<code>int32</code> (seconds)	<code>30</code>	
<code>appgw.ingress.kubernetes.io/use-private-ip</code>	<code>bool</code>	<code>false</code>	
<code>appgw.ingress.kubernetes.io/backend-protocol</code>	<code>string</code>	<code>http</code>	<code>http, https</code>

## 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.

## SSL 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 be 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 supports `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
```

# Azure Resource Manager overview

12/23/2019 • 5 minutes to read • [Edit Online](#)

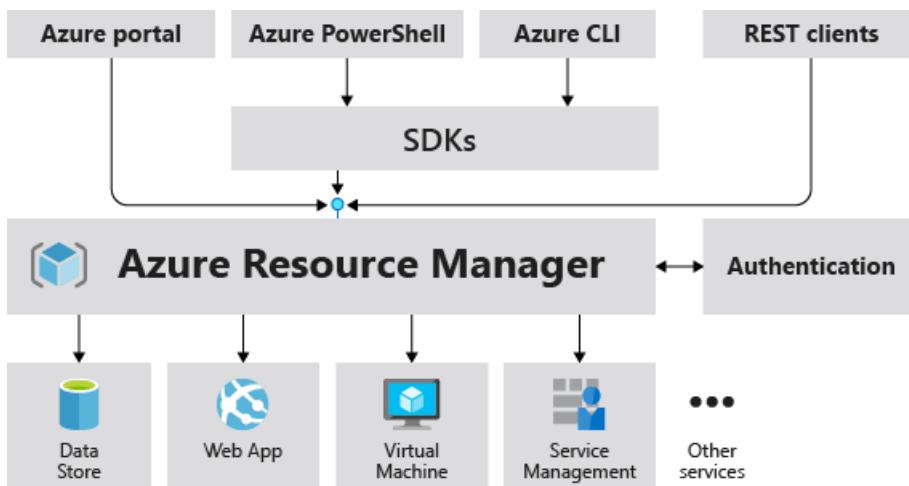
Azure Resource Manager is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure subscription. You use management features, like access control, locks, and tags, to secure and organize your resources after deployment.

To learn about Azure Resource Manager templates, see [Template deployment overview](#).

## Consistent management layer

When a user sends a request from any of the Azure tools, APIs, or SDKs, Resource Manager receives the request. It authenticates and authorizes the request. Resource Manager sends the request to the Azure service, which takes the requested action. 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 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](#).
- **Resource Manager template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group or subscription. The template can be used to deploy the resources consistently and repeatedly. See [Template deployment overview](#).

- **declarative syntax** - Syntax that lets you state "Here is what I intend to create" without having to write the sequence of programming commands to create it. The Resource Manager template is an example of declarative syntax. In the file, you define the properties for the infrastructure to deploy to Azure. See [Template deployment overview](#).

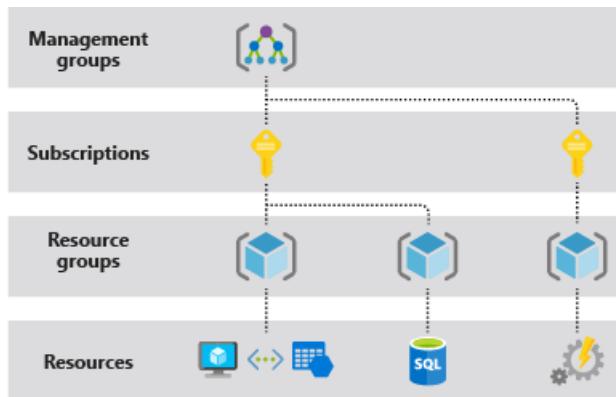
## The benefits of using Resource Manager

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 in your resource group because Role-Based Access Control (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.

You can deploy templates to 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 group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a database server, needs to exist on a different deployment cycle it should be in another resource group.

- Each resource can only exist in 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](#).
- A resource group can contain resources that are located in different regions.
- A resource group can be used to scope access control for administrative actions.
- A resource can interact with resources in other resource groups. This interaction is common when the two resources are related but don't share the same lifecycle (for example, web apps connecting to a database).

When creating 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 the 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. For more information about building reliable applications, see [Designing reliable Azure applications](#).

## 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 (as well 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

- For all the operations offered by resource providers, see the [Azure REST APIs](#).
- 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](#).
- For information about creating templates for deployments, see [Template deployment overview](#).

# Understand the structure and syntax of Azure Resource Manager templates

2/26/2020 • 13 minutes to read • [Edit Online](#)

This article describes the structure of an Azure Resource Manager 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 Resource Manager 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 Azure Resource Manager template](#).

## Template format

In its simplest structure, a template has the following elements:

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "",
 "apiProfile": "",
 "parameters": { },
 "variables": { },
 "functions": [],
 "resources": [],
 "outputs": { }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
\$schema	Yes	<p>Location of the JSON schema file that describes the version of the template language.</p> <p>For resource group deployments, use: <a href="https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#">https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#</a></p> <p>For subscription deployments, use: <a href="https://schema.management.azure.com/schemas/2015-01-01/subscriptionDeploymentTemplate.json#">https://schema.management.azure.com/schemas/2015-01-01/subscriptionDeploymentTemplate.json#</a></p>
contentVersion	Yes	<p>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.</p>

ELEMENT NAME	REQUIRED	DESCRIPTION
apiProfile	No	<p>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.</p> <p>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 <a href="#">API Profile</a>.</p> <p>For more information, see <a href="#">Track versions using API profiles</a>.</p>
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 <b>string</b> , <b>securestring</b> , <b>int</b> , <b>bool</b> , <b>object</b> , <b>secureObject</b> , and <b>array</b> . See <a href="#">Data types</a> .
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 <a href="#">Comments in templates</a> .

For examples of how to use parameters, see [Parameters in Azure Resource Manager templates](#).

## Data types

For integers passed as inline parameters, the range of values may be limited by the SDK or command-line tool you use for deployment. For example, when using PowerShell to deploy a template, integer types can range from -2147483648 to 2147483647. To avoid this limitation, specify large integer values in a [parameter file](#). Resource types apply their own limits for integer properties.

When specifying boolean and integer values in your template, don't surround the value with quotation marks. Start and end string values with double quotation marks.

Objects start with a left brace and end with a right brace. Arrays start with a left bracket and end with a right bracket.

Secure strings and secure objects can't be read after resource deployment.

For samples of formatting data types, see [Parameter type formats](#).

## 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 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 Azure Resource Manager template](#).

## 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 <code>"[contoso.uniqueName()]"</code> .
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 <b>string</b> , <b>securestring</b> , <b>int</b> , <b>bool</b> , <b>object</b> , <b>secureObject</b> , and <b>array</b> .
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 Azure Resource Manager template](#).

## 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>"
 },
 "sku": {
 "name": "<sku-name>",
 "tier": "<sku-tier>",
 "size": "<sku-size>",
 "family": "<sku-family>",
 "capacity": <sku-capacity>
 },
 "kind": "<type-of-resource>",
 "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 <code>true</code> , the resource is created during deployment. When <code>false</code> , the resource is skipped for this deployment. See <a href="#">condition</a> .

ELEMENT NAME	REQUIRED	DESCRIPTION
type	Yes	<p>Type of the resource. This value is a combination of the namespace of the resource provider and the resource type (such as <b>Microsoft.Storage/storageAccounts</b>). To determine available values, see <a href="#">template reference</a>. 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 <a href="#">Set name and type for child resources</a>.</p>
apiVersion	Yes	<p>Version of the REST API to use for creating the resource. To determine available values, see <a href="#">template reference</a>.</p>
name	Yes	<p>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 <a href="#">Set name and type for child resources</a>.</p>
comments	No	<p>Your notes for documenting the resources in your template. For more information, see <a href="#">Comments in templates</a>.</p>
location	Varies	<p>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 <a href="#">Set resource location</a>.</p>
dependsOn	No	<p>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 <a href="#">Defining dependencies in Azure Resource Manager templates</a>.</p>

ELEMENT NAME	REQUIRED	DESCRIPTION
tags	No	Tags that are associated with the resource. Apply tags to logically organize resources across your subscription.
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.
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 <a href="#">Create several instances of resources in Azure Resource Manager</a> .
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 <a href="#">template reference</a> .
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 <a href="#">Set name and type for child resources</a> .

## 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.
condition	No	Boolean value that indicates whether this output value is returned. When <code>true</code> , the value is included in the output for the deployment. When <code>false</code> , the output value is skipped for this deployment. When not specified, the default value is <code>true</code> .
type	Yes	Type of the output value. Output values support the same types as template input parameters. If you specify <b>securestring</b> 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 <a href="#">Use Azure Key Vault to pass secure parameter value during deployment</a> .
value	No	Template language expression that is evaluated and returned as output value. Specify either <b>value</b> or <b>copy</b> .
copy	No	Used to return more than one value for an output. Specify <b>value</b> or <b>copy</b> . For more information, see <a href="#">Output iteration in Azure Resource Manager templates</a> .

For examples of how to use outputs, see [Outputs in Azure Resource Manager template](#).

## Comments and metadata

You have a few options for adding comments and metadata to your template.

### Comments

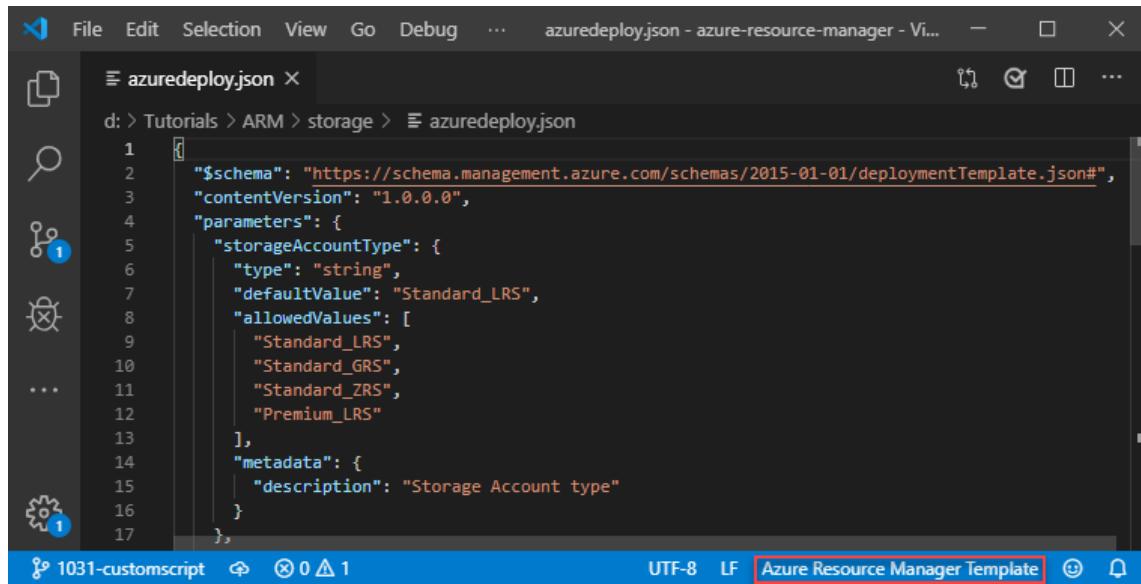
For inline comments, you can use either `//` or `/* ... */` but this syntax doesn't work with all tools. You can't use the portal template editor to work on templates with inline comments. If you add this style of comment, be sure the tools you use support inline JSON comments.

#### NOTE

To deploy templates with comments by using Azure CLI, you must use 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 Resource Manager template and change the language mode accordingly. If you see **Azure Resource Manager Template** at the bottom-right corner of VS Code, you can use the inline comments. The inline comments are no longer marked as invalid.



## 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/2015-01-01/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.

SETTINGS	User name for the Virtual Machine.
* Admin Username <small>i</small>	<input type="text"/>
* Admin Password <small>i</small>	<input type="password"/>

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.

```
{
 "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'))]"
],
```

To deploy templates with multi-line strings by using Azure CLI, you must use the `--handle-extended-json-format` switch.

## 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 [Azure Resource Manager Template Functions](#).
- To combine several templates during deployment, see [Using linked templates with Azure Resource Manager](#).
- For recommendations about creating templates, see [Azure Resource Manager template best practices](#).
- For recommendations on creating Resource Manager templates that you can use across all Azure environments and Azure Stack, see [Develop Azure Resource Manager templates for cloud consistency](#).