

# Table of Contents

## Overview

[What is Azure CDN?](#)

## Get Started

[Enable Azure CDN](#)

## How To

### Integrate

[Web Apps](#)

[Cloud Services](#)

[Storage](#)

[Cross-origin resource sharing](#)

### Manage

[Enable HTTPS on a custom domain](#)

[Manage with PowerShell](#)

[Configure time-to-live](#)

[Map a custom domain to CDN](#)

[Restrict access by country](#)

[Improve performance by compressing files](#)

[Cache content by query string](#)

[Purge cached assets](#)

[Pre-load cached assets](#)

[Token authentication](#)

[Monitor resources](#)

[Override behavior with rules](#)

[Get real-time alerts](#)

[HTTP/2 support](#)

### Analyze

[Analyze usage patterns](#)

[Generate advanced HTTP reports](#)

[View real-time statistics](#)

[Analyze edge node performance](#)

[Export metrics with Diagnostics Logs](#)

## [Develop](#)

[.NET](#)

[Node.js](#)

## [Troubleshoot](#)

[404 status](#)

[File compression](#)

## [Reference](#)

[PowerShell](#)

[.NET](#)

[Java](#)

[REST](#)

## [Resources](#)

[Rules Engine reference](#)

[Rules Engine conditional expressions](#)

[Rules Engine match conditions](#)

[Rules Engine features](#)

[Azure CDN POP locations](#)

[Service updates](#)

[Pricing](#)

[MSDN forum](#)

[Stack Overflow](#)

[Videos](#)

# Overview of the Azure Content Delivery Network (CDN)

5/5/2017 • 3 min to read • [Edit Online](#)

## NOTE

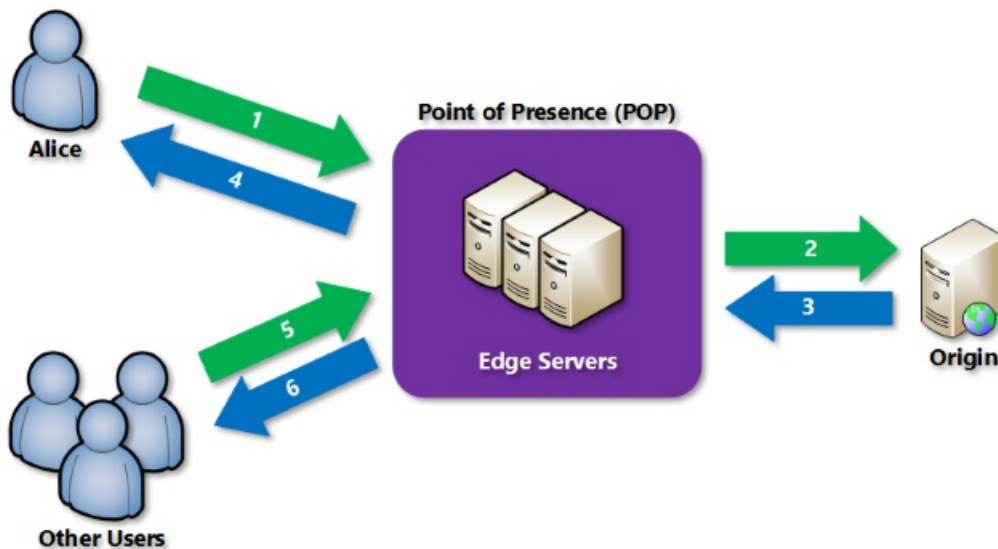
This document describes what the Azure Content Delivery Network (CDN) is, how it works, and the features of each Azure CDN product. If you want to skip this information and go straight to a tutorial on how to create a CDN endpoint, see [Using Azure CDN](#). If you want to see a list of current CDN node locations, see [Azure CDN POP Locations](#).

The Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN offers developers a global solution for delivering high-bandwidth content by caching the content at physical nodes across the world.

The benefits of using the CDN to cache web site assets include:

- Better performance and user experience for end users, especially when using applications where multiple round-trips are required to load content.
- Large scaling to better handle instantaneous high load, like at the start of a product launch event.
- By distributing user requests and serving content from edge servers, less traffic is sent to the origin.

## How it works



1. A user (Alice) requests a file (also called an asset) using a URL with a special domain name, such as `<endpointname>.azureedge.net`. DNS routes the request to the best performing Point-of-Presence (POP) location. Usually this is the POP that is geographically closest to the user.
2. If the edge servers in the POP do not have the file in their cache, the edge server requests the file from the origin. The origin can be an Azure Web App, Azure Cloud Service, Azure Storage account, or any publicly accessible web server.
3. The origin returns the file to the edge server, including optional HTTP headers describing the file's Time-to-Live (TTL).
4. The edge server caches the file and returns the file to the original requestor (Alice). The file remains cached on the edge server until the TTL expires. If the origin didn't specify a TTL, the default TTL is seven days.

5. Additional users may then request the same file using that same URL, and may also be directed to that same POP.
6. If the TTL for the file hasn't expired, the edge server returns the file from the cache. This results in a faster, more responsive user experience.

## Azure CDN Features

There are three Azure CDN products: **Azure CDN Standard from Akamai**, **Azure CDN Standard from Verizon**, and **Azure CDN Premium from Verizon**. The following table lists the features available with each product.

	STANDARD AKAMAI	STANDARD VERIZON	PREMIUM VERIZON
Easy integration with Azure services such as <a href="#">Storage</a> , <a href="#">Cloud Services</a> , <a href="#">Web Apps</a> , and <a href="#">Media Services</a>	✓	✓	✓
Management via <a href="#">REST API</a> , <a href="#">.NET</a> , <a href="#">Node.js</a> , or <a href="#">PowerShell</a> .	✓	✓	✓
HTTPS support with CDN endpoint	✓	✓	✓
Custom domain HTTPS		✓	✓
Load balancing	✓	✓	✓
<a href="#">DDOS</a> protection	✓	✓	✓
IPv4/IPv6 dual-stack	✓	✓	✓
<a href="#">Custom domain name support</a>	✓	✓	✓
<a href="#">Query string caching</a>	✓	✓	✓
<a href="#">Geo-filtering</a>	✓	✓	✓
<a href="#">Fast purge</a>	✓	✓	✓
<a href="#">Asset pre-loading</a>		✓	✓
<a href="#">Core analytics</a>		✓	✓
<a href="#">HTTP/2 support</a>	✓	✓	✓
<a href="#">Advanced HTTP reports</a>			✓
<a href="#">Real-time stats</a>			✓
<a href="#">Real-time alerts</a>			✓

	STANDARD AKAMAI	STANDARD VERIZON	PREMIUM VERIZON
Customizable, rule-based content delivery engine			✓
Cache/header settings (using <a href="#">rules engine</a> )			✓
URL redirect/rewrite (using <a href="#">rules engine</a> )			✓
Mobile device rules (using <a href="#">rules engine</a> )			✓
Token authentication			✓

#### TIP

Is there a feature you'd like to see in Azure CDN? [Give us feedback!](#)

## Next steps

To get started with CDN, see [Using Azure CDN](#).

If you are an existing CDN customer, you can now manage your CDN endpoints through the [Microsoft Azure portal](#) or with [PowerShell](#).

To see the CDN in action, check out the [video of our Build 2016 session](#).

Learn how to automate Azure CDN with [.NET](#) or [Node.js](#).

For pricing information, see [CDN Pricing](#).

4 min to read •

# Getting started with Azure CDN

1/24/2017 • 4 min to read • [Edit Online](#)

This topic walks through enabling Azure CDN by creating a new CDN profile and endpoint.

## IMPORTANT

For an introduction to how CDN works, as well as a list of features, see the [CDN Overview](#).

## Create a new CDN profile

A CDN profile is a collection of CDN endpoints. Each profile contains one or more CDN endpoints. You may wish to use multiple profiles to organize your CDN endpoints by internet domain, web application, or some other criteria.

## NOTE

By default, a single Azure subscription is limited to eight CDN profiles. Each CDN profile is limited to ten CDN endpoints.

CDN pricing is applied at the CDN profile level. If you wish to use a mix of Azure CDN pricing tiers, you will need multiple CDN profiles.

### To create a new CDN profile

1. In the [Azure Portal](#), in the upper left, click **New**. In the **New** blade, select **Web + Mobile**, then **CDN**.

The new CDN profile blade appears.

CDN profile

\* Name

\* Subscription

msdn

\* Resource group ⓘ

☒ Create new ☐ Use existing

\* Resource group location ⓘ

Central US

\* Pricing tier

Configure required settings

☐ Pin to dashboard

Create Automation options

2. Enter a name for your CDN profile.
3. Select a **Location**. This is the Azure location where your CDN profile information will be stored. It has no impact on CDN endpoint locations.
4. Select or create a **Resource Group**. For more information on Resource Groups, see [Azure Resource Manager overview](#).
5. Select a **Pricing tier**. See the [CDN Overview](#) for a comparison of pricing tiers.

Choose your pricing tier

Browse the available plans and their features

The premium pricing tier adds powerful features, advanced analytics, and more. Price varies based on region and data usage. [Learn more](#)

P1 Premium Verizon	S1 Standard Verizon	S2 Standard Akamai
SSL for CDN endpoint	SSL for CDN endpoint	SSL for CDN endpoint
Content Purge/Load	Content Purge/Load	Content Purge
Compression	Compression	Compression
Query strings caching	Query strings caching	Query strings caching
Country filtering	Country filtering	
Rules engine	Core analytics	
Advanced analytics		
Realtime analytics		

6. Select the **Subscription** for this CDN profile.



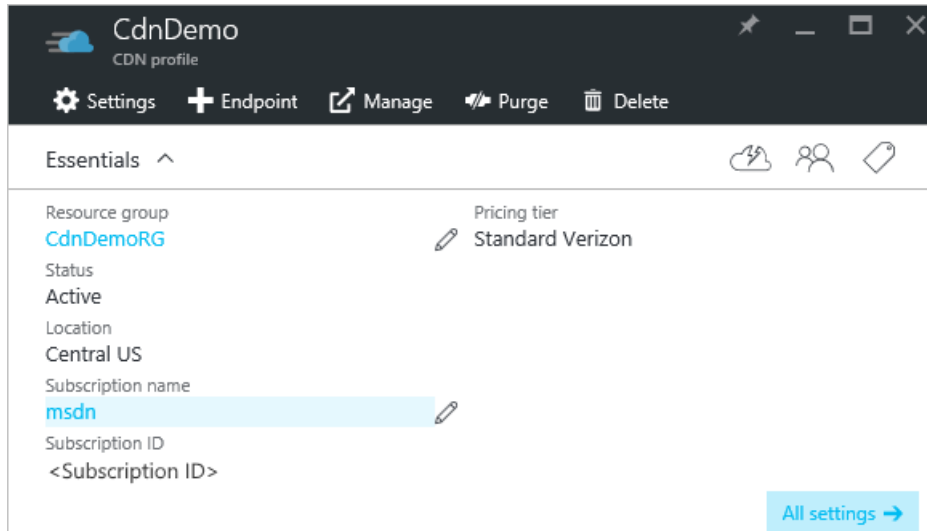
7. Click the **Create** button to create the new profile.

## Create a new CDN endpoint

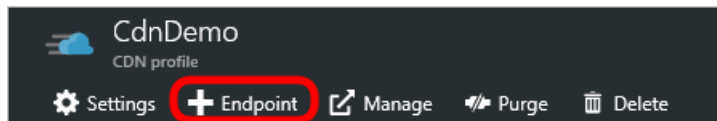
### To create a new CDN endpoint

1. In the [Azure Portal](#), navigate to your CDN profile. You may have pinned it to the dashboard in the previous step. If you not, you can find it by clicking **Browse**, then **CDN profiles**, and clicking on the profile you plan to add your endpoint to.

The CDN profile blade appears.



2. Click the **Add Endpoint** button.



The **Add an endpoint** blade appears.

Add an endpoint
Allows configuring content delivery behavior and access.

\* Name

.azureedge.net

\* Origin type

▼

\* Origin hostname ⓘ

▼

Origin path ⓘ

Origin host header ⓘ

Protocol ⓘ

☒ HTTP
☒ HTTPS

Origin port ⓘ

80
443

Add

- Enter a **Name** for this CDN endpoint. This name will be used to access your cached resources at the domain `<endpointname>.azureedge.net`.
- In the **Origin type** dropdown, select your origin type. Select **Storage** for an Azure Storage account, **Cloud service** for an Azure Cloud Service, **Web App** for an Azure Web App, or **Custom origin** for any other publicly accessible web server origin (hosted in Azure or elsewhere).

\* Origin type

Storage
Cloud service
Web App
Custom origin

- In the **Origin hostname** dropdown, select or type your origin domain. The dropdown will list all available origins of the type you specified in step 4. If you selected *Custom origin* as your **Origin type**, you will type in the domain of your custom origin.
- In the **Origin path** text box, enter the path to the resources you want to cache, or leave blank to allow cache any resource at the domain you specified in step 5.
- In the **Origin host header**, enter the host header you want the CDN to send with each request, or leave the default.

### WARNING

Some types of origins, such as Azure Storage and Web Apps, require the host header to match the domain of the origin. Unless you have an origin that requires a host header different from its domain, you should leave the default value.

- For **Protocol** and **Origin port**, specify the protocols and ports used to access your resources at the origin. At least one protocol (HTTP or HTTPS) must be selected.

### NOTE



The **Origin port** only affects what port the endpoint uses to retrieve information from the origin. The endpoint itself will only be available to end clients on the default HTTP and HTTPS ports (80 and 443), regardless of the **Origin port**.

**Azure CDN from Akamai** endpoints do not allow the full TCP port range for origins. For a list of origin ports that are not allowed, see [Azure CDN from Akamai Allowed Origin Ports](#).

Accessing CDN content using HTTPS has the following constraints:

- You must use the SSL certificate provided by the CDN. Third party certificates are not supported.
- You must use the CDN-provided domain ( `<endpointname>.azureedge.net` ) to access HTTPS content. HTTPS support is not available for custom domain names (CNAMEs) since the CDN does not support custom certificates at this time.

- Click the **Add** button to create the new endpoint.
- Once the endpoint is created, it appears in a list of endpoints for the profile. The list view shows the URL to use to access cached content, as well as the origin domain.

Endpoints		
1 		
HOSTNAME	STATUS	PROTOCOL
cdndemo.azureedge.net	 Running	HTTP, HTTPS

### IMPORTANT

The endpoint will not immediately be available for use, as it takes time for the registration to propagate through the CDN. For **Azure CDN from Akamai** profiles, propagation will usually complete within one minute. For **Azure CDN from Verizon** profiles, propagation will usually complete within 90 minutes, but in some cases can take longer.

Users who try to use the CDN domain name before the endpoint configuration has propagated to the POPs will receive HTTP 404 response codes. If it's been several hours since you created your endpoint and you're still receiving 404 responses, please see [Troubleshooting CDN endpoints returning 404 statuses](#).

## See Also

- [Controlling caching behavior of requests with query strings](#)
- [How to Map CDN Content to a Custom Domain](#)
- [Pre-load assets on an Azure CDN endpoint](#)
- [Purge an Azure CDN Endpoint](#)
- [Troubleshooting CDN endpoints returning 404 statuses](#)



# Use Azure CDN in Azure App Service

2/28/2017 • 19 min to read • [Edit Online](#)

[App Service](#) can be integrated with [Azure CDN](#), adding to the global scaling capabilities inherent in [App Service Web Apps](#) by serving your web app content globally from server nodes near your customers (an updated list of all current node locations can be found [here](#)). In scenarios like serving static images, this integration can dramatically increase the performance of your Azure App Service Web Apps and significantly improves your web app's user experience worldwide.

Integrating Web Apps with Azure CDN gives you the following advantages:

- Integrate content deployment (images, scripts, and stylesheets) as part of your web app's [continuous deployment](#) process
- Easily upgrade the NuGet packages in your web app in Azure App Service, such as jQuery or Bootstrap versions
- Manage your Web application and your CDN-served content from the same Visual Studio interface
- Integrate ASP.NET bundling and minification with Azure CDN

## NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

## What you will build

You will deploy a web app to Azure App Service using the default ASP.NET MVC template in Visual Studio, add code to serve content from an integrated Azure CDN, such as an image, controller action results, and the default JavaScript and CSS files, and also write code to configure the fallback mechanism for bundles served in the event that the CDN is offline.

## What you will need

This tutorial has the following prerequisites:

- An active [Microsoft Azure account](#)
- Visual Studio 2015 with the [Azure SDK for .NET](#). If you use Visual Studio, the steps may vary.

## NOTE

You need an Azure account to complete this tutorial:

- You can [open an Azure account for free](#) - You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services, such as Web Apps.
- You can [activate Visual Studio subscriber benefits](#) - Your Visual Studio subscription gives you credits every month that you can use for paid Azure services.

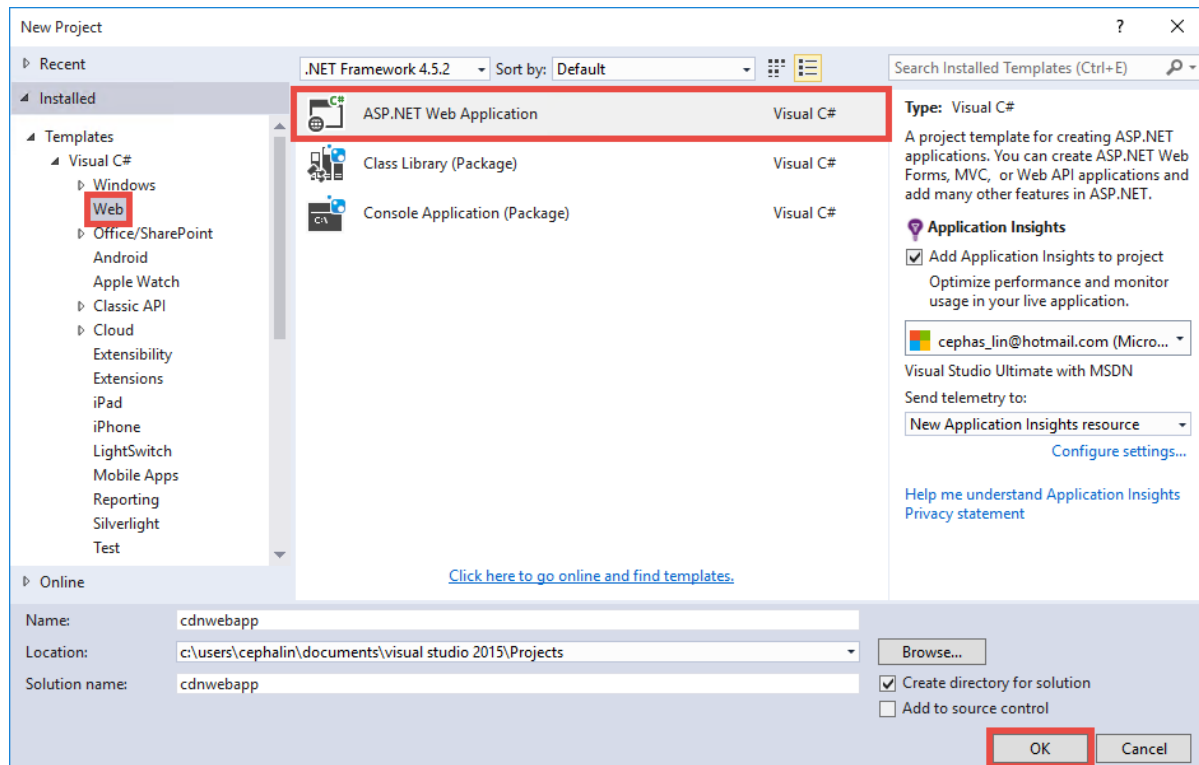
If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

## Deploy a web app to Azure with an integrated CDN endpoint

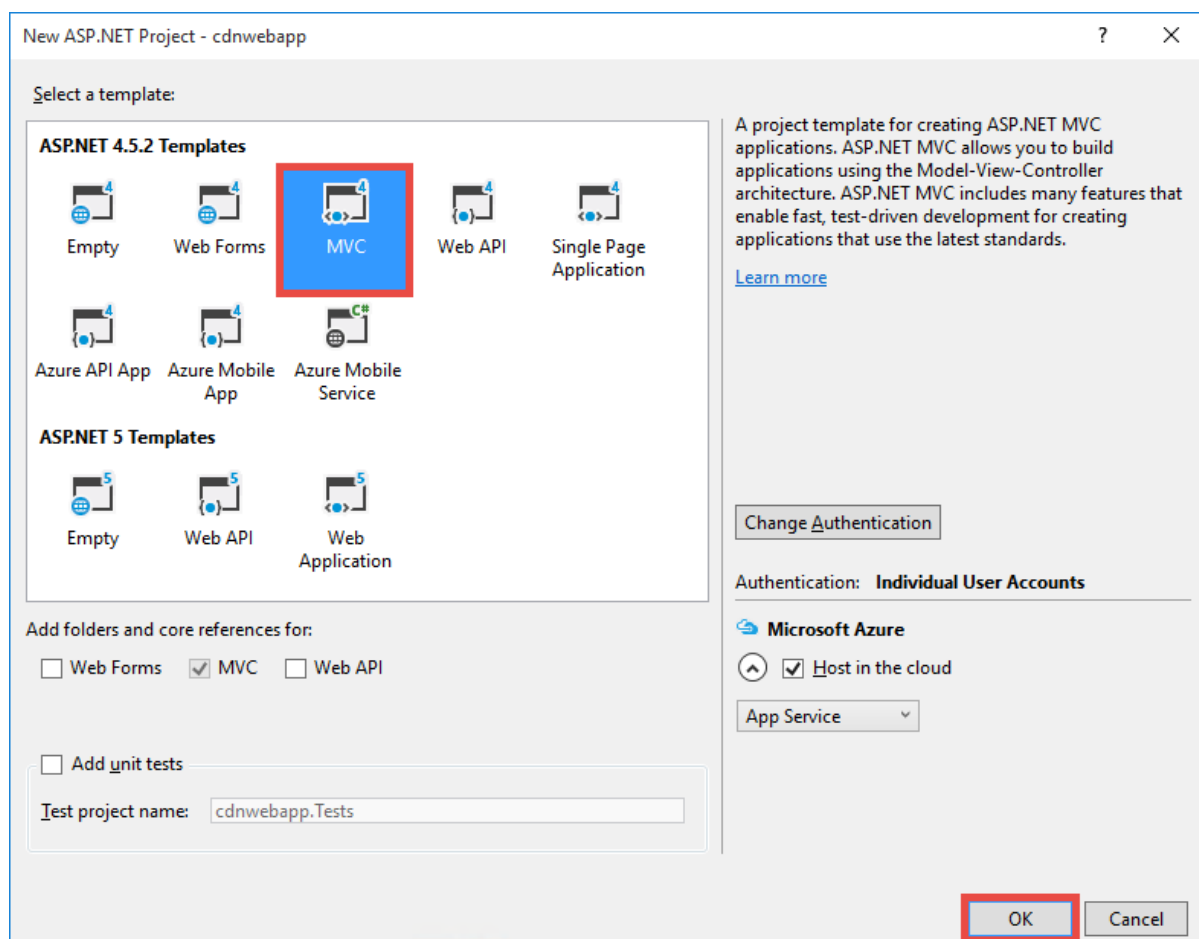
In this section, you will deploy the default ASP.NET MVC application template in Visual Studio 2015 to App Service,

and then integrate it with a new CDN endpoint. Follow the instructions below:

1. In Visual Studio 2015, create a new ASP.NET web application from the menu bar by going to **File > New > Project > Web > ASP.NET Web Application**. Give it a name and click **OK**.



2. Select **MVC** and click **OK**.



3. If you haven't logged into your Azure account yet, click the account icon in the upper-right corner and follow the dialog to log into your Azure account. Once you're done, configure your app as shown below, then click

**New** to create a new App Service plan for your app.

**Create App Service**  
Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
cephas\_lin@hotmail.com

Hosting Services

Web App Name Change Type  
cdnwebapp1

Subscription  
Visual Studio Ultimate with MSDN

Resource Group  
cdnwebapp

App Service Plan  
New...

Clicking the Create button will create the following Azure resources

App Service - cdnwebapp1

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.  
[Learn More](#)

Export... Create Cancel

4. Configure a new App Service plan in the dialog as shown below and click **OK**.

**Configure App Service Plan**  
An App Service plan is the container for your app. The App...

App Service Plan  
cdnwebapp

Location  
Central US

Size  
Free

OK Cancel

5. Click **Create** to create the web app.

**Create App Service**  
Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
cephas\_lin@hotmail.com

**Hosting** ⓘ  
**Services**

Web App Name:  [Change Type](#)

Subscription:

Resource Group:  ⓘ

App Service Plan:

Clicking the Create button will create the following Azure resources

App Service - cdnwebapp1  
App Service Plan - cdnwebapp

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.  
[Learn More](#)

6. Once your ASP.NET application is created, publish it to Azure in the Azure App Service Activity pane by clicking **Publish**  **to this Web App now**. Click **Publish** to complete the process.

Azure App Service Activity

Publish:

**Overall status**

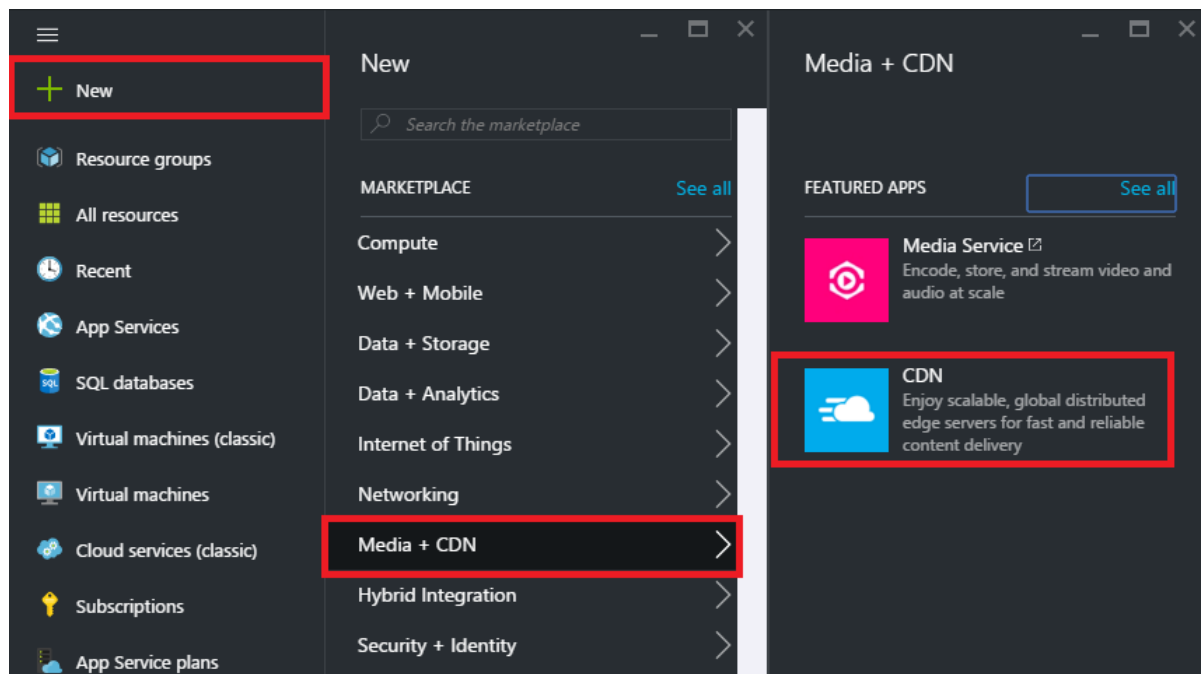
Publish profile added to the web project

**Publish cdnwebapp to this Web App now**

You will see your published web app in the browser when publishing is complete.

7. To create a CDN endpoint, log into the [Azure portal](#).
8. Click **+ New > Media + CDN > CDN**.





9. Specify the **CDN**, **Location**, **Resource group**, **Pricing tier**, then click **Create**

CDN profile

\*

Name

\*

Location ⓘ

Choose a location

>

\*

Resource group

Configure required settings

>

Or create new

\*

Pricing tier

Configure required settings

>

\*

Subscription

Pay-As-You-Go

>

☒ Pin to dashboard

Create

10. In the **CDN Profile** blade click on + **Endpoint** button. Give it a name, select **Web App** in the **Origin Type** dropdown and your web app in the **Origin hostname** dropdown, then click **Add**.

**cdnapp**  
CDN profile

Settings **Endpoint** Manage Purge Delete

Essentials

Resource group: [AzureAppService](#) Pricing tier: Standard

Status: Active

Location: West US

Subscription name: [Pay-As-You-Go](#)

Subscription ID: 61926bc1-4c0f-453b-8eb7-e3955e7d7f93

All settings →

Add tiles +

Endpoints

1

HOSTNAME	STATUS	PROTOCOL
cdnwebappendpoint.azure...	Running	HTTP, HTTPS

Add a group +

**Add an endpoint**

\* Name:  ✓

\* Origin type:

\* Origin hostname:

Origin path:

Origin host header:

Protocol: ☒ HTTP ☒ HTTPS

Origin port:

**Add**

[AZURE.NOTE] Once your CDN endpoint is created, the **Endpoint** blade will show you its CDN URL and the origin domain that it's integrated with. However, it can take a while for the new CDN endpoint's configuration to be fully propagated to all the CDN node locations.

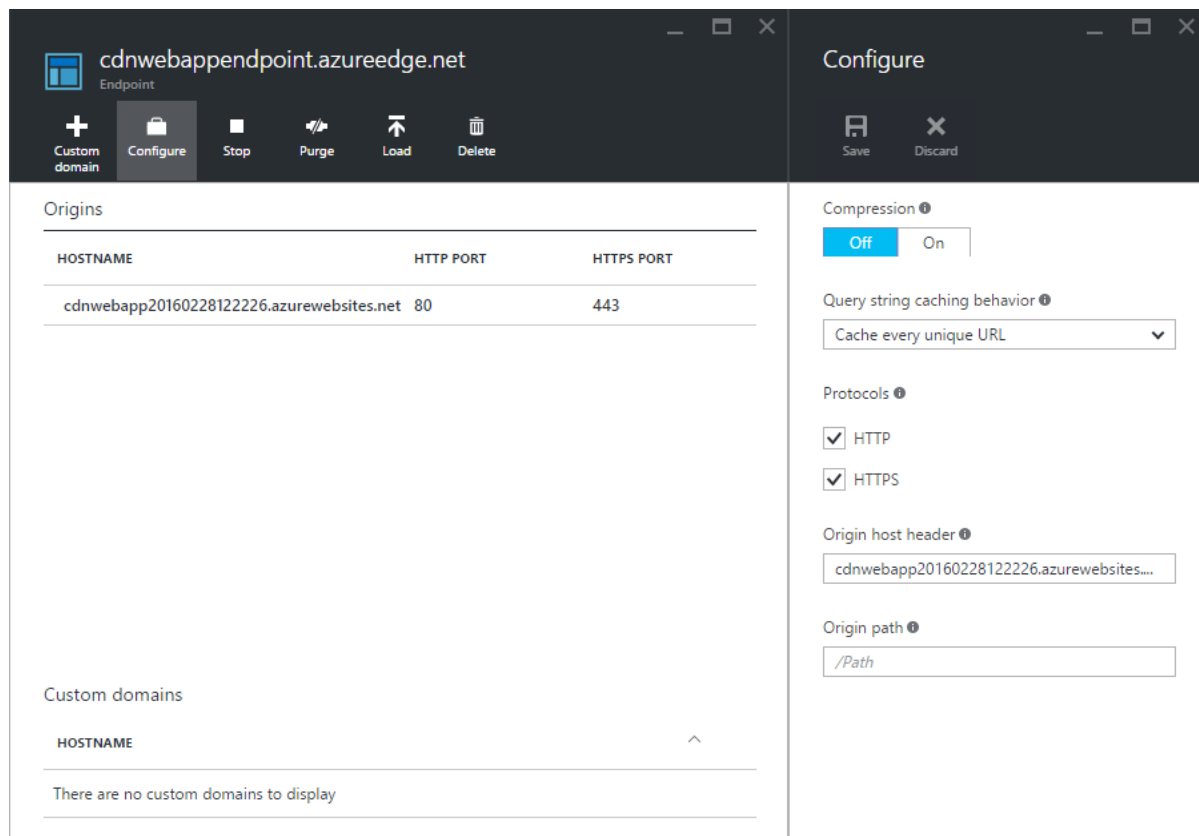
- Back in the **Endpoint** blade, click the name of the CDN endpoint you just created.

**Endpoints**

+ Endpoint

HOSTNAME	STATUS	PROTOCOL
cdnwebappendpoint.azure...	Running	HTTP, HTTPS

- Click the **Configure** button. In the **Configure** blade, select **Cache every unique URL** in **Query string caching behavior** dropdown, then click the **Save** button.

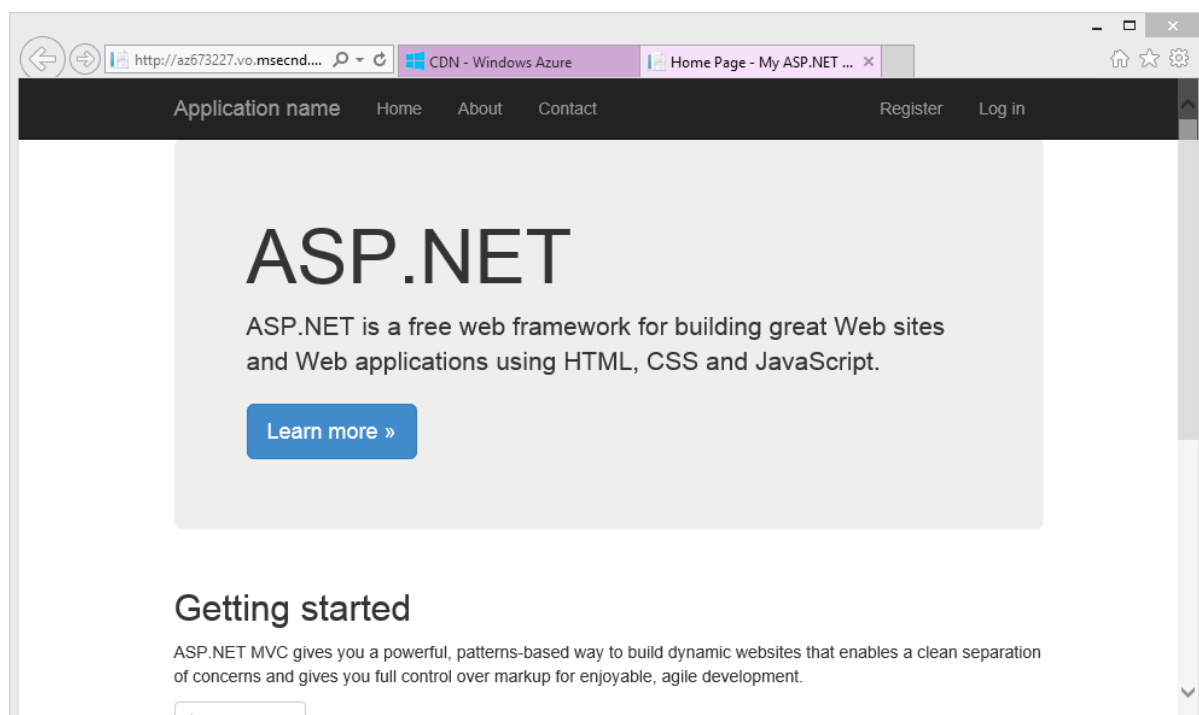


Once you enable this, the same link accessed with different query strings will be cached as separate entries.

#### NOTE

While enabling the query string is not necessary for this tutorial section, you want to do this as early as possible for convenience since any change here is going to take time to propagate to all the CDN nodes, and you don't want any non-query-string-enabled content to clog up the CDN cache (updating CDN content will be discussed later).

1. Now, navigate to the CDN endpoint address. If the endpoint is ready, you should see your web app displayed. If you get an **HTTP 404** error, the CDN endpoint is not ready. You may need to wait up to an hour for the CDN configuration to be propagated to all the edge nodes.



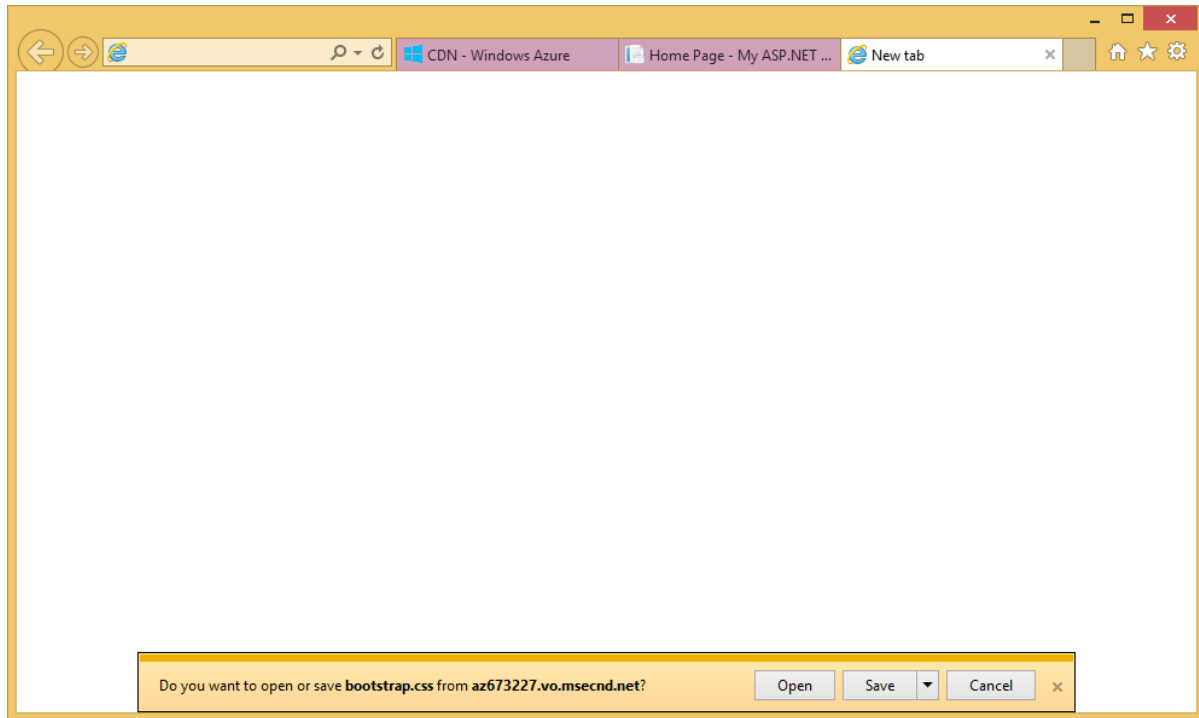
2. Next, try to access the `~/Content/bootstrap.css` file in your ASP.NET project. In the browser window, navigate to **`http://<cdnName>.azureedge.net/Content/bootstrap.css`**. In my setup, this URL is:

```
http://az673227.azureedge.net/Content/bootstrap.css
```

Which corresponds to the following origin URL at the CDN endpoint:

```
http://cdnwebapp.azurewebsites.net/Content/bootstrap.css
```

When you navigate to **`http://<cdnName>.azureedge.net/Content/bootstrap.css`**, you will be prompted to download the bootstrap.css that came from your web app in Azure.



You can similarly access any publicly accessible URL at **`http://<serviceName>.cloudapp.net/`**, straight from your CDN endpoint. For example:

- A .js file from the /Script path
- Any content file from the /Content path
- Any controller/action
- If the query string is enabled at your CDN endpoint, any URL with query strings
- The entire Azure web app, if all content is public

Note that it may not be always a good idea (or generally a good idea) to serve an entire Azure web app through Azure CDN. Some of the caveats are:

- This approach requires your entire site to be public, because Azure CDN cannot serve any private content.
- If the CDN endpoint goes offline for any reason, whether scheduled maintenance or user error, your entire web app goes offline unless the customers can be redirected to the origin URL

**`http://<sitenam>.azurewebsites.net/`**.

- Even with the custom Cache-Control settings (see [Configure caching options for static files in your Azure web app](#)), a CDN endpoint does not improve the performance of highly-dynamic content. If you tried to load the home page from your CDN endpoint as shown above, notice that it took at least 5 seconds to load the default home page the first time, which is a fairly simple page. Imagine what would happen to the client experience if this page contains dynamic content that must update every minute. Serving dynamic content from a CDN

endpoint requires short cache expiration, which translates to frequent cache misses at the CDN endpoint. This hurts the performance of your Azure web app and defeats the purpose of a CDN.

The alternative is to determine which content to serve from Azure CDN on a case-by-case basis in your Azure web app. To that end, you have already seen how to access individual content files from the CDN endpoint. I will show you how to serve a specific controller action through the CDN endpoint in [Serve content from controller actions through Azure CDN](#).

## Configure caching options for static files in your Azure web app

With Azure CDN integration in your Azure web app, you can specify how you want static content to be cached in the CDN endpoint. To do this, open *Web.config* from your ASP.NET project (e.g. **cdnwebapp**) and add a `<staticContent>` element to `<system.webServer>`. The XML below configures the cache to expire in 3 days.

```
<system.webServer>
  <staticContent>
    <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="3.00:00:00" />
  </staticContent>
  ...
</system.webServer>
```

Once you do this, all static files in your Azure web app will observe the same rule in your CDN cache. For more granular control of cache settings, add a *Web.config* file into a folder and add your settings there. For example, add a *Web.config* file to the `\Content` folder and replace the content with the following XML:

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <staticContent>
      <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="15.00:00:00" />
    </staticContent>
  </system.webServer>
</configuration>
```

This setting causes all static files from the `\Content` folder to be cached for 15 days.

For more information on how to configure the `<clientCache>` element, see [Client Cache <clientCache>](#).

In the next section, I will also show you how you can configure cache settings for controller action results in the CDN cache.

## Serve content from controller actions through Azure CDN

When you integrate Web Apps with Azure CDN, it is relatively easy to serve content from controller actions through the Azure CDN. Again, if you decide to serve the entire Azure web app through your CDN, you don't need to do this at all since all the controller actions are reachable through the CDN already. But for the reasons I already pointed out in [Deploy an Azure web app with an integrated CDN endpoint](#), you may decide against this and choose instead to select the controller action you want to serve from Azure CDN. [Maarten Balliauw](#) shows you how to do it with a fun MemeGenerator controller in [Reducing latency on the web with the Azure CDN](#). I will simply reproduce it here.

Suppose in your web app you want to generate memes based on a young Chuck Norris image (photo by [Alan Light](#)) like this:



You have a simple `Index` action that allows the customers to specify the superlatives in the image, then generates the meme once they post to the action. Since it's Chuck Norris, you would expect this page to become wildly popular globally. This is a good example of serving semi-dynamic content with Azure CDN.

Follow the steps above to setup this controller action:

1. In the `\Controllers` folder, create a new `.cs` file called `MemeGeneratorController.cs` and replace the content with the following code. Substitute your file path for `~/Content/chuck.bmp` and your CDN name for `yourCDNName`.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Net;
using System.Web.Hosting;
using System.Web.Mvc;
using System.Web.UI;

namespace cdnwebapp.Controllers
{
    public class MemeGeneratorController : Controller
    {
        static readonly Dictionary<string, Tuple<string, string>> Memes = new Dictionary<string,
        Tuple<string, string>>();

        public ActionResult Index()
        {
            return View();
        }

        [HttpPost, ActionName("Index")]
        public ActionResult Index_Post(string top, string bottom)
        {
            var identifier = Guid.NewGuid().ToString();
            if (!Memes.ContainsKey(identifier))
            {
                Memes.Add(identifier, new Tuple<string, string>(top, bottom));
            }

            return Content("<a href=\"" + Url.Action("Show", new {id = identifier}) + "\">here's your
```

```

meme</a>");
    }

    [OutputCache(VaryByParam = "*", Duration = 1, Location = OutputCacheLocation.Downstream)]
    public ActionResult Show(string id)
    {
        Tuple<string, string> data = null;
        if (!Memes.TryGetValue(id, out data))
        {
            return new HttpStatusCodeResult(HttpStatusCode.NotFound);
        }

        if (Debugger.IsAttached) // Preserve the debug experience
        {
            return Redirect(string.Format("/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1,
data.Item2));
        }
        else // Get content from Azure CDN
        {
            return Redirect(string.Format("http://<yourCDNName>.azureedge.net/MemeGenerator/Generate?top=
{0}&bottom={1}", data.Item1, data.Item2));
        }
    }

    [OutputCache(VaryByParam = "*", Duration = 3600, Location = OutputCacheLocation.Downstream)]
    public ActionResult Generate(string top, string bottom)
    {
        string imageFilePath = HostingEnvironment.MapPath("~/Content/chuck.bmp");
        Bitmap bitmap = (Bitmap)Image.FromFile(imageFilePath);

        using (Graphics graphics = Graphics.FromImage(bitmap))
        {
            SizeF size = new SizeF();
            using (Font arialFont = FindBestFitFont(bitmap, graphics, top.ToUpperInvariant(), new
Font("Arial Narrow", 100), out size))
            {
                graphics.DrawString(top.ToUpperInvariant(), arialFont, Brushes.White, new
PointF(((bitmap.Width - size.Width) / 2), 10f));
            }
            using (Font arialFont = FindBestFitFont(bitmap, graphics, bottom.ToUpperInvariant(), new
Font("Arial Narrow", 100), out size))
            {
                graphics.DrawString(bottom.ToUpperInvariant(), arialFont, Brushes.White, new
PointF(((bitmap.Width - size.Width) / 2), bitmap.Height - 10f - arialFont.Height));
            }
        }
        MemoryStream ms = new MemoryStream();
        bitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Png);
        return File(ms.ToArray(), "image/png");
    }

    private Font FindBestFitFont(Image i, Graphics g, String text, Font font, out SizeF size)
    {
        // Compute actual size, shrink if needed
        while (true)
        {
            size = g.MeasureString(text, font);

            // It fits, back out
            if (size.Height < i.Height &&
                size.Width < i.Width) { return font; }

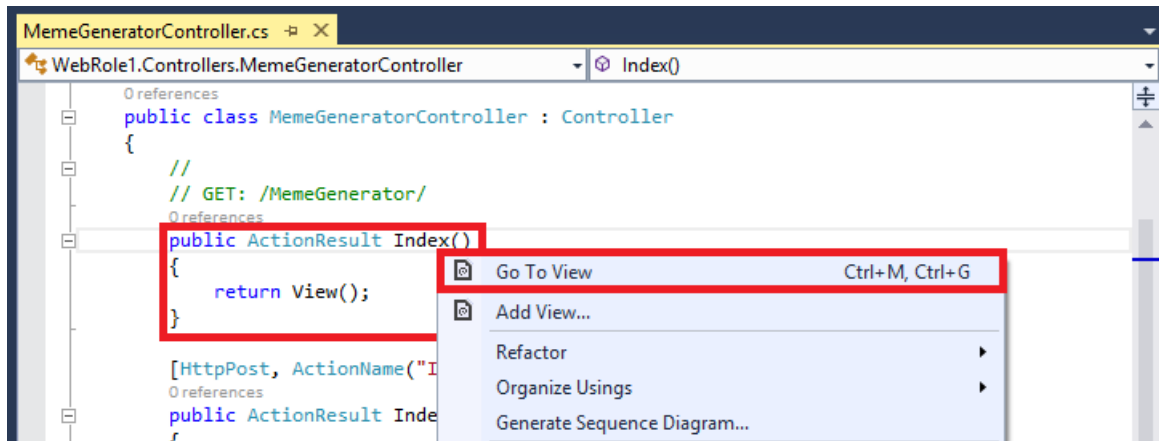
            // Try a smaller font (90% of old size)
            Font oldFont = font;
            font = new Font(font.Name, (float)(font.Size * .9), font.Style);
            oldFont.Dispose();
        }
    }
}
}

```

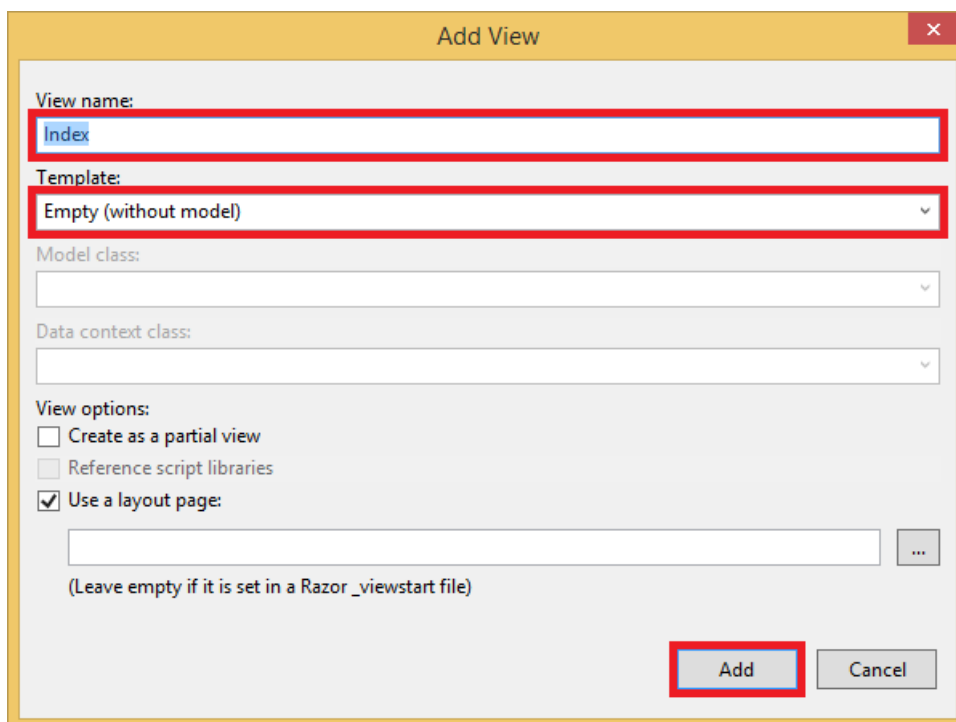


```
}
```

2. Right-click in the default `Index()` action and select **Add View**.



3. Accept the settings below and click **Add**.



4. Open the new `Views\MemeGenerator\Index.cshtml` and replace the content with the following simple HTML for submitting the superlatives:

```
<h2>Meme Generator</h2>

<form action="" method="post">
  <input type="text" name="top" placeholder="Enter top text here" />
  <br />
  <input type="text" name="bottom" placeholder="Enter bottom text here" />
  <br />
  <input class="btn" type="submit" value="Generate meme" />
</form>
```

5. Publish to the Azure web app again and navigate to **`http://<serviceName>.cloudapp.net/MemeGenerator/Index`** in your browser.

When you submit the form values to `/MemeGenerator/Index`, the `Index_Post` action method returns a link to the `Show` action method with the respective input identifier. When you click the link, you reach the following code:

```
[OutputCache(VaryByParam = "*", Duration = 1, Location = OutputCacheLocation.Downstream)]
public ActionResult Show(string id)
{
    Tuple<string, string> data = null;
    if (!Memes.TryGetValue(id, out data))
    {
        return new HttpStatusCodeResult(HttpStatusCode.NotFound);
    }

    if (Debugger.IsAttached) // Preserve the debug experience
    {
        return Redirect(string.Format("/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1, data.Item2));
    }
    else // Get content from Azure CDN
    {
        return Redirect(string.Format("http://<yourCDNName>.azureedge.net/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1, data.Item2));
    }
}
```

If your local debugger is attached, then you will get the regular debug experience with a local redirect. If it's running in the Azure web app, then it will redirect to:

```
http://<yourCDNName>.azureedge.net/MemeGenerator/Generate?top=<formInput>&bottom=<formInput>
```

Which corresponds to the following origin URL at your CDN endpoint:

```
http://<yourSiteName>.azurewebsites.net/cdn/MemeGenerator/Generate?top=<formInput>&bottom=<formInput>
```

After URL rewrite rule previously applied, the actual file that gets cached to your CDN endpoint is:

```
http://<yourSiteName>.azurewebsites.net/MemeGenerator/Generate?top=<formInput>&bottom=<formInput>
```

You can then use the `OutputCacheAttribute` attribute on the `Generate` method to specify how the action result should be cached, which Azure CDN will honor. The code below specifies a cache expiration of 1 hour (3,600 seconds).

```
[OutputCache(VaryByParam = "*", Duration = 3600, Location = OutputCacheLocation.Downstream)]
```

Likewise, you can serve up content from any controller action in your Azure web app through your Azure CDN, with the desired caching option.

In the next section, I will show you how to serve the bundled and minified scripts and CSS through Azure CDN.

## Integrate ASP.NET bundling and minification with Azure CDN

Scripts and CSS stylesheets change infrequently and are prime candidates for the Azure CDN cache. Serving the entire web app through your Azure CDN is the easiest way to integrate bundling and minification with Azure CDN. However, as you may elect against this approach for the reasons described in [Integrate an Azure CDN endpoint with your Azure web app and serve static content in your Web pages from Azure CDN](#), I will show you how to do it while preserving the desired developer experience of ASP.NET bundling and minification, such as:

- Great debug mode experience
- Streamlined deployment
- Immediate updates to clients for script/CSS version upgrades

- Fallback mechanism when your CDN endpoint fails
- Minimize code modification

In the ASP.NET project that you created in [Integrate an Azure CDN endpoint with your Azure web app and serve static content in your Web pages from Azure CDN](#), open *App\_Start\BundleConfig.cs* and take a look at the

`bundles.Add()` method calls.

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
        "~/Scripts/jquery-{version}.js"));
    ...
}
```

The first `bundles.Add()` statement adds a script bundle at the virtual directory `~/bundles/jquery`. Then, open *Views\Shared\\_Layout.cshtml* to see how the script bundle tag is rendered. You should be able to find the following line of Razor code:

```
@Scripts.Render("~/bundles/jquery")
```

When this Razor code is run in the Azure web app, it will render a `<script>` tag for the script bundle similar to the following:

```
<script src="/bundles/jquery?v=FVs3ACwOLIVInrA15sdzR2jrCDmVOWFbZMY6g6Q0u1E1"></script>
```

However, when it is run in Visual Studio by typing `F5`, it will render each script file in the bundle individually (in the case above, only one script file is in the bundle):

```
<script src="/Scripts/jquery-1.10.2.js"></script>
```

This enables you to debug the JavaScript code in your development environment while reducing concurrent client connections (bundling) and improving file download performance (minification) in production. It's a great feature to preserve with Azure CDN integration. Furthermore, since the rendered bundle already contains an automatically generated version string, you want to replicate that functionality so that whenever you update your jQuery version through NuGet, it can be updated at the client side as soon as possible.

Follow the steps below to integration ASP.NET bundling and minification with your CDN endpoint.

1. Back in *App\_Start\BundleConfig.cs*, modify the `bundles.Add()` methods to use a different [Bundle constructor](#), one that specifies a CDN address. To do this, replace the `RegisterBundles` method definition with the following code:

```

public static void RegisterBundles(BundleCollection bundles)
{
    bundles.UseCdn = true;
    var version = System.Reflection.Assembly.GetAssembly(typeof(Controllers.HomeController))
        .GetName().Version.ToString();
    var cdnUrl = "http://<yourCDNName>.azureedge.net/{0}?" + version;

    bundles.Add(new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery")).Include(
        "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval", string.Format(cdnUrl,
"bundles/jqueryval")).Include(
        "~/Scripts/jquery.validate*"));

    // Use the development version of Modernizr to develop with and learn from. Then, when you're
    // ready for production, use the build tool at http://modernizr.com to pick only the tests you need.
    bundles.Add(new ScriptBundle("~/bundles/modernizr", string.Format(cdnUrl,
"bundles/modernizr")).Include(
        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap", string.Format(cdnUrl,
"bundles/bootstrap")).Include(
        "~/Scripts/bootstrap.js",
        "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css")).Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));
}

```

Be sure to replace `<yourCDNName>` with the name of your Azure CDN.

In plain words, you are setting `bundles.UseCdn = true` and added a carefully crafted CDN URL to each bundle. For example, the first constructor in the code:

```
new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery"))
```

is the same as:

```
new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl,
"http://<yourCDNName>.azureedge.net/bundles/jquery?<W.X.Y.Z>"))
```

This constructor tells ASP.NET bundling and minification to render individual script files when debugged locally, but use the specified CDN address to access the script in question. However, note two important characteristics with this carefully crafted CDN URL:

- The origin for this CDN URL is `http://<yourSiteName>.azurewebsites.net/bundles/jquery?<W.X.Y.Z>`, which is actually the virtual directory of the script bundle in your Web application.
  - Since you are using CDN constructor, the CDN script tag for the bundle no longer contains the automatically generated version string in the rendered URL. You must manually generate a unique version string every time the script bundle is modified to force a cache miss at your Azure CDN. At the same time, this unique version string must remain constant through the life of the deployment to maximize cache hits at your Azure CDN after the bundle is deployed.
2. The query string `<W.X.Y.Z>` pulls from `Properties\AssemblyInfo.cs` in your ASP.NET project. You can have a deployment workflow that includes incrementing the assembly version every time you publish to Azure. Or, you can just modify `Properties\AssemblyInfo.cs` in your project to automatically increment the version string every time you build, using the wildcard character `'*'`. For example, change `AssemblyVersion` as shown below:

```
[assembly: AssemblyVersion("1.0.0.*")]
```

Any other strategy to streamline generating a unique string for the life of a deployment will work here.

3. Republish the ASP.NET application and access the home page.
4. View the HTML code for the page. You should be able to see the CDN URL rendered, with a unique version string every time you republish changes to your Azure web app. For example:

```
...  
<link href="http://az673227.azureedge.net/Content/css?1.0.0.25449" rel="stylesheet"/>  
<script src="http://az673227.azureedge.net/bundles/modernizer?1.0.0.25449"></script>  
...  
<script src="http://az673227.azureedge.net/bundles/jquery?1.0.0.25449"></script>  
<script src="http://az673227.azureedge.net/bundles/bootstrap?1.0.0.25449"></script>  
...
```

5. In Visual Studio, debug the ASP.NET application in Visual Studio by typing `F5`.
6. View the HTML code for the page. You will still see each script file individually rendered so that you can have a consistent debug experience in Visual Studio.

```
...  
<link href="/Content/bootstrap.css" rel="stylesheet"/>  
<link href="/Content/site.css" rel="stylesheet"/>  
<script src="/Scripts/modernizr-2.6.2.js"></script>  
...  
<script src="/Scripts/jquery-1.10.2.js"></script>  
<script src="/Scripts/bootstrap.js"></script>  
<script src="/Scripts/respond.js"></script>  
...
```

## Fallback mechanism for CDN URLs

When your Azure CDN endpoint fails for any reason, you want your Web page to be smart enough to access your origin Web server as the fallback option for loading JavaScript or Bootstrap. It's serious enough to lose images on your web app due to CDN unavailability, but much more severe to lose crucial page functionality provided by your scripts and stylesheets.

The [Bundle](#) class contains a property called [CdnFallbackExpression](#) that enables you to configure the fallback mechanism for CDN failure. To use this property, follow the steps below:

1. In your ASP.NET project, open *App\_Start\BundleConfig.cs*, where you added a CDN URL in each [Bundle constructor](#), and add `CdnFallbackExpression` code in four places as shown to add a fallback mechanism to the default bundles.

```

public static void RegisterBundles(BundleCollection bundles)
{
    var version = System.Reflection.Assembly.GetAssembly(typeof(BundleConfig))
        .GetName().Version.ToString();
    var cdnUrl = "http://cdnurl.azureedge.net/.../{0}?" + version;
    bundles.UseCdn = true;

    bundles.Add(new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery")))
        { CdnFallbackExpression = "window.jquery" }
        .Include("~/Scripts/jquery-{version}.js");

    bundles.Add(new ScriptBundle("~/bundles/jqueryval", string.Format(cdnUrl, "bundles/jqueryval")))
        { CdnFallbackExpression = "$.validator" }
        .Include("~/Scripts/jquery.validate*");

    // Use the development version of Modernizr to develop with and learn from. Then, when you're
    // ready for production, use the build tool at http://modernizr.com to pick only the tests you need.
    bundles.Add(new ScriptBundle("~/bundles/modernizr", string.Format(cdnUrl, "bundles/modernizer")))
        { CdnFallbackExpression = "window.Modernizr" }
        .Include("~/Scripts/modernizr-*");

    bundles.Add(new ScriptBundle("~/bundles/bootstrap", string.Format(cdnUrl, "bundles/bootstrap")))
        { CdnFallbackExpression = "$.fn.modal" }
        .Include(
            "~/Scripts/bootstrap.js",
            "~/Scripts/respond.js");

    bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css")).Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));
}

```

When `CdnFallbackExpression` is not null, script is injected into the HTML to test whether the bundle is loaded successfully and, if not, access the bundle directly from the origin Web server. This property needs to be set to a JavaScript expression that tests whether the respective CDN bundle is loaded properly. The expression needed to test each bundle differs according to the content. For the default bundles above:

- `window.jquery` is defined in `jquery-{version}.js`
- `$.validator` is defined in `jquery.validate.js`
- `window.Modernizr` is defined in `modernizer-{version}.js`
- `$.fn.modal` is defined in `bootstrap.js`

You might have noticed that I did not set `CdnFallbackExpression` for the `~/Content/css` bundle. This is because currently there is a [bug in System.Web.Optimization](#) that injects a `<script>` tag for the fallback CSS instead of the expected `<link>` tag.

There is, however, a good [Style Bundle Fallback](#) offered by [Ember Consulting Group](#).

2. To use the workaround for CSS, create a new `.cs` file in your ASP.NET project's `App_Start` folder called `StyleBundleExtensions.cs`, and replace its content with the [code from GitHub](#).
3. In `App_Start\StyleFundleExtensions.cs`, rename the namespace to your ASP.NET application's namespace (e.g. **cdnwebapp**).
4. Go back to `App_Start\BundleConfig.cs` and replace the last `bundles.Add` statement with the following code:

```

bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css"))
    .IncludeFallback("~/Content/css", "sr-only", "width", "1px")
    .Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));

```

This new extension method uses the same idea to inject script in the HTML to check the DOM for the a matching class name, rule name, and rule value defined in the CSS bundle, and falls back to the origin Web server if it fails to find the match.

5. Publish to your Azure web app again and access the home page.
6. View the HTML code for the page. You should find injected scripts similar to the following:

```
...
<link href="http://az673227.azureedge.net/Content/css?1.0.0.25474" rel="stylesheet"/>
<script>(function() {
    var loadFallback,
        len = document.styleSheets.length;
    for (var i = 0; i < len; i++) {
        var sheet = document.styleSheets[i];
        if (sheet.href.indexOf('http://az673227.azureedge.net/Content/css?1.0.0.25474') !== -1)
        {
            var meta = document.createElement('meta');
            meta.className = 'sr-only';
            document.head.appendChild(meta);
            var value = window.getComputedStyle(meta).getPropertyValue('width');
            document.head.removeChild(meta);
            if (value !== '1px') {
                document.write('<link href="/Content/css" rel="stylesheet" type="text/css" />');
            }
        }
    }
    return true;
})();||document.write('<script src="/Content/css"><\script>');</script>

<script src="http://az673227.azureedge.net/bundles/modernizer?1.0.0.25474"></script>
<script>(window.Modernizr)||document.write('<script src="/bundles/modernizr"><\script>');</script>
...
<script src="http://az673227.azureedge.net/bundles/jquery?1.0.0.25474"></script>
<script>(window.jquery)||document.write('<script src="/bundles/jquery"><\script>');</script>

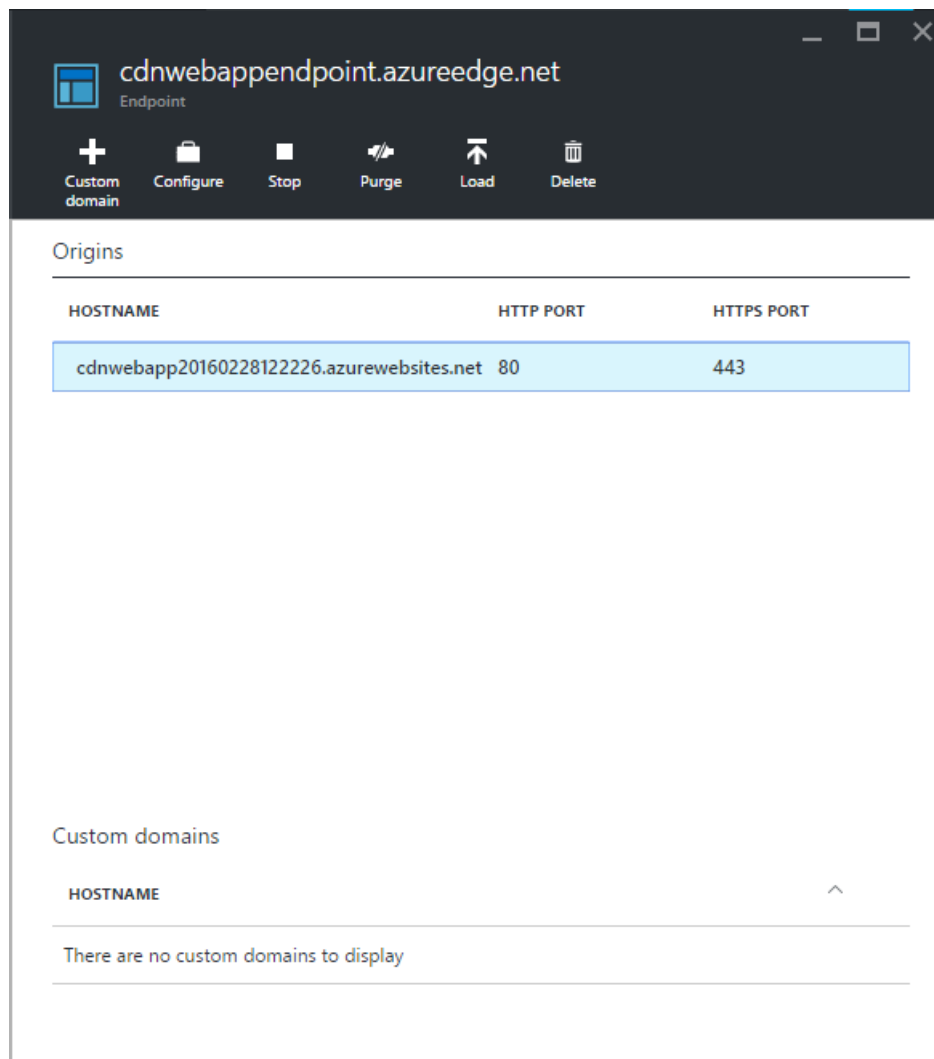
<script src="http://az673227.azureedge.net/bundles/bootstrap?1.0.0.25474"></script>
<script>($.fn.modal)||document.write('<script src="/bundles/bootstrap"><\script>');</script>
...
```

Note that injected script for the CSS bundle still contains the errant remnant from the `CdnFallbackExpression` property in the line:

```
})();||document.write('<script src="/Content/css"><\script>');</script>
```

But since the first part of the `||` expression will always return true (in the line directly above that), the `document.write()` function will never run.

7. To test whether the fallback script is working, go back to the your CDN endpoint's blade and click **Stop**.



8. Refresh your browser window for the Azure web app. You should now see that the all scripts and stylesheets are properly loaded.

## More Information

- [Overview of the Azure Content Delivery Network \(CDN\)](#)
- [Using Azure CDN](#)
- [Integrate a cloud service with Azure CDN](#)
- [ASP.NET Bundling and Minification](#)



# Integrate a cloud service with Azure CDN

1/24/2017 • 19 min to read • [Edit Online](#)

A cloud service can be integrated with Azure CDN, serving any content from the cloud service's location. This approach gives you the following advantages:

- Easily deploy and update images, scripts, and stylesheets in your cloud service's project directories
- Easily upgrade the NuGet packages in your cloud service, such as jQuery or Bootstrap versions
- Manage your Web application and your CDN-served content all from the same Visual Studio interface
- Unified deployment workflow for your Web application and your CDN-served content
- Integrate ASP.NET bundling and minification with Azure CDN

## What you will learn

In this tutorial, you will learn how to:

- [Integrate an Azure CDN endpoint with your cloud service and serve static content in your Web pages from Azure CDN](#)
- [Configure cache settings for static content in your cloud service](#)
- [Serve content from controller actions through Azure CDN](#)
- [Serve bundled and minified content through Azure CDN while preserving the script debugging experience in Visual Studio](#)
- [Configure fallback your scripts and CSS when your Azure CDN is offline](#)

## What you will build

You will deploy a cloud service Web role using the default ASP.NET MVC template, add code to serve content from an integrated Azure CDN, such as an image, controller action results, and the default JavaScript and CSS files, and also write code to configure the fallback mechanism for bundles served in the event that the CDN is offline.

## What you will need

This tutorial has the following prerequisites:

- An active [Microsoft Azure account](#)
- Visual Studio 2015 with [Azure SDK](#)

### NOTE

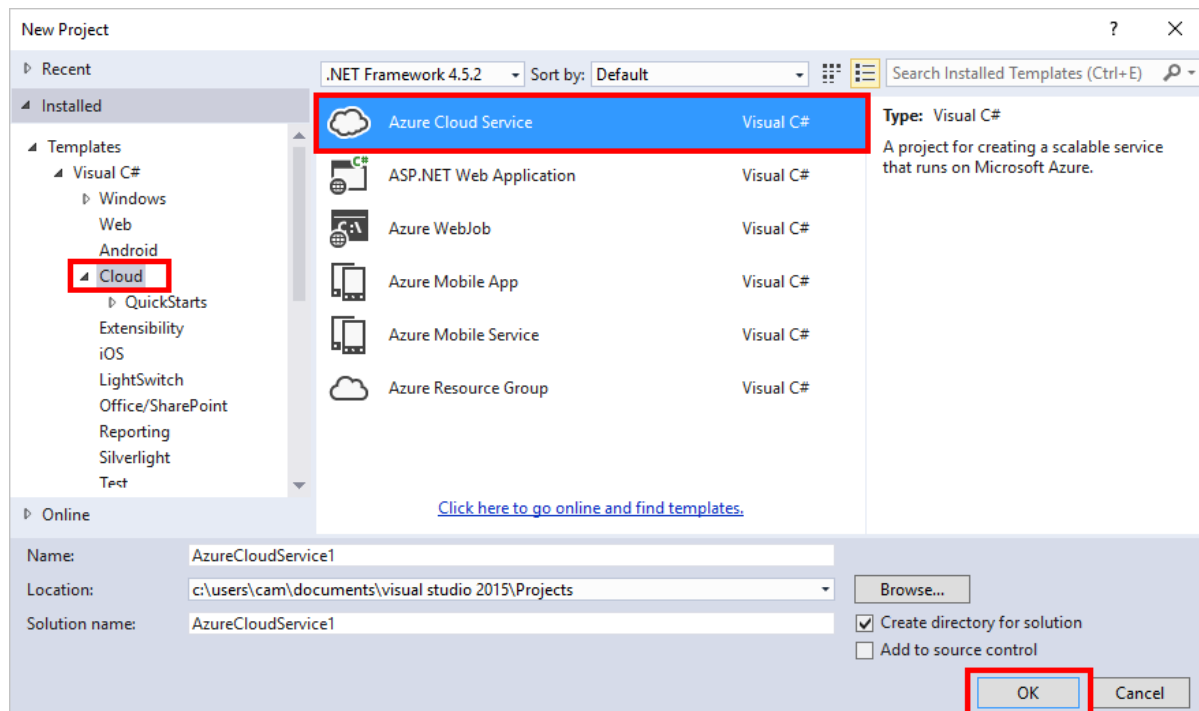
You need an Azure account to complete this tutorial:

- You can [open an Azure account for free](#) - You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services, such as Websites.
- You can [activate MSDN subscriber benefits](#) - Your MSDN subscription gives you credits every month that you can use for paid Azure services.

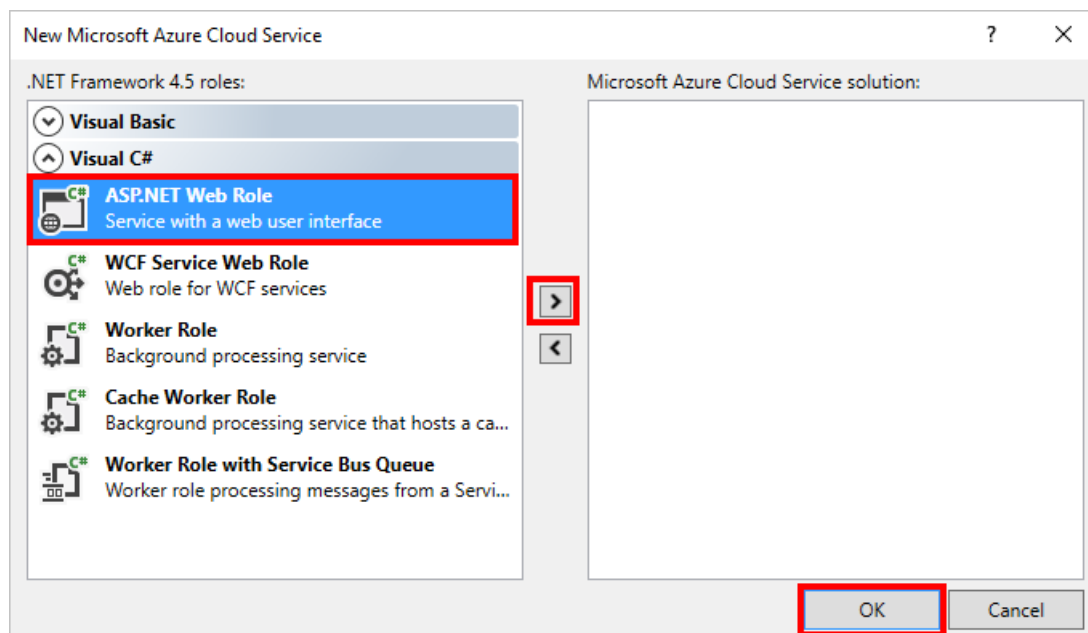
## Deploy a cloud service

In this section, you will deploy the default ASP.NET MVC application template in Visual Studio 2015 to a cloud service Web role, and then integrate it with a new CDN endpoint. Follow the instructions below:

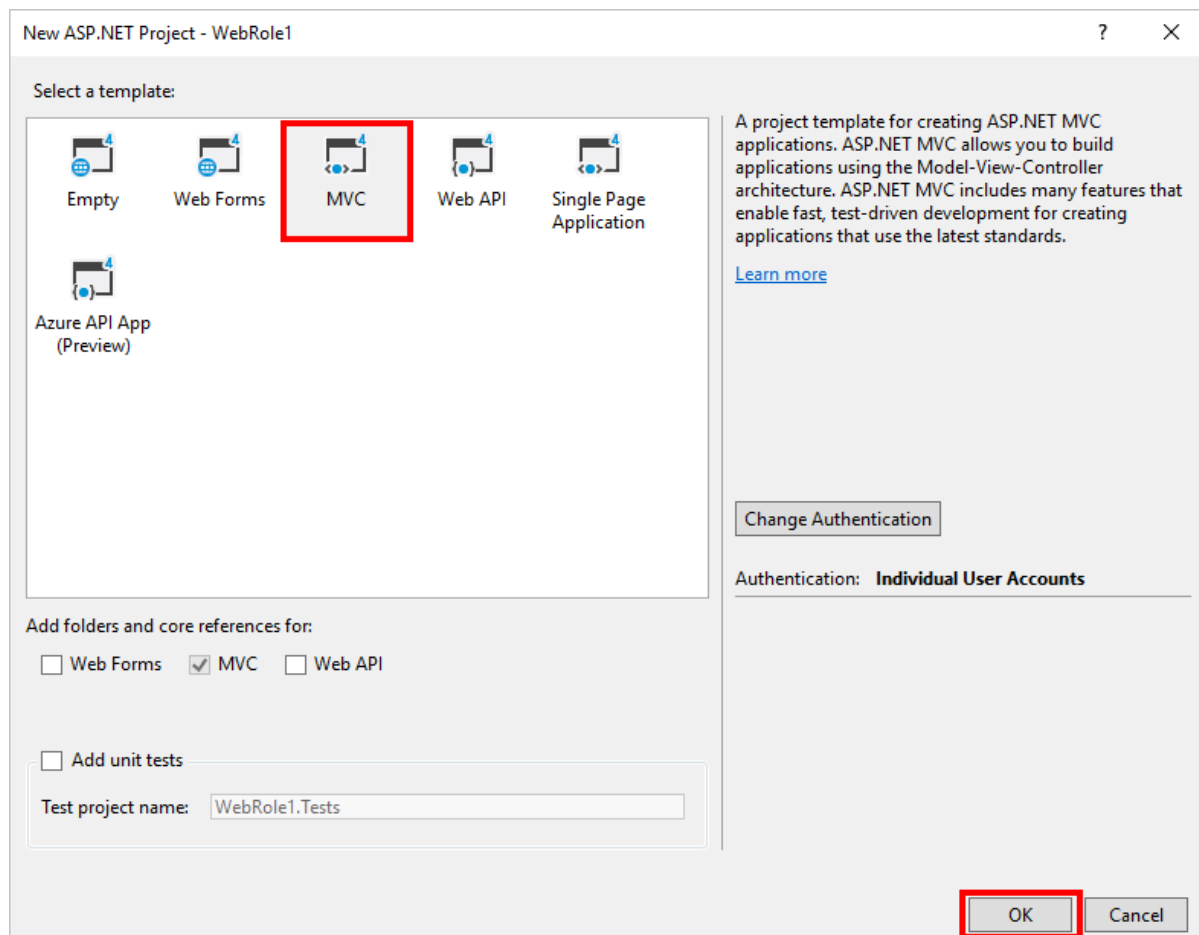
1. In Visual Studio 2015, create a new Azure cloud service from the menu bar by going to **File > New > Project > Cloud > Azure Cloud Service**. Give it a name and click **OK**.



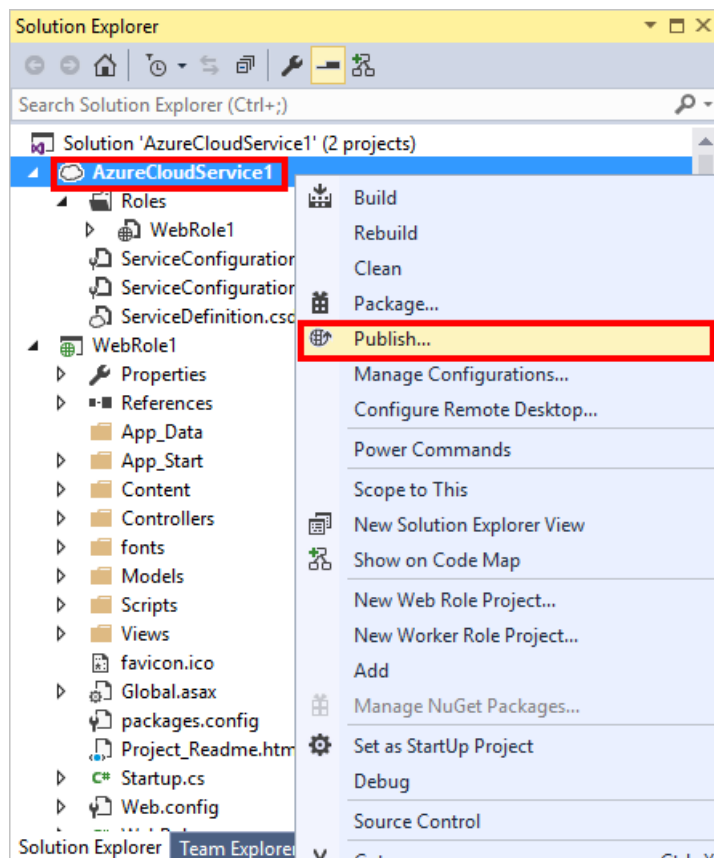
2. Select **ASP.NET Web Role** and click the > button. Click OK.



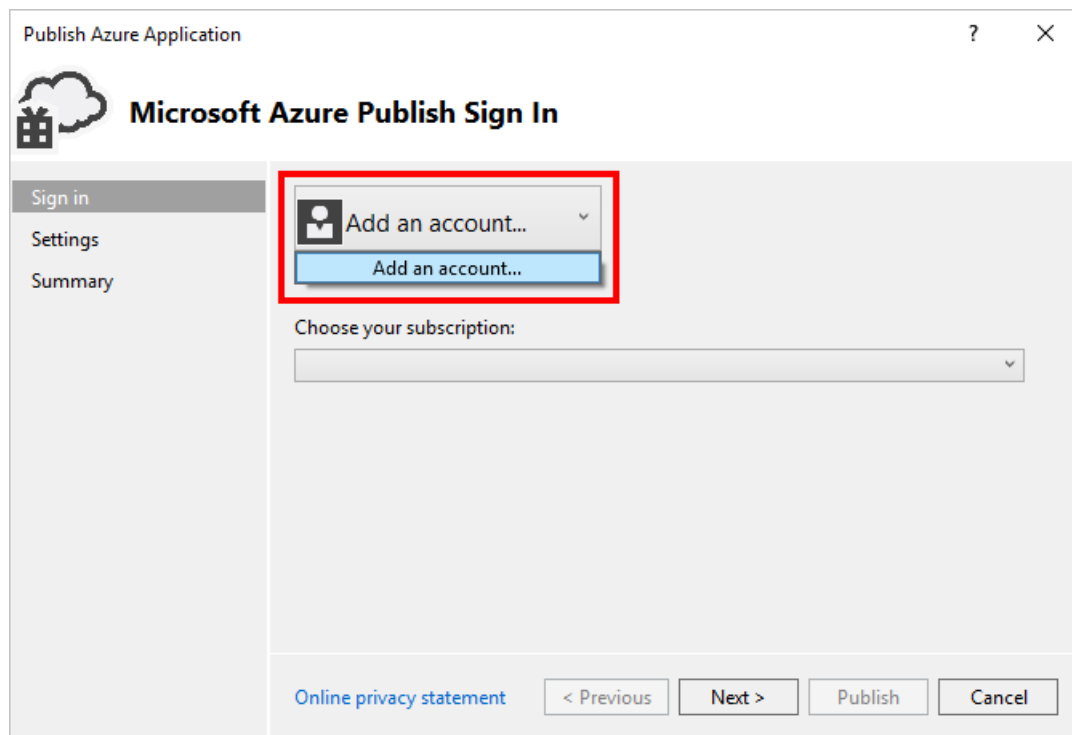
3. Select **MVC** and click **OK**.



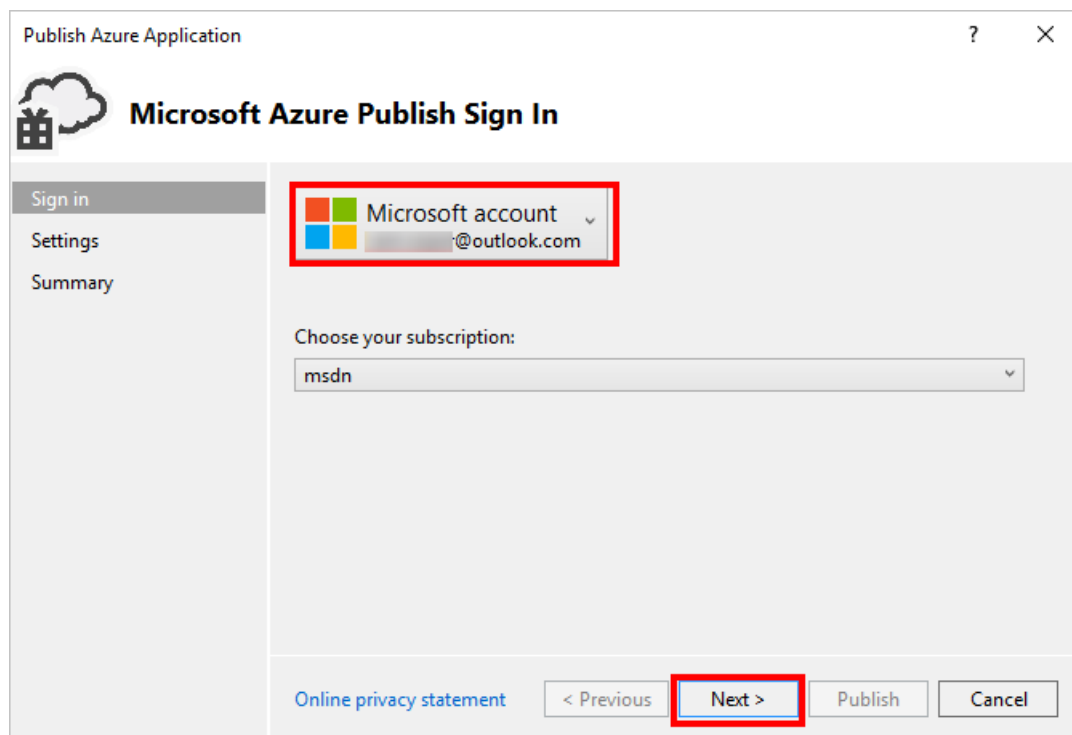
- Now, publish this Web role to an Azure cloud service. Right-click the cloud service project and select **Publish**.



- If you have not yet signed into Microsoft Azure, click the **Add an account...** dropdown and click the **Add an account** menu item.



6. In the sign-in page, sign in with the Microsoft account you used to activate your Azure account.
7. Once you're signed in, click **Next**.



8. Assuming that you haven't created a cloud service or storage account, Visual Studio will help you create both. In the **Create Cloud Service and Account** dialog, type the desired service name and select the desired region. Then, click **Create**.

Create Cloud Service and Storage Account

Microsoft account  
@outlook.com

Subscription:  
msdn

Name:  
camcdnservice

Region or Affinity Group:  
Central US

Replication:  
Geo-Redundant

[Read more about replication services and pricing details.](#)

Create Close

9. In the publish settings page, verify the configuration and click **Publish**.

Publish Azure Application

Microsoft Azure Publish Settings

Sign in  
Settings  
Summary

Common Settings Advanced Settings

Cloud Service:  
camcdnservice (Central US)

Environment:  
Production

Build configuration:  
Release

Service configuration:  
Cloud

☐ Enable Remote Desktop for all roles [Settings...](#)

☐ Enable Web Deploy for all web roles (requires Remote Desktop)

[Online privacy statement](#) < Previous Next > Publish Cancel

#### NOTE

The publishing process for cloud services takes a long time. The Enable Web Deploy for all roles option can make debugging your cloud service much quicker by providing fast (but temporary) updates to your Web roles. For more information on this option, see [Publishing a Cloud Service using the Azure Tools](#).

When the **Microsoft Azure Activity Log** shows that publishing status is **Completed**, you will create a CDN endpoint that's integrated with this cloud service.

#### WARNING

If, after publishing, the deployed cloud service displays an error screen, it's likely because the cloud service you've deployed is using a [guest OS](#) that does not include .NET 4.5.2. You can work around this issue by [deploying .NET 4.5.2](#) as a startup task.

# Create a new CDN profile

A CDN profile is a collection of CDN endpoints. Each profile contains one or more CDN endpoints. You may wish to use multiple profiles to organize your CDN endpoints by internet domain, web application, or some other criteria.

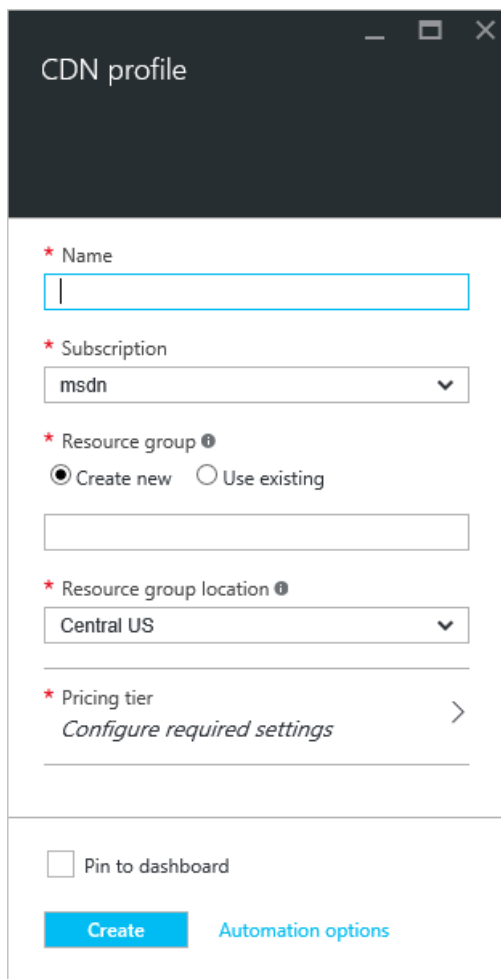
## TIP

If you already have a CDN profile that you want to use for this tutorial, proceed to [Create a new CDN endpoint](#).

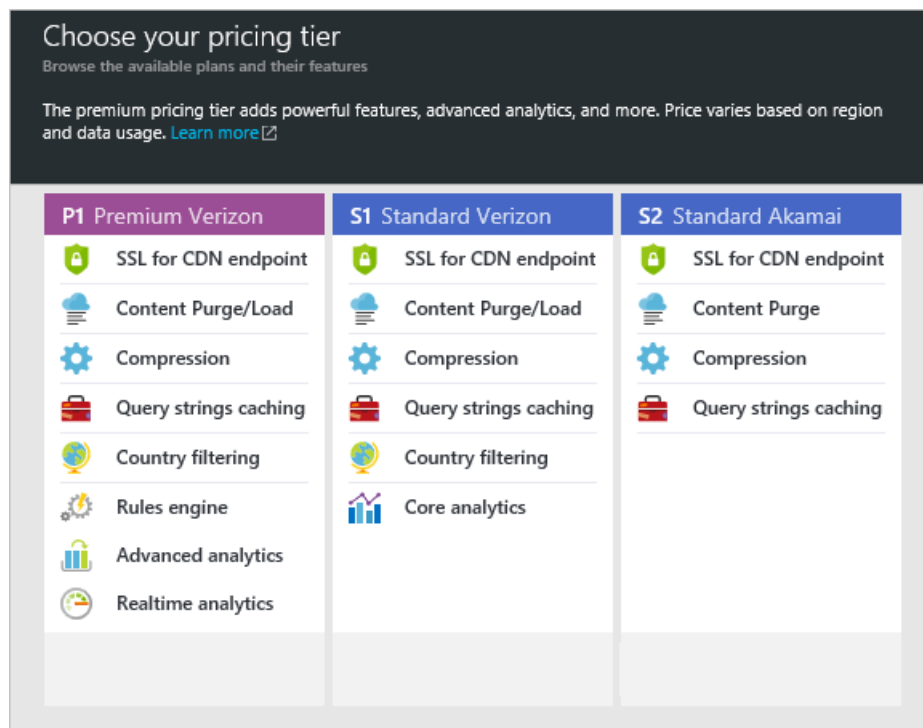
## To create a new CDN profile

1. In the [Azure Portal](#), in the upper left, click **New**. In the **New** blade, select **Web + Mobile**, then **CDN**.

The new CDN profile blade appears.



2. Enter a name for your CDN profile.
3. Select a **Location**. This is the Azure location where your CDN profile information will be stored. It has no impact on CDN endpoint locations.
4. Select or create a **Resource Group**. For more information on Resource Groups, see [Azure Resource Manager overview](#).
5. Select a **Pricing tier**. See the [CDN Overview](#) for a comparison of pricing tiers.



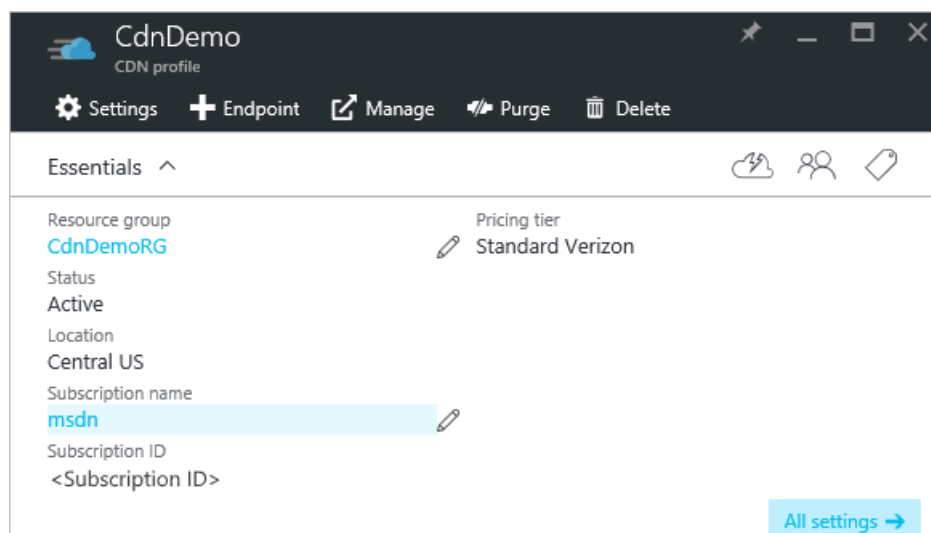
6. Select the **Subscription** for this CDN profile.
7. Click the **Create** button to create the new profile.

## Create a new CDN endpoint

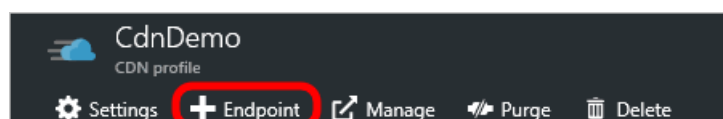
### To create a new CDN endpoint for your storage account

1. In the [Azure Management Portal](#), navigate to your CDN profile. You may have pinned it to the dashboard in the previous step. If you not, you can find it by clicking **Browse**, then **CDN profiles**, and clicking on the profile you plan to add your endpoint to.

The CDN profile blade appears.



2. Click the **Add Endpoint** button.



The **Add an endpoint** blade appears.

Add an endpoint

Allows configuring content delivery behavior and access.

\*

Name

.azureedge.net

\*

Origin type

\*

Origin hostname ⓘ

Origin path ⓘ

/Path

Origin host header ⓘ

Protocol ⓘ

☒ HTTP

☒ HTTPS



Origin port ⓘ

80

443

Add

- Enter a **Name** for this CDN endpoint. This name will be used to access your cached resources at the domain `<EndpointName>.azureedge.net`.
- In the **Origin type** dropdown, select *Cloud service*.
- In the **Origin hostname** dropdown, select your cloud service.
- Leave the defaults for **Origin path**, **Origin host header**, and **Protocol/Origin port**. You must specify at least one protocol (HTTP or HTTPS).
- Click the **Add** button to create the new endpoint.
- Once the endpoint is created, it appears in a list of endpoints for the profile. The list view shows the URL to use to access cached content, as well as the origin domain.

Endpoints		
1 		
HOSTNAME	STATUS	PROTOCOL
camservice.azureedge.net	 Running	HTTP, HTTPS



#### NOTE

The endpoint will not immediately be available for use. It can take up to 90 minutes for the registration to propagate through the CDN network. Users who try to use the CDN domain name immediately may receive status code 404 until the content is available via the CDN.

## Test the CDN endpoint

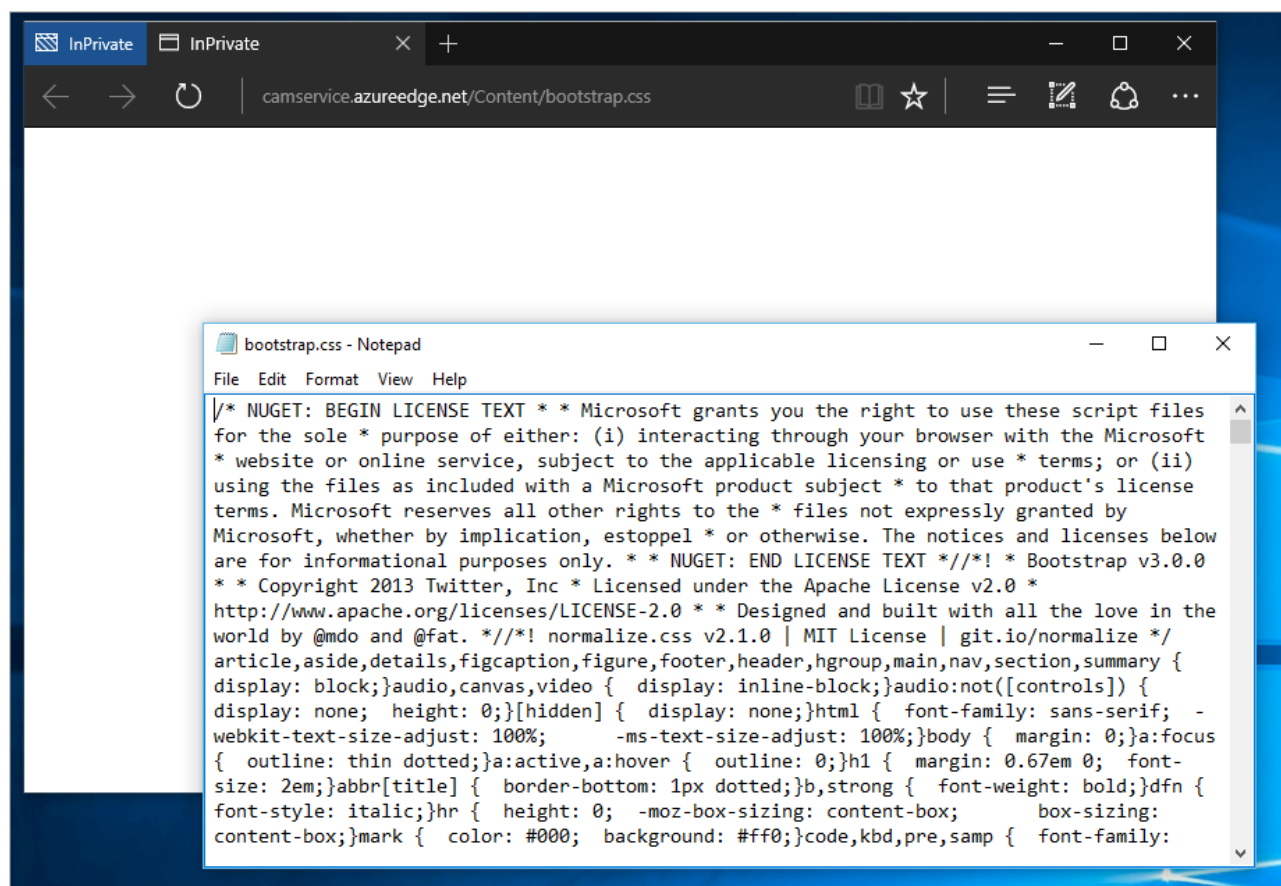
When the publishing status is **Completed**, open a browser window and navigate to **[http://\\*.azureedge.net/Content/bootstrap.css](http://*.azureedge.net/Content/bootstrap.css)**. In my setup, this URL is:

```
http://camservice.azureedge.net/Content/bootstrap.css
```

Which corresponds to the following origin URL at the CDN endpoint:

```
http://camcdnservice.cloudapp.net/Content/bootstrap.css
```

When you navigate to **<http://<cdnName>.azureedge.net/Content/bootstrap.css>**, depending on your browser, you will be prompted to download or open the bootstrap.css that came from your published Web app.

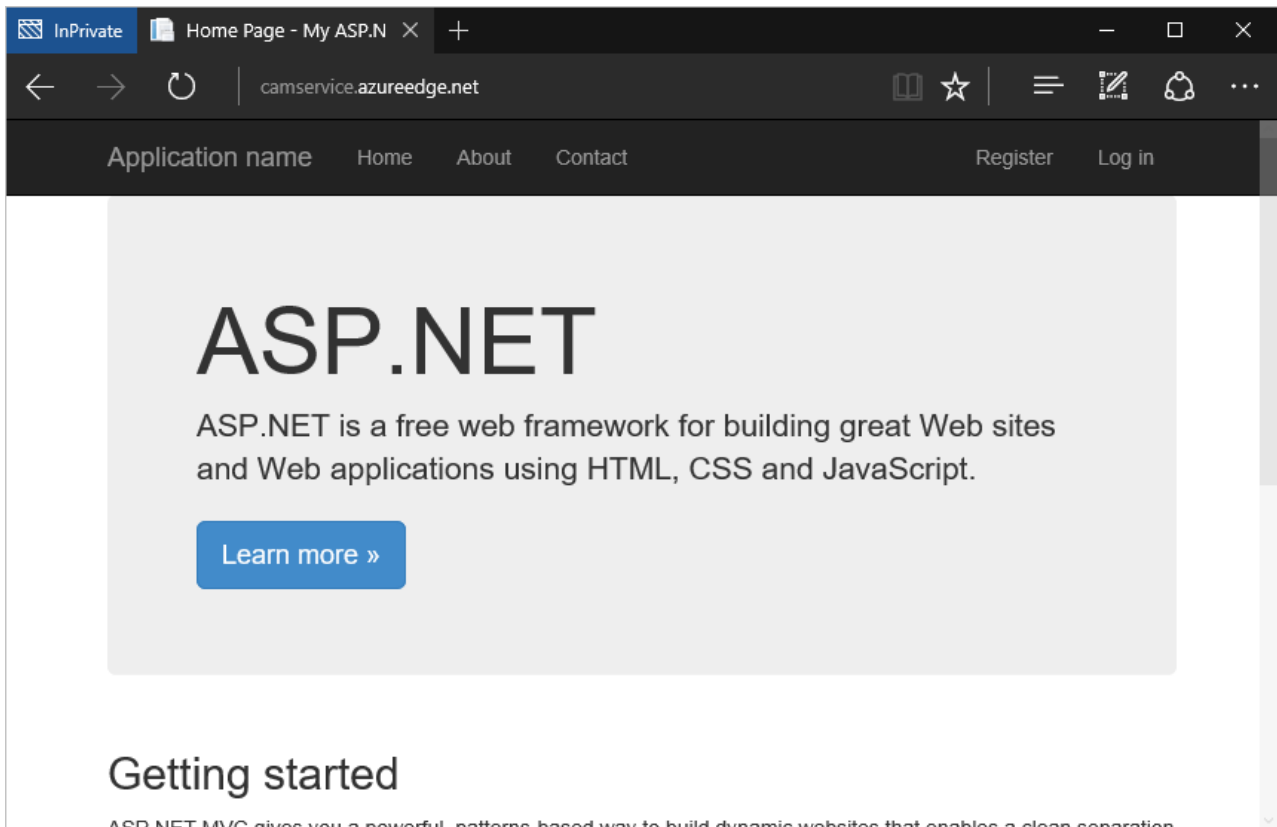


You can similarly access any publicly accessible URL at **<http://<serviceName>.cloudapp.net/>**, straight from your CDN endpoint. For example:

- A .js file from the /Script path
- Any content file from the /Content path
- Any controller/action
- If the query string is enabled at your CDN endpoint, any URL with query strings

In fact, with the above configuration, you can host the entire cloud service from

<http://<cdnName>.azureedge.net/>. If I navigate to <http://camservice.azureedge.net/>, I get the action result from Home/Index.



This does not mean, however, that it's always a good idea (or generally a good idea) to serve an entire cloud service through Azure CDN. Some of the caveats are:

- This approach requires your entire site to be public, because Azure CDN cannot serve any private content at this time.
- If the CDN endpoint goes offline for any reason, whether scheduled maintenance or user error, your entire cloud service goes offline unless the customers can be redirected to the origin URL  
**<http://<serviceName>.cloudapp.net/>.**
- Even with the custom Cache-Control settings (see [Configure caching options for static files in your cloud service](#)), a CDN endpoint does not improve the performance of highly-dynamic content. If you tried to load the home page from your CDN endpoint as shown above, notice that it took at least 5 seconds to load the default home page the first time, which is a fairly simple page. Imagine what would happen to the client experience if this page contains dynamic content that must update every minute. Serving dynamic content from a CDN endpoint requires short cache expiration, which translates to frequent cache misses at the CDN endpoint. This hurts the performance of your cloud service and defeats the purpose of a CDN.

The alternative is to determine which content to serve from Azure CDN on a case-by-case basis in your cloud service. To that end, you have already seen how to access individual content files from the CDN endpoint. I will show you how to serve a specific controller action through the CDN endpoint in [Serve content from controller actions through Azure CDN](#).

## Configure caching options for static files in your cloud service

With Azure CDN integration in your cloud service, you can specify how you want static content to be cached in the CDN endpoint. To do this, open *Web.config* from your Web role project (e.g. WebRole1) and add a

`<staticContent>` element to `<system.webServer>`. The XML below configures the cache to expire in 3 days.

```
<system.webServer>
  <staticContent>
    <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="3.00:00:00"/>
  </staticContent>
  ...
</system.webServer>
```

Once you do this, all static files in your cloud service will observe the same rule in your CDN cache. For more granular control of cache settings, add a *Web.config* file into a folder and add your settings there. For example, add a *Web.config* file to the `\Content` folder and replace the content with the following XML:

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <staticContent>
      <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="15.00:00:00"/>
    </staticContent>
  </system.webServer>
</configuration>
```

This setting causes all static files from the `\Content` folder to be cached for 15 days.

For more information on how to configure the `<clientCache>` element, see [Client Cache <clientCache>](#).

In [Serve content from controller actions through Azure CDN](#), I will also show you how you can configure cache settings for controller action results in the CDN cache.

## Serve content from controller actions through Azure CDN

When you integrate a cloud service Web role with Azure CDN, it is relatively easy to serve content from controller actions through the Azure CDN. Other than serving your cloud service directly through Azure CDN (demonstrated above), [Maarten Balliauw](#) shows you how to do it with a fun MemeGenerator controller in [Reducing latency on the web with the Azure CDN](#). I will simply reproduce it here.

Suppose in your cloud service you want to generate memes based on a young Chuck Norris image (photo by [Alan Light](#)) like this:



You have a simple `Index` action that allows the customers to specify the superlatives in the image, then generates the meme once they post to the action. Since it's Chuck Norris, you would expect this page to become wildly popular globally. This is a good example of serving semi-dynamic content with Azure CDN.

Follow the steps above to setup this controller action:

1. In the `\Controllers` folder, create a new `.cs` file called `MemeGeneratorController.cs` and replace the content with the following code. Be sure to replace the highlighted portion with your CDN name.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Net;
using System.Web.Hosting;
using System.Web.Mvc;
using System.Web.UI;

namespace WebRole1.Controllers
{
    public class MemeGeneratorController : Controller
    {
        static readonly Dictionary<string, Tuple<string, string>> Memes = new Dictionary<string,
        Tuple<string, string>>>();

        public ActionResult Index()
        {
            return View();
        }

        [HttpPost, ActionName("Index")]
        public ActionResult Index_Post(string top, string bottom)
        {
            var identifier = Guid.NewGuid().ToString();
            if (!Memes.ContainsKey(identifier))
            {
                Memes.Add(identifier, new Tuple<string, string>(top, bottom));
            }

            return Content("<a href=\"" + Url.Action("Show", new {id = identifier}) + "\">here's your
            meme</a>");
        }

        [OutputCache(VaryByParam = "*", Duration = 1, Location = OutputCacheLocation.Downstream)]
        public ActionResult Show(string id)
        {
            Tuple<string, string> data = null;
            if (!Memes.TryGetValue(id, out data))
            {
                return new HttpStatusCodeResult(HttpStatusCode.NotFound);
            }

            if (Debugger.IsAttached) // Preserve the debug experience
            {
                return Redirect(string.Format("/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1,
                data.Item2));
            }
            else // Get content from Azure CDN
            {
                return
                Redirect(string.Format("http://<yourCdnName>.azureedge.net/MemeGenerator/Generate?top={0}&bottom={1}",
                data.Item1, data.Item2));
            }
        }

        [OutputCache(VaryByParam = "*", Duration = 3600, Location = OutputCacheLocation.Downstream)]
    }
}
```

```

[HttpPost("{top}/{bottom}")]
public ActionResult Generate(string top, string bottom)
{
    string imageFilePath = HostingEnvironment.MapPath("~/Content/chuck.bmp");
    Bitmap bitmap = (Bitmap)Image.FromFile(imageFilePath);

    using (Graphics graphics = Graphics.FromImage(bitmap))
    {
        SizeF size = new SizeF();
        using (Font arialFont = FindBestFitFont(bitmap, graphics, top.ToUpperInvariant(), new
Font("Arial Narrow", 100), out size))
        {
            graphics.DrawString(top.ToUpperInvariant(), arialFont, Brushes.White, new
PointF(((bitmap.Width - size.Width) / 2), 10f));
        }
        using (Font arialFont = FindBestFitFont(bitmap, graphics, bottom.ToUpperInvariant(),
new Font("Arial Narrow", 100), out size))
        {
            graphics.DrawString(bottom.ToUpperInvariant(), arialFont, Brushes.White, new
PointF(((bitmap.Width - size.Width) / 2), bitmap.Height - 10f - arialFont.Height));
        }
    }

    MemoryStream ms = new MemoryStream();
    bitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Png);
    return File(ms.ToArray(), "image/png");
}

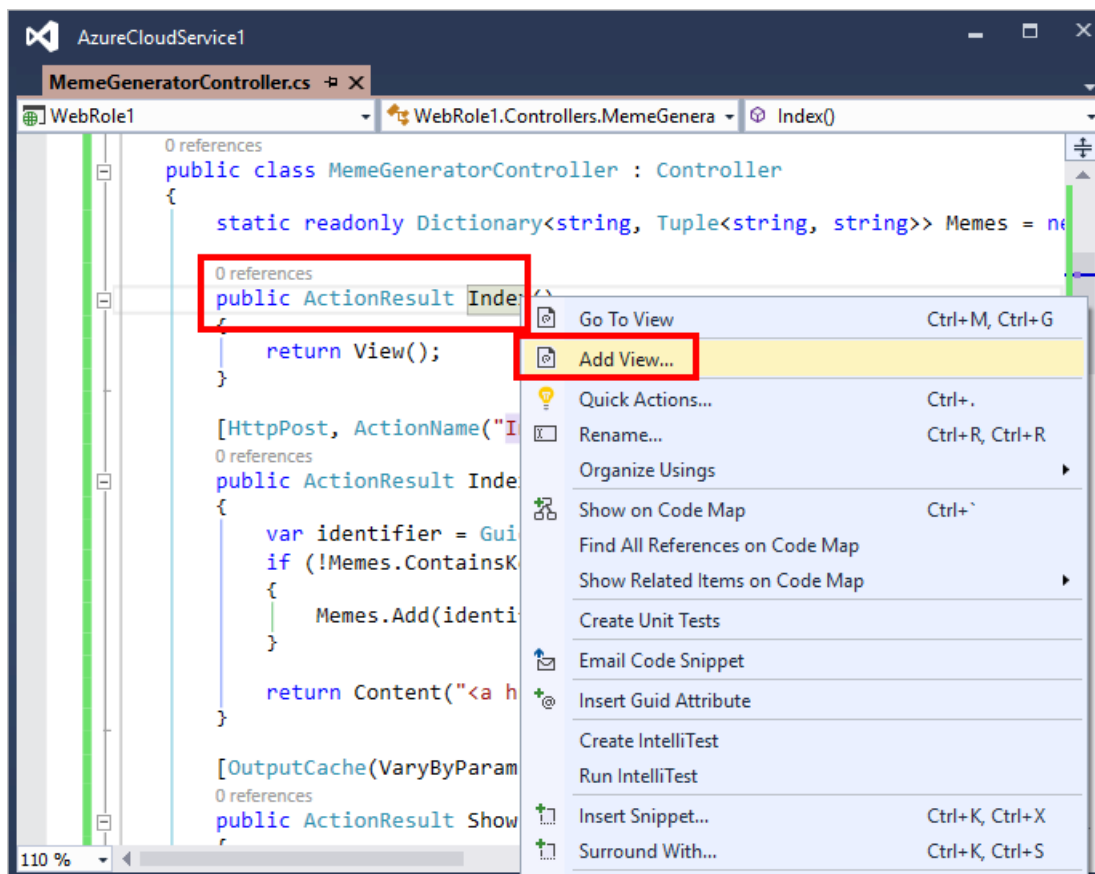
private Font FindBestFitFont(Image i, Graphics g, String text, Font font, out SizeF size)
{
    // Compute actual size, shrink if needed
    while (true)
    {
        size = g.MeasureString(text, font);

        // It fits, back out
        if (size.Height < i.Height &&
            size.Width < i.Width) { return font; }

        // Try a smaller font (90% of old size)
        Font oldFont = font;
        font = new Font(font.Name, (float)(font.Size * .9), font.Style);
        oldFont.Dispose();
    }
}
}
}

```

2. Right-click in the default `Index()` action and select **Add View**.



3. Accept the settings below and click **Add**.

The 'Add View' dialog box is shown. The 'View name' field contains 'Index'. The 'Template' dropdown is set to 'Empty (without model)'. The 'Model class' and 'Data context class' fields are empty. Under the 'Options' section, the checkboxes for 'Create as a partial view', 'Reference script libraries', and 'Use a layout page:' are all checked. The 'Add' button at the bottom right is highlighted with a red box.

4. Open the new `Views\MemeGenerator\Index.cshtml` and replace the content with the following simple HTML for submitting the superlatives:

```
<h2>Meme Generator</h2>

<form action="" method="post">
  <input type="text" name="top" placeholder="Enter top text here" />
  <br />
  <input type="text" name="bottom" placeholder="Enter bottom text here" />
  <br />
  <input class="btn" type="submit" value="Generate meme" />
</form>
```

5. Publish the cloud service again and navigate to

**http://<serviceName>.cloudapp.net/MemeGenerator/Index** in your browser.

When you submit the form values to `/MemeGenerator/Index`, the `Index_Post` action method returns a link to the `Show` action method with the respective input identifier. When you click the link, you reach the following code:

```
[OutputCache(VaryByParam = "*", Duration = 1, Location = OutputCacheLocation.Downstream)]
public ActionResult Show(string id)
{
    Tuple<string, string> data = null;
    if (!Memes.TryGetValue(id, out data))
    {
        return new HttpStatusCodeResult(HttpStatusCode.NotFound);
    }

    if (Debugger.IsAttached) // Preserve the debug experience
    {
        return Redirect(string.Format("/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1, data.Item2));
    }
    else // Get content from Azure CDN
    {
        return Redirect(string.Format("http://<yourCDNName>.azureedge.net/MemeGenerator/Generate?top={0}&bottom={1}", data.Item1, data.Item2));
    }
}
```

If your local debugger is attached, then you will get the regular debug experience with a local redirect. If it's running in the cloud service, then it will redirect to:

```
http://<yourCDNName>.azureedge.net/MemeGenerator/Generate?top=<formInput>&bottom=<formInput>
```

Which corresponds to the following origin URL at your CDN endpoint:

```
http://<youCloudServiceName>.cloudapp.net/MemeGenerator/Generate?top=<formInput>&bottom=<formInput>
```

You can then use the `OutputCacheAttribute` attribute on the `Generate` method to specify how the action result should be cached, which Azure CDN will honor. The code below specifies a cache expiration of 1 hour (3,600 seconds).

```
[OutputCache(VaryByParam = "*", Duration = 3600, Location = OutputCacheLocation.Downstream)]
```

Likewise, you can serve up content from any controller action in your cloud service through your Azure CDN, with the desired caching option.

In the next section, I will show you how to serve the bundled and minified scripts and CSS through Azure CDN.

## Integrate ASP.NET bundling and minification with Azure CDN

Scripts and CSS stylesheets change infrequently and are prime candidates for the Azure CDN cache. Serving the entire Web role through your Azure CDN is the easiest way to integrate bundling and minification with Azure CDN. However, as you may not want to do this, I will show you how to do it while preserving the desired developer experience of ASP.NET bundling and minification, such as:

- Great debug mode experience
- Streamlined deployment
- Immediate updates to clients for script/CSS version upgrades
- Fallback mechanism when your CDN endpoint fails

- Minimize code modification

In the **WebRole1** project that you created in [Integrate an Azure CDN endpoint with your Azure website and serve static content in your Web pages from Azure CDN](#), open *App\_Start\BundleConfig.cs* and take a look at the `bundles.Add()` method calls.

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
        "~/Scripts/jquery-{version}.js"));
    ...
}
```

The first `bundles.Add()` statement adds a script bundle at the virtual directory `~/bundles/jquery`. Then, open *Views\Shared\\_Layout.cshtml* to see how the script bundle tag is rendered. You should be able to find the following line of Razor code:

```
@Scripts.Render("~/bundles/jquery")
```

When this Razor code is run in the Azure Web role, it will render a `<script>` tag for the script bundle similar to the following:

```
<script src="/bundles/jquery?v=FVs3ACwOLIVInrA15sdzR2jrCDmVOWFbZMY6g6Q0u1E1"></script>
```

However, when it is run in Visual Studio by typing `F5`, it will render each script file in the bundle individually (in the case above, only one script file is in the bundle):

```
<script src="/Scripts/jquery-1.10.2.js"></script>
```

This enables you to debug the JavaScript code in your development environment while reducing concurrent client connections (bundling) and improving file download performance (minification) in production. It's a great feature to preserve with Azure CDN integration. Furthermore, since the rendered bundle already contains an automatically generated version string, you want to replicate that functionality so the whenever you update your jQuery version through NuGet, it can be updated at the client side as soon as possible.

Follow the steps below to integration ASP.NET bundling and minification with your CDN endpoint.

1. Back in *App\_Start\BundleConfig.cs*, modify the `bundles.Add()` methods to use a different [Bundle constructor](#), one that specifies a CDN address. To do this, replace the `RegisterBundles` method definition with the following code:



```

public static void RegisterBundles(BundleCollection bundles)
{
    bundles.UseCdn = true;
    var version = System.Reflection.Assembly.GetAssembly(typeof(Controllers.HomeController))
        .GetName().Version.ToString();
    var cdnUrl = "http://<yourCDNName>.azureedge.net/{0}?v=" + version;

    bundles.Add(new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery")).Include(
        "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval", string.Format(cdnUrl,
"bundles/jqueryval")).Include(
        "~/Scripts/jquery.validate*"));

    // Use the development version of Modernizr to develop with and learn from. Then, when you're
    // ready for production, use the build tool at http://modernizr.com to pick only the tests you
    need.
    bundles.Add(new ScriptBundle("~/bundles/modernizr", string.Format(cdnUrl,
"bundles/modernizer")).Include(
        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap", string.Format(cdnUrl,
"bundles/bootstrap")).Include(
        "~/Scripts/bootstrap.js",
        "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css")).Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));
}

```

Be sure to replace `<yourCDNName>` with the name of your Azure CDN.

In plain words, you are setting `bundles.UseCdn = true` and added a carefully crafted CDN URL to each bundle. For example, the first constructor in the code:

```
new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery"))
```

is the same as:

```
new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl,
"http://<yourCDNName>.azureedge.net/bundles/jquery?v=<W.X.Y.Z>"))
```

This constructor tells ASP.NET bundling and minification to render individual script files when debugged locally, but use the specified CDN address to access the script in question. However, note two important characteristics with this carefully crafted CDN URL:

- The origin for this CDN URL is `http://<yourCloudService>.cloudapp.net/bundles/jquery?v=<W.X.Y.Z>`, which is actually the virtual directory of the script bundle in your cloud service.
- Since you are using CDN constructor, the CDN script tag for the bundle no longer contains the automatically generated version string in the rendered URL. You must manually generate a unique version string every time the script bundle is modified to force a cache miss at your Azure CDN. At the same time, this unique version string must remain constant through the life of the deployment to maximize cache hits at your Azure CDN after the bundle is deployed.
- The query string `v=` pulls from `Properties\AssemblyInfo.cs` in your Web role project. You can have a deployment workflow that includes incrementing the assembly version every time you publish to Azure. Or, you can just modify `Properties\AssemblyInfo.cs` in your project to automatically increment the version string every time you build, using the wildcard character `'*'`. For example:

```
[assembly: AssemblyVersion("1.0.0.*")]
```

Any other strategy to streamline generating a unique string for the life of a deployment will work here.

2. Republish the cloud service and access the home page.
3. View the HTML code for the page. You should be able to see the CDN URL rendered, with a unique version string every time you republish changes to your cloud service. For example:

```
...  
  
<link href="http://camservice.azureedge.net/Content/css?v=1.0.0.25449" rel="stylesheet"/>  
  
<script src="http://camservice.azureedge.net/bundles/modernizer?v=1.0.0.25449"></script>  
  
...  
  
<script src="http://camservice.azureedge.net/bundles/jquery?v=1.0.0.25449"></script>  
  
<script src="http://camservice.azureedge.net/bundles/bootstrap?v=1.0.0.25449"></script>  
  
...
```

4. In Visual Studio, debug the cloud service in Visual Studio by typing `F5`.
5. View the HTML code for the page. You will still see each script file individually rendered so that you can have a consistent debug experience in Visual Studio.

```
...  
  
<link href="/Content/bootstrap.css" rel="stylesheet"/>  
<link href="/Content/site.css" rel="stylesheet"/>  
  
<script src="/Scripts/modernizr-2.6.2.js"></script>  
  
...  
  
<script src="/Scripts/jquery-1.10.2.js"></script>  
  
<script src="/Scripts/bootstrap.js"></script>  
<script src="/Scripts/respond.js"></script>  
  
...
```

## Fallback mechanism for CDN URLs

When your Azure CDN endpoint fails for any reason, you want your Web page to be smart enough to access your origin Web server as the fallback option for loading JavaScript or Bootstrap. It's serious enough to lose images on your website due to CDN unavailability, but much more severe to lose crucial page functionality provided by your scripts and stylesheets.

The [Bundle](#) class contains a property called [CdnFallbackExpression](#) that enables you to configure the fallback mechanism for CDN failure. To use this property, follow the steps below:

1. In your Web role project, open *App\_Start\BundleConfig.cs*, where you added a CDN URL in each [Bundle constructor](#), and make the following highlighted changes to add fallback mechanism to the default bundles:

```

public static void RegisterBundles(BundleCollection bundles)
{
    var version = System.Reflection.Assembly.GetAssembly(typeof(BundleConfig))
        .GetName().Version.ToString();
    var cdnUrl = "http://cdnurl.azureedge.net/.../{0}?" + version;
    bundles.UseCdn = true;

    bundles.Add(new ScriptBundle("~/bundles/jquery", string.Format(cdnUrl, "bundles/jquery"))
        { CdnFallbackExpression = "window.jquery" }
        .Include("~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval", string.Format(cdnUrl, "bundles/jqueryval"))
        { CdnFallbackExpression = "$.validator" }
        .Include("~/Scripts/jquery.validate*"));

    // Use the development version of Modernizr to develop with and learn from. Then, when you're
    // ready for production, use the build tool at http://modernizr.com to pick only the tests you
    need.
    bundles.Add(new ScriptBundle("~/bundles/modernizr", string.Format(cdnUrl, "bundles/modernizr"))
        { CdnFallbackExpression = "window.Modernizr" }
        .Include("~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap", string.Format(cdnUrl, "bundles/bootstrap"))
        { CdnFallbackExpression = "$.fn.modal" }
        .Include(
            "~/Scripts/bootstrap.js",
            "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css")).Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css"));
}

```

When `CdnFallbackExpression` is not null, script is injected into the HTML to test whether the bundle is loaded successfully and, if not, access the bundle directly from the origin Web server. This property needs to be set to a JavaScript expression that tests whether the respective CDN bundle is loaded properly. The expression needed to test each bundle differs according to the content. For the default bundles above:

- `window.jquery` is defined in `jquery-{version}.js`
- `$.validator` is defined in `jquery.validate.js`
- `window.Modernizr` is defined in `modernizr-{version}.js`
- `$.fn.modal` is defined in `bootstrap.js`

You might have noticed that I did not set `CdnFallbackExpression` for the `~/Content/css` bundle. This is because currently there is a [bug in System.Web.Optimization](#) that injects a `<script>` tag for the fallback CSS instead of the expected `<link>` tag.

There is, however, a good [Style Bundle Fallback](#) offered by [Ember Consulting Group](#).

2. To use the workaround for CSS, create a new .cs file in your Web role project's `App_Start` folder called `StyleBundleExtensions.cs`, and replace its content with the [code from GitHub](#).
3. In `App_Start\StyleFundleExtensions.cs`, rename the namespace to your Web role's name (e.g. **WebRole1**).
4. Go back to `App_Start\BundleConfig.cs` and modify the last `bundles.Add` statement with the following highlighted code:

```

bundles.Add(new StyleBundle("~/Content/css", string.Format(cdnUrl, "Content/css")))
<mark>.IncludeFallback("~/Content/css", "sr-only", "width", "1px")</mark>
.Include(
    "~/Content/bootstrap.css",
    "~/Content/site.css"));

```

This new extension method uses the same idea to inject script in the HTML to check the DOM for the a matching class name, rule name, and rule value defined in the CSS bundle, and falls back to the origin Web server if it fails to find the match.

5. Publish the cloud service again and access the home page.
6. View the HTML code for the page. You should find injected scripts similar to the following:

```

...

<link href="http://az632148.azureedge.net/Content/css?v=1.0.0.25474" rel="stylesheet"/>
<script>(function() {
    var loadFallback,
        len = document.styleSheets.length;
    for (var i = 0; i < len; i++) {
        var sheet = document.styleSheets[i];
        if (sheet.href.indexOf('http://camservice.azureedge.net/Content/css?v=1.0.0.25474')
!= -1) {
            var meta = document.createElement('meta');
            meta.className = 'sr-only';
            document.head.appendChild(meta);
            var value = window.getComputedStyle(meta).getPropertyValue('width');
            document.head.removeChild(meta);
            if (value !== '1px') {
                document.write('<link href="/Content/css" rel="stylesheet" type="text/css"
/>');
            }
        }
    }
    return true;
})();||document.write('<script src="/Content/css"></script>');</script>

<script src="http://camservice.azureedge.net/bundles/modernizr?v=1.0.0.25474"></script>
<script>(window.Modernizr)||document.write('<script src="/bundles/modernizr"></script>');</script>

...

<script src="http://camservice.azureedge.net/bundles/jquery?v=1.0.0.25474"></script>
<script>(window.jquery)||document.write('<script src="/bundles/jquery"></script>');</script>

<script src="http://camservice.azureedge.net/bundles/bootstrap?v=1.0.0.25474"></script>
<script>($.fn.modal)||document.write('<script src="/bundles/bootstrap"></script>');</script>

...

```

Note that injected script for the CSS bundle still contains the errant remnant from the `CdnFallbackExpression` property in the line:

```

})();||document.write('<script src="/Content/css"></script>');</script>

```

But since the first part of the `||` expression will always return true (in the line directly above that), the `document.write()` function will never run.

## More Information

- [Overview of the Azure Content Delivery Network \(CDN\)](#)
- [Using Azure CDN](#)
- [ASP.NET Bundling and Minification](#)

# Integrate an Azure storage account with Azure CDN

5/11/2017 • 4 min to read • [Edit Online](#)

CDN can be enabled to cache content from your Azure storage. It offers developers a global solution for delivering high-bandwidth content by caching blobs and static content of compute instances at physical nodes in the United States, Europe, Asia, Australia and South America.

## Step 1: Create a storage account

Use the following procedure to create a new storage account for a Azure subscription. A storage account gives access to Azure storage services. The storage account represents the highest level of the namespace for accessing each of the Azure storage service components: Blob services, Queue services, and Table services. For more information, refer to the [Introduction to Microsoft Azure Storage](#).

To create a storage account, you must be either the service administrator or a co-administrator for the associated subscription.

### NOTE

There are several methods you can use to create a storage account, including the Azure Portal and Powershell. For this tutorial, we'll be using the Azure Portal.

### To create a storage account for an Azure subscription

1. Sign in to the [Azure Portal](#).
2. In the upper left corner, select **New**. In the **New** Dialog, select **Data + Storage**, then click **Storage account**.

The **Create storage account** blade appears.

Create storage ac...

The cost of your storage account depends on the usage and the options [Learn more](#)

\* Name ?

.core.windows.net

Deployment model ?

**Resource manager** Classic

Account kind ?

General purpose

Performance ?

**Standard** Premium

Replication ?

Read-access geo-redundant storage (RA...)

\* Subscription

msdn

\* Resource group ?

☒ Create new ☐ Use existing

\* Location

Central US

☐ Pin to dashboard

**Create**

3. In the **Name** field, type a subdomain name. This entry can contain 3-24 lowercase letters and numbers.

This value becomes the host name within the URI that is used to address Blob, Queue, or Table resources for the subscription. To address a container resource in the Blob service, you would use a URI in the following format, where *<StorageAccountLabel>* refers to the value you typed in **Enter a URL**:

`http://<StorageAccountLabel>.blob.core.windows.net/<mycontainer>`

**Important:** The URL label forms the subdomain of the storage account URI and must be unique among all hosted services in Azure.

This value is also used as the name of this storage account in the portal, or when accessing this account programmatically.

4. Leave the defaults for **Deployment model**, **Account kind**, **Performance**, and **Replication**.
5. Select the **Subscription** that the storage account will be used with.
6. Select or create a **Resource Group**. For more information on Resource Groups, see [Azure Resource Manager overview](#).
7. Select a location for your storage account.
8. Click **Create**. The process of creating the storage account might take several minutes to complete.

## Step 2: Enable CDN for the storage account

With the newest integration, you can now enable CDN for your storage account without leaving your storage

portal extension.

1. Select the storage account, search "CDN" or scroll down from the left navigation menu, then click "Azure CDN".

The **Azure CDN** blade appears.

## New endpoint

CDN profile ⓘ

☒ Create new ☐ Use existing

\* Pricing tier ([View full pricing details](#))

\* CDN endpoint name ⓘ

.azureedge.net

Origin hostname ⓘ

Create

2. Create a new endpoint by entering the required information

- **CDN Profile:** You can create a new or use an existing profile.
- **Pricing tier:** You only need to select a pricing tier if you create a new CDN profile.
- **CDN endpoint name:** Enter an endpoint name per your choice.

### TIP


The created CDN endpoint uses the hostname of your storage account as origin by default.

![cdn new endpoint creation][cdn-new-endpoint-creation]

3. After creation, the new endpoint will show up in the endpoint list above.



Azure CDN



Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements. [Learn more](#)

Endpoints

HOSTNAME	STATUS	PROTOCOL	
dustydogg.azureedge.net	Running	HTTP, HTTPS	...

New endpoint

CDN profile

Create new

Use existing

\* Pricing tier (View full pricing details)

\* CDN endpoint name

.azureedge.net

Origin hostname

dustydoggwebapp.azurewebsites.net

Create

#### NOTE

You can also go to Azure CDN extension to enable CDN.[Tutorial](#).

### To create a new CDN profile

1. In the [Azure Portal](#), in the upper left, click **New**. In the **New** blade, select **Web + Mobile**, then **CDN**.

The new CDN profile blade appears.

CDN profile

\* Name

\* Subscription

msdn

\* Resource group ⓘ

☒ Create new ☐ Use existing

\* Resource group location ⓘ

Central US

\* Pricing tier

Configure required settings

☐ Pin to dashboard

Create Automation options

2. Enter a name for your CDN profile.
3. Select a **Location**. This is the Azure location where your CDN profile information will be stored. It has no impact on CDN endpoint locations.
4. Select or create a **Resource Group**. For more information on Resource Groups, see [Azure Resource Manager overview](#).
5. Select a **Pricing tier**. See the [CDN Overview](#) for a comparison of pricing tiers.

Choose your pricing tier

Browse the available plans and their features

The premium pricing tier adds powerful features, advanced analytics, and more. Price varies based on region and data usage. [Learn more](#)

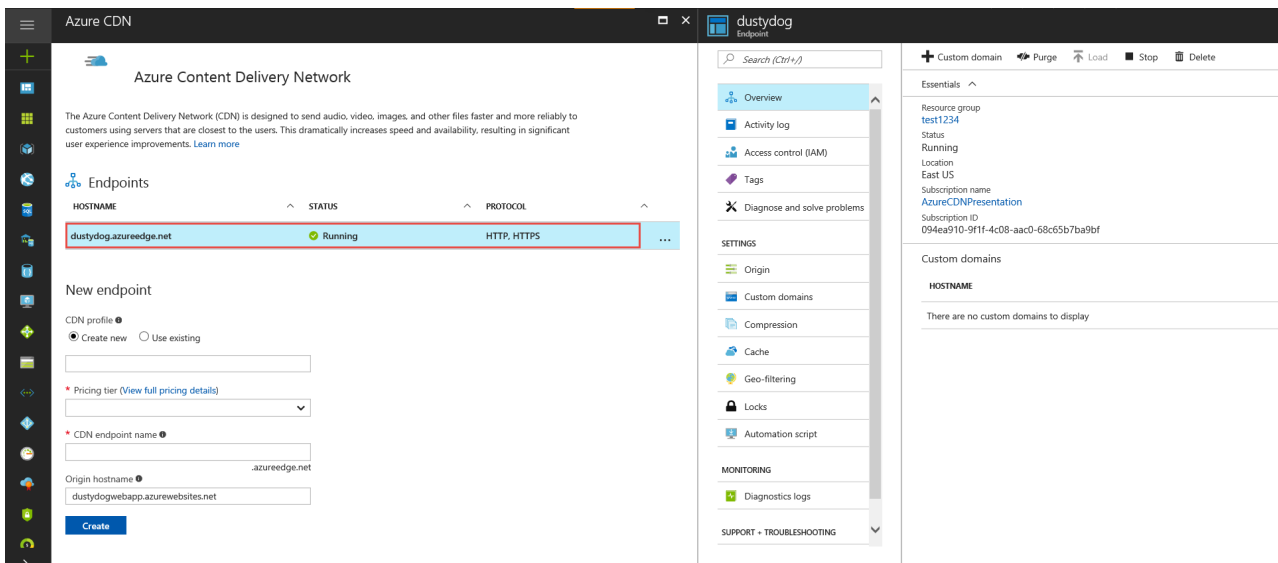
P1 Premium Verizon	S1 Standard Verizon	S2 Standard Akamai
SSL for CDN endpoint	SSL for CDN endpoint	SSL for CDN endpoint
Content Purge/Load	Content Purge/Load	Content Purge
Compression	Compression	Compression
Query strings caching	Query strings caching	Query strings caching
Country filtering	Country filtering	
Rules engine	Core analytics	
Advanced analytics		
Realtime analytics		

6. Select the **Subscription** for this CDN profile.

7. Click the **Create** button to create the new profile.

## Step 3: Enable additional CDN features

From storage account "Azure CDN" blade, click the CDN endpoint from the list to open CDN configuration blade. You can enable additional CDN features for your delivery, such as compression, query string, geo filtering. You can also add custom domain mapping to your CDN endpoint and enable custom domain HTTPS.



## Step 4: Access CDN content

To access cached content on the CDN, use the CDN URL provided in the portal. The address for a cached blob will be similar to the following:

`http://<EndpointName>.azureedge.net/<myPublicContainer>/<BlobName>`

### NOTE

Once you enable CDN access to a storage account, all publicly available objects are eligible for CDN edge caching. If you modify an object that is currently cached in the CDN, the new content will not be available via the CDN until the CDN refreshes its content when the cached content time-to-live period expires.

## Step 5: Remove content from the CDN

If you no longer wish to cache an object in the Azure Content Delivery Network (CDN), you can take one of the following steps:

- You can make the container private instead of public. See [Manage anonymous read access to containers and blobs](#) for more information.
- You can disable or delete the CDN endpoint using the Management Portal.
- You can modify your hosted service to no longer respond to requests for the object.

An object already cached in the CDN will remain cached until the time-to-live period for the object expires or until the endpoint is purged. When the time-to-live period expires, the CDN will check to see whether the CDN endpoint is still valid and the object still anonymously accessible. If it is not, then the object will no longer be cached.

## Additional resources

- [How to Map CDN Content to a Custom Domain](#)

- [Enable HTTPS for your custom domain](#)

# Using Azure CDN with CORS

1/24/2017 • 4 min to read • [Edit Online](#)

## What is CORS?

CORS (Cross Origin Resource Sharing) is an HTTP feature that enables a web application running under one domain to access resources in another domain. In order to reduce the possibility of cross-site scripting attacks, all modern web browsers implement a security restriction known as [same-origin policy](#). This prevents a web page from calling APIs in a different domain. CORS provides a secure way to allow one origin (the origin domain) to call APIs in another origin.

## How it works

There are two types of CORS requests, *simple requests* and *complex requests*.

### For simple requests:

1. The browser sends the CORS request with an additional **Origin** HTTP request header. The value of this header is the origin that served the parent page, which is defined as the combination of *protocol*, *domain*, and *port*. When a page from <https://www.contoso.com> attempts to access a user's data in the fabrikam.com origin, the following request header would be sent to fabrikam.com:

```
Origin: https://www.contoso.com
```

2. The server may respond with any of the following:

- An **Access-Control-Allow-Origin** header in its response indicating which origin site is allowed. For example:

```
Access-Control-Allow-Origin: https://www.contoso.com
```

- An HTTP error code such as 403 if the server does not allow the cross-origin request after checking the Origin header
- An **Access-Control-Allow-Origin** header with a wildcard that allows all origins:

```
Access-Control-Allow-Origin: *
```

### For complex requests:

A complex request is a CORS request where the browser is required to send a *preflight request* (i.e. a preliminary probe) before sending the actual CORS request. The preflight request asks the server permission if the original CORS request can proceed and is an `OPTIONS` request to the same URL.

#### TIP

For more details on CORS flows and common pitfalls, view the [Guide to CORS for REST APIs](#).

## Wildcard or single origin scenarios

CORS on Azure CDN will work automatically with no additional configuration when the **Access-Control-Allow-Origin** header is set to wildcard (\*) or a single origin. The CDN will cache the first response and subsequent requests will use the same header.

If requests have already been made to the CDN prior to CORS being set on the your origin, you will need to purge content on your endpoint content to reload the content with the **Access-Control-Allow-Origin** header.

## Multiple origin scenarios

If you need to allow a specific list of origins to be allowed for CORS, things get a little more complicated. The problem occurs when the CDN caches the **Access-Control-Allow-Origin** header for the first CORS origin. When a different CORS origin makes a subsequent request, the CDN will serve the cached **Access-Control-Allow-Origin** header, which won't match. There are several ways to correct this.

### Azure CDN Premium from Verizon

The best way to enable this is to use **Azure CDN Premium from Verizon**, which exposes some advanced functionality.

You'll need to [create a rule](#) to check the **Origin** header on the request. If it's a valid origin, your rule will set the **Access-Control-Allow-Origin** header with the origin provided in the request. If the origin specified in the **Origin** header is not allowed, your rule should omit the **Access-Control-Allow-Origin** header which will cause the browser to reject the request.

There are two ways to do this with the rules engine. In both cases, the **Access-Control-Allow-Origin** header from the file's origin server is completely ignored, the CDN's rules engine completely manages the allowed CORS origins.

#### One regular expression with all valid origins

In this case, you'll create a regular expression that includes all of the origins you want to allow:

```
https?:\\/(www\\.contoso\\.com|contoso\\.com|www\\.microsoft\\.com|microsoft\\.com\\.com)$
```

#### TIP

**Azure CDN from Verizon** uses [Perl Compatible Regular Expressions](#) as its engine for regular expressions. You can use a tool like [Regular Expressions 101](#) to validate your regular expression. Note that the "/" character is valid in regular expressions and doesn't need to be escaped, however, escaping that character is considered a best practice and is expected by some regex validators.

If the regular expression matches, your rule will replace the **Access-Control-Allow-Origin** header (if any) from the origin with the origin that sent the request. You can also add additional CORS headers, such as **Access-Control-Allow-Methods**.

IF		Request Header Regex	Name	Origin	Matches	Value	https?:\\/(www\\.contoso\\.com contoso\\.com www\\.microsoft\\.com microsoft\\.com\\.com)\$	Ignore Case
Features	+	+	Modify Client Response Header	Overwrite	Name	Access-Control-Allow-Origin	Value	%{http_origin}
			Modify Client Response Header	Overwrite	Name	Access-Control-Allow-Headers	Value	*
			Modify Client Response Header	Overwrite	Name	Access-Control-Allow-Methods	Value	GET, HEAD, OPTIONS
			Modify Client Response Header	Overwrite	Name	Access-Control-Expose-Headers	Value	*
Matches		+						

#### Request header rule for each origin.

Rather than regular expressions, you can instead create a separate rule for each origin you wish to allow using the **Request Header Wildcard match condition**. As with the regular expression method, the rules engine alone sets the CORS headers.

IF		Request Header Wildcard	Name	Origin	Matches	Value(s)	*.//contoso.com	Ignore Case
Features	+	+	Modify Client Response Header	Overwrite	Name	Access-Control-Allow-Origin	Value	%{http_origin}
			Matches					+

**TIP**

In the example above, the use of the wildcard character \* tells the rules engine to match both HTTP and HTTPS.

**Azure CDN Standard**

On Azure CDN Standard profiles, the only mechanism to allow for multiple origins without the use of the wildcard origin is to use [query string caching](#). You need to enable query string setting for the CDN endpoint and then use a unique query string for requests from each allowed domain. Doing this will result in the CDN caching a separate object for each unique query string. This approach is not ideal, however, as it will result in multiple copies of the same file cached on the CDN.

# Enable HTTPS on an Azure CDN custom domain

3/3/2017 • 3 min to read • [Edit Online](#)

## IMPORTANT

This feature is available with **Azure CDN from Verizon** products (Standard and Premium). It is not supported on **Azure CDN from Akamai**. For a comparison of CDN features, see [Azure CDN Overview](#).

HTTPS support for Azure CDN custom domains enables you to deliver secure content via SSL using your own domain name to improve the security of data while in transit. The end-to-end workflow to enable HTTPS for your custom domain is simplified via one-click enablement, complete certificate management, and all with no additional cost.

It's critical to ensure the privacy and data integrity of all of your web applications sensitive data while in transit. Using the HTTPS protocol ensures that your sensitive data is encrypted when it is sent across the internet. It provides trust, authentication and protects your web applications from attacks. Currently, Azure CDN supports HTTPS on a CDN endpoint. For example, if you create a CDN endpoint from Azure CDN (e.g. <https://contoso.azureedge.net>), HTTPS is enabled by default. Now, with custom domain HTTPS, you can enable secure delivery for a custom domain (e.g. <https://www.contoso.com>) as well.

Some of the key attributes of HTTPS feature are:

- No additional cost: There are no costs for certificate acquisition or renewal and no additional cost for HTTPS traffic. You just pay for GB egress from the CDN.
- Simple enablement: One click provisioning is available from the [Azure portal](#). You can also use REST API or other developer tools to enable the feature.
- Complete certificate management: All certificate procurement and management is handled for you. Certificates are automatically provisioned and renewed prior to expiration. This completely removes the risks of service interruption as a result of a certificate expiring.

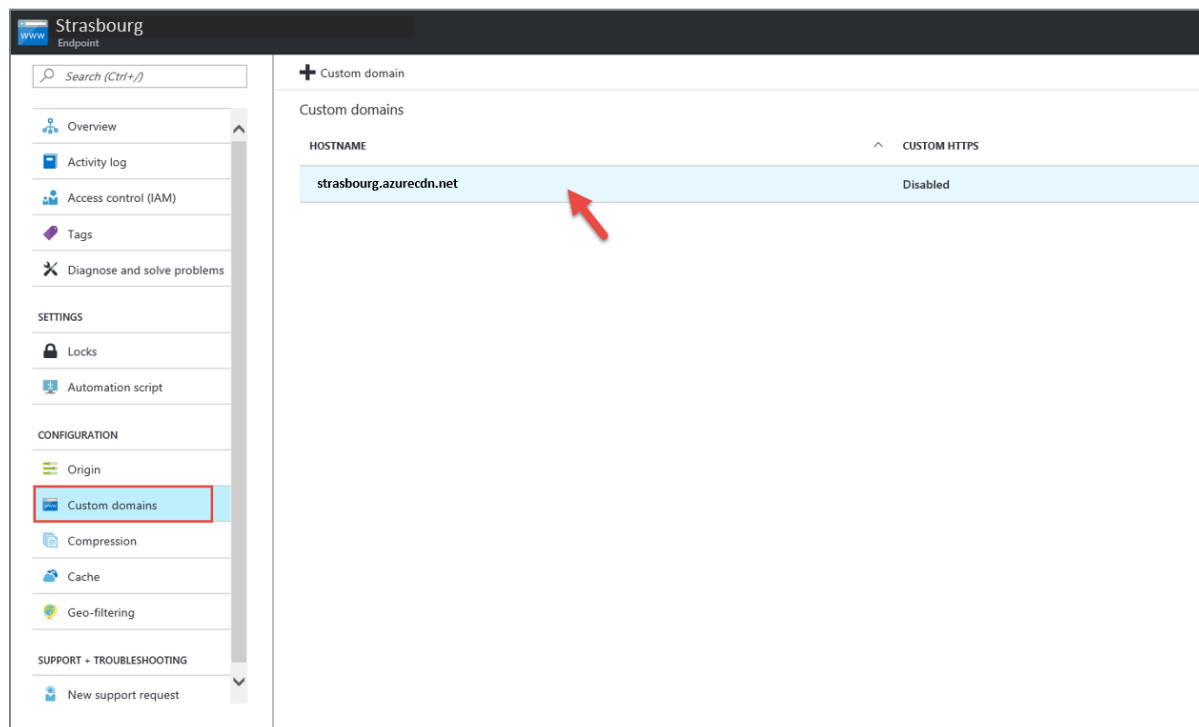
## NOTE

Prior to enabling HTTPS support, you must have already established an [Azure CDN custom domain](#).

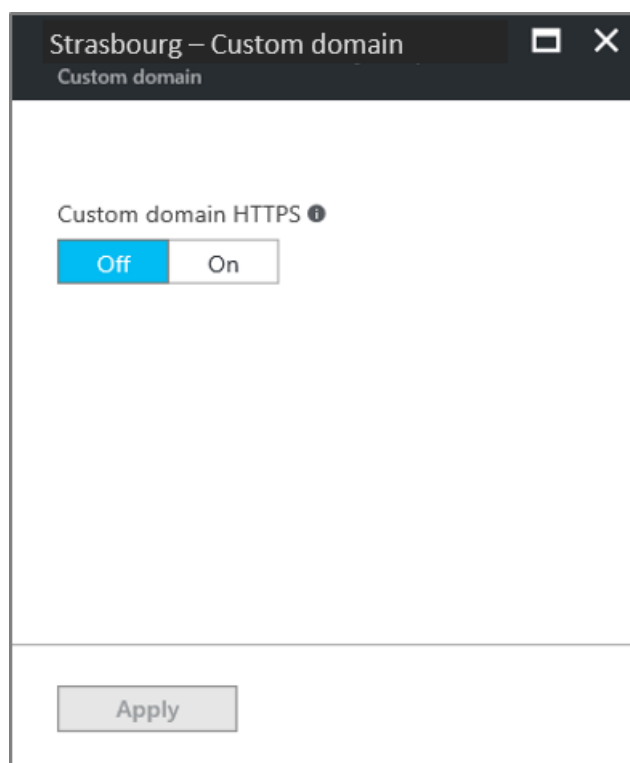
## Step 1: Enabling the feature

1. In the [Azure portal](#), browse to your Verizon standard or premium CDN profile.
2. In the list of endpoints, click the endpoint containing your custom domain.
3. Click the custom domain for which you want to enable HTTPS.





- Click **On** to enable HTTPS and save the change.



## Step 2: Domain validation

### IMPORTANT

You must complete domain validation before HTTPS will be active on your custom domain. You have 6 business days to approve the domain. Request will be canceled with no approval within 6 business days.

After enabling HTTPS on your custom domain, our HTTPS certificate provider DigiCert will validate ownership of your domain by contacting the registrant for your domain, based on WHOIS registrant information, via email (by default) or phone. DigiCert will also send the verification email to the below addresses. If WHOIS registrant information is private, make sure you can approve directly from one of these addresses.

admin@ administrator@  
webmaster@  
hostmaster@  
postmaster@

Upon receiving the email, you have two verification options:

1. You can approve all future orders placed through the same account for the same root domain, e.g. consoto.com. This is a recommended approach if you are planning to add additional custom domains in the future for the same root domain.
2. You can just approve the specific host name used in this request. Additional approval will be required for subsequent requests.

Example email:

Please review and approve the request below. Approval is required prior to the certificate's issuance. If this approval request contains any errors or if you wish to cancel this request, please contact DigiCert by phone or email.

Domain [REDACTED]

Organization Verizon Digital Media Services, Inc.  
Los Angeles, California, USA

Product 1 Year SSL Plus

**Authorization:**

I am the registrant or am authorized by the registrant of the domain name(s) referenced above and have the authority to approve certificate requests for the domain. I confirm and agree to the following:

1. **Verizon Digital Media Services** (account [REDACTED]) has the authority to apply for SSL Certificates for this domain on behalf of **Verizon Digital Media Services, Inc.** (order # [REDACTED]),
2. **Verizon Digital Media Services, Inc.** has the right to use and obtain digital certificates for both the domain(s) referenced above and any sub domain(s) of the referenced domains,
3. DigiCert may rely on this authorization for any subsequent digital certificate renewals and/or orders by **Verizon Digital Media Services, Inc.** until this authorization is revoked by written notice to DigiCert using one of the following methods: 1) Email to admin@digicert.com or 2) Mail to DigiCert, Inc. (Attention: Legal) - 2801 N. Thanksgiving Way - Suite 500 - Lehi, UT 84043
4. I will promptly notify DigiCert if this authorization is revoked or if a domain name listed above is transferred to a third party, and
5. DigiCert may periodically (typically once every three years) reconfirm the Applicant's control over the domain name(s) and approval of the corresponding certificate(s) using a reconfirmation email sent to this email address. I acknowledge that I may not opt out of receiving reconfirmation emails.

☒ I approve all future orders placed through account [REDACTED]B for orders under the [REDACTED] domain name(s). (Recommended)

☐ I approve the specific host names used in this request.

After approval, DigiCert will add your custom domain name to the SAN certificate. The certificate will be valid for one year and will be auto renewed before it's expired.

## Step 3: Wait for the propagation then start using your feature

After the domain name is validated it will take up to 6-8 hours for the custom domain HTTPS feature to be active. After the process is complete, the "custom HTTPS" status in the Azure portal will be set to "Enabled". HTTPS with your custom domain is now ready for your use.

## Frequently asked questions

1. *Who is the certificate provider and what type of certificate is used?*

We use Subject Alternative Names (SAN) certificate provided by DigiCert. A SAN certificate can secure multiple fully qualified domain names with one certificate.

2. *Can I use my dedicated certificate?*

Not currently, but it's on the roadmap.

3. *What if I don't receive the domain verification email from DigiCert?*

Please contact Microsoft if you don't receive an email within 24 hours.

4. *Is using a SAN certificate less secure than a dedicated certificate?*

A SAN cert follows the same encryption and security standards as a dedicated cert. All issued SSL certificates are using SHA-256 for enhanced server security.

5. *Can I use custom domain HTTPS with Azure CDN from Akamai?*

Currently, this feature is only available with Azure CDN from Verizon. We are working on supporting this feature with Azure CDN from Akamai in the coming months.

## Next steps

- Learn how to set up a [custom domain on your Azure CDN endpoint](#)

# Manage Azure CDN with PowerShell

4/27/2017 • 4 min to read • [Edit Online](#)

PowerShell provides one of the most flexible methods to manage your Azure CDN profiles and endpoints. You can use PowerShell interactively or by writing scripts to automate management tasks. This tutorial demonstrates several of the most common tasks you can accomplish with PowerShell to manage your Azure CDN profiles and endpoints.

## Prerequisites

To use PowerShell to manage your Azure CDN profiles and endpoints, you must have the Azure PowerShell module installed. To learn how to install Azure PowerShell and connect to Azure using the `Login-AzureRmAccount` cmdlet, see [How to install and configure Azure PowerShell](#).

### IMPORTANT

You must log in with `Login-AzureRmAccount` before you can execute Azure PowerShell cmdlets.

## Listing the Azure CDN cmdlets

You can list all the Azure CDN cmdlets using the `Get-Command` cmdlet.

```
PS C:\> Get-Command -Module AzureRM.Cdn
```

CommandType	Name	Version	Source
-----	----	-----	-----
Cmdlet	Get-AzureRmCdnCustomDomain	2.0.0	AzureRm.Cdn
Cmdlet	Get-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	Get-AzureRmCdnEndpointNameAvailability	2.0.0	AzureRm.Cdn
Cmdlet	Get-AzureRmCdnOrigin	2.0.0	AzureRm.Cdn
Cmdlet	Get-AzureRmCdnProfile	2.0.0	AzureRm.Cdn
Cmdlet	Get-AzureRmCdnProfileSsoUrl	2.0.0	AzureRm.Cdn
Cmdlet	New-AzureRmCdnCustomDomain	2.0.0	AzureRm.Cdn
Cmdlet	New-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	New-AzureRmCdnProfile	2.0.0	AzureRm.Cdn
Cmdlet	Publish-AzureRmCdnEndpointContent	2.0.0	AzureRm.Cdn
Cmdlet	Remove-AzureRmCdnCustomDomain	2.0.0	AzureRm.Cdn
Cmdlet	Remove-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	Remove-AzureRmCdnProfile	2.0.0	AzureRm.Cdn
Cmdlet	Set-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	Set-AzureRmCdnOrigin	2.0.0	AzureRm.Cdn
Cmdlet	Set-AzureRmCdnProfile	2.0.0	AzureRm.Cdn
Cmdlet	Start-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	Stop-AzureRmCdnEndpoint	2.0.0	AzureRm.Cdn
Cmdlet	Test-AzureRmCdnCustomDomain	2.0.0	AzureRm.Cdn
Cmdlet	Unpublish-AzureRmCdnEndpointContent	2.0.0	AzureRm.Cdn

## Getting help

You can get help with any of these cmdlets using the `Get-Help` cmdlet. `Get-Help` provides usage and syntax, and optionally shows examples.

```
PS C:\> Get-Help Get-AzureRmCdnProfile
```

#### NAME

Get-AzureRmCdnProfile

#### SYNOPSIS

Gets an Azure CDN profile.

#### SYNTAX

```
Get-AzureRmCdnProfile [-ProfileName <String>] [-ResourceGroupName <String>] [-InformationAction  
<ActionPreference>] [-InformationVariable <String>] [<CommonParameters>]
```

#### DESCRIPTION

Gets an Azure CDN profile and all related information.

#### RELATED LINKS

#### REMARKS

To see the examples, type: "get-help Get-AzureRmCdnProfile -examples".  
For more information, type: "get-help Get-AzureRmCdnProfile -detailed".  
For technical information, type: "get-help Get-AzureRmCdnProfile -full".

## Listing existing Azure CDN profiles

The `Get-AzureRmCdnProfile` cmdlet without any parameters retrieves all your existing CDN profiles.

```
Get-AzureRmCdnProfile
```

This output can be piped to cmdlets for enumeration.

```
# Output the name of all profiles on this subscription.  
Get-AzureRmCdnProfile | ForEach-Object { Write-Host $_.Name }  
  
# Return only **Azure CDN from Verizon** profiles.  
Get-AzureRmCdnProfile | Where-Object { $_.Sku.Name -eq "StandardVerizon" }
```

You can also return a single profile by specifying the profile name and resource group.

```
Get-AzureRmCdnProfile -ProfileName CdnDemo -ResourceGroupName CdnDemoRG
```

#### TIP

It is possible to have multiple CDN profiles with the same name, so long as they are in different resource groups. Omitting the `ResourceGroupName` parameter returns all profiles with a matching name.

## Listing existing CDN endpoints

`Get-AzureRmCdnEndpoint` can retrieve an individual endpoint or all the endpoints on a profile.

```
# Get a single endpoint.
Get-AzureRmCdnEndpoint -ProfileName CdnDemo -ResourceGroupName CdnDemoRG -EndpointName cdndocdemo

# Get all of the endpoints on a given profile.
Get-AzureRmCdnEndpoint -ProfileName CdnDemo -ResourceGroupName CdnDemoRG

# Return all of the endpoints on all of the profiles.
Get-AzureRmCdnProfile | Get-AzureRmCdnEndpoint

# Return all of the endpoints in this subscription that are currently running.
Get-AzureRmCdnProfile | Get-AzureRmCdnEndpoint | Where-Object { $_.ResourceState -eq "Running" }
```

## Creating CDN profiles and endpoints

`New-AzureRmCdnProfile` and `New-AzureRmCdnEndpoint` are used to create CDN profiles and endpoints.

```
# Create a new profile
New-AzureRmCdnProfile -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -Sku StandardAkamai -Location "Central US"

# Create a new endpoint
New-AzureRmCdnEndpoint -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -Location "Central US" -
EndpointName cdnposhdcc -OriginName "Contoso" -OriginHostName "www.contoso.com"

# Create a new profile and endpoint (same as above) in one line
New-AzureRmCdnProfile -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -Sku StandardAkamai -Location "Central US" | New-AzureRmCdnEndpoint -EndpointName cdnposhdcc -OriginName "Contoso" -OriginHostName "www.contoso.com"
```

## Checking endpoint name availability

`Get-AzureRmCdnEndpointNameAvailability` returns an object indicating if an endpoint name is available.

```
# Retrieve availability
$availability = Get-AzureRmCdnEndpointNameAvailability -EndpointName "cdnposhdcc"

# If available, write a message to the console.
If($availability.NameAvailable) { Write-Host "Yes, that endpoint name is available." }
Else { Write-Host "No, that endpoint name is not available." }
```

## Adding a custom domain

`New-AzureRmCdnCustomDomain` adds a custom domain name to an existing endpoint.

### IMPORTANT

You must set up the CNAME with your DNS provider as described in [How to map Custom Domain to Content Delivery Network \(CDN\) endpoint](#). You can test the mapping before modifying your endpoint using `Test-AzureRmCdnCustomDomain`.

```
# Get an existing endpoint
$endpoint = Get-AzureRmCdnEndpoint -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -EndpointName
cdnposhdoc

# Check the mapping
$result = Test-AzureRmCdnCustomDomain -CdnEndpoint $endpoint -CustomDomainHostName "cdn.contoso.com"

# Create the custom domain on the endpoint
If($result.CustomDomainValidated){ New-AzureRmCdnCustomDomain -CustomDomainName Contoso -HostName
"cdn.contoso.com" -CdnEndpoint $endpoint }
```

## Modifying an endpoint

`Set-AzureRmCdnEndpoint` modifies an existing endpoint.

```
# Get an existing endpoint
$endpoint = Get-AzureRmCdnEndpoint -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -EndpointName
cdnposhdoc

# Set up content compression
$endpoint.IsCompressionEnabled = $true
$endpoint.ContentTypesToCompress = "text/javascript","text/css","application/json"

# Save the changed endpoint and apply the changes
Set-AzureRmCdnEndpoint -CdnEndpoint $endpoint
```

## Purging/Pre-loading CDN assets

`Unpublish-AzureRmCdnEndpointContent` purges cached assets, while `Publish-AzureRmCdnEndpointContent` pre-loads assets on supported endpoints.

```
# Purge some assets.
Unpublish-AzureRmCdnEndpointContent -ProfileName CdnDemo -ResourceGroupName CdnDemoRG -EndpointName cdndocdemo
-PurgeContent "/images/kitten.png","/video/rickroll.mp4"

# Pre-load some assets.
Publish-AzureRmCdnEndpointContent -ProfileName CdnDemo -ResourceGroupName CdnDemoRG -EndpointName cdndocdemo -
LoadContent "/images/kitten.png","/video/rickroll.mp4"

# Purge everything in /images/ on all endpoints.
Get-AzureRmCdnProfile | Get-AzureRmCdnEndpoint | Unpublish-AzureRmCdnEndpointContent -PurgeContent "/images/*"
```

## Starting/Stopping CDN endpoints

`Start-AzureRmCdnEndpoint` and `Stop-AzureRmCdnEndpoint` can be used to start and stop individual endpoints or groups of endpoints.

```
# Stop the cdndocdemo endpoint
Stop-AzureRmCdnEndpoint -ProfileName CdnDemo -ResourceGroupName CdnDemoRG -EndpointName cdndocdemo

# Stop all endpoints
Get-AzureRmCdnProfile | Get-AzureRmCdnEndpoint | Stop-AzureRmCdnEndpoint

# Start all endpoints
Get-AzureRmCdnProfile | Get-AzureRmCdnEndpoint | Start-AzureRmCdnEndpoint
```

# Deleting CDN resources

`Remove-AzureRmCdnProfile` and `Remove-AzureRmCdnEndpoint` can be used to remove profiles and endpoints.

```
# Remove a single endpoint
Remove-AzureRmCdnEndpoint -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG -EndpointName cdnposhdoc

# Remove all the endpoints on a profile and skip confirmation (-Force)
Get-AzureRmCdnProfile -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG | Get-AzureRmCdnEndpoint | Remove-
AzureRmCdnEndpoint -Force

# Remove a single profile
Remove-AzureRmCdnProfile -ProfileName CdnPoshDemo -ResourceGroupName CdnDemoRG
```

## Next Steps

Learn how to automate Azure CDN with [.NET](#) or [Node.js](#).

To learn about CDN features, see [CDN Overview](#).



# Manage expiration of Azure Web Apps/Cloud Services, ASP.NET, or IIS content in Azure CDN

4/28/2017 • 2 min to read • [Edit Online](#)

Files from any publicly accessible origin web server can be cached in Azure CDN until its time-to-live (TTL) elapses. The TTL is determined by the [Cache-Control header](#) in the HTTP response from the origin server. This article describes how to set `Cache-Control` headers for Azure Web Apps, Azure Cloud Services, ASP.NET applications, and Internet Information Services sites, all of which are configured similarly.

## TIP

You may choose to set no TTL on a file. In this case, Azure CDN automatically applies a default TTL of seven days.

For more information about how Azure CDN works to speed up access to files and other resources, see the [Azure CDN Overview](#).

## Setting Cache-Control Headers in configuration

For static content, such as images and style sheets, you can control the update frequency by modifying the **applicationHost.config** or **web.config** files for your web application. The **system.webServer\staticContent\clientCache** element in the configuration file will set the `Cache-Control` header for your content. For **web.config**, the configuration settings will affect everything in the folder and all subfolders, unless overridden at the subfolder level. For example, you can set a default time-to-live at the root to have all static content cached for 3 days, but have a subfolder that has more variable content with a cache setting of 6 hours. For **applicationHost.config**, all applications on the site will be affected, but can be overridden in **web.config** files in the applications.

The following XML shows an example of setting **clientCache** to specify a maximum age of 3 days:

```
<configuration>
  <system.webServer>
    <staticContent>
      <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="3.00:00:00" />
    </staticContent>
  </system.webServer>
</configuration>
```

Specifying **UseMaxAge** adds a `Cache-Control: max-age=<nnn>` header to the response based on the value specified in the **CacheControlMaxAge** attribute. The format of the timespan is for the **cacheControlMaxAge** attribute is ... For more information on the **clientCache** node, see [Client Cache](#).

## Setting Cache-Control Headers in Code

For ASP.NET applications, you can set the CDN caching behavior programmatically by setting the **HttpResponse.Cache** property. For more information on the **HttpResponse.Cache** property, see [HttpResponse.Cache Property](#) and [HttpCachePolicy Class](#).

If you want to programmatically cache application content in ASP.NET, make sure that the content is marked as cacheable by setting `HttpCacheability` to `Public`. Also, ensure that a cache validator is set. The cache validator can be a Last Modified timestamp set by calling `SetLastModified`, or an etag value set by calling `SetETag`. Optionally, you

can also specify a cache expiration time by calling `SetExpires`, or you can rely on the default cache heuristics described earlier in this document.

For example, to cache content for one hour, add the following:

```
// Set the caching parameters.  
Response.Cache.SetExpires(DateTime.Now.AddHours(1));  
Response.Cache.SetCacheability(HttpCacheability.Public);  
Response.Cache.SetLastModified(DateTime.Now);
```

## Next Steps

- [Read details about the `clientCache` element](#)
- [Read the documentation for the `HttpResponse.Cache` Property](#)
- [Read the documentation for the `HttpCachePolicy Class`.](#)

# Manage expiration of Azure Storage blobs in Azure CDN

4/27/2017 • 2 min to read • [Edit Online](#)

The [blob service](#) in [Azure Storage](#) is one of several Azure-based origins integrated with Azure CDN. Any publicly accessible blob content can be cached in Azure CDN until its time-to-live (TTL) elapses. The TTL is determined by the [Cache-Control header](#) in the HTTP response from Azure Storage.

## TIP

You may choose to set no TTL on a blob. In this case, Azure CDN automatically applies a default TTL of seven days.

For more information about how Azure CDN works to speed up access to blobs and other files, see the [Azure CDN Overview](#).

For more details on the Azure Storage blob service, see [Blob Service Concepts](#).

This tutorial demonstrates several ways that you can set the TTL on a blob in Azure Storage.

## Azure PowerShell

[Azure PowerShell](#) is one of the quickest, most powerful ways to administer your Azure services. Use the

`Get-AzureStorageBlob` cmdlet to get a reference to the blob, then set the `.ICloudBlob.Properties.CacheControl` property.

```
# Create a storage context
$context = New-AzureStorageContext -StorageAccountName "<storage account name>" -StorageAccountKey "<storage account key>"

# Get a reference to the blob
$blob = Get-AzureStorageBlob -Context $context -Container "<container name>" -Blob "<blob name>"

# Set the CacheControl property to expire in 1 hour (3600 seconds)
$blob.ICloudBlob.Properties.CacheControl = "public, max-age=3600"

# Send the update to the cloud
$blob.ICloudBlob.SetProperties()
```

## TIP

You can also use PowerShell to [manage your CDN profiles and endpoints](#).

## Azure Storage Client Library for .NET

To set a blob's TTL using .NET, use the [Azure Storage Client Library for .NET](#) to set the `CloudBlob.Properties.CacheControl` property.

```

class Program
{
    const string connectionString = "<storage connection string>";
    static void Main()
    {
        // Retrieve storage account information from connection string
        CloudStorageAccount storageAccount = CloudStorageAccount.Parse(connectionString);

        // Create a blob client for interacting with the blob service.
        CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

        // Create a reference to the container
        CloudBlobContainer container = blobClient.GetContainerReference("<container name>");

        // Create a reference to the blob
        CloudBlob blob = container.GetBlobReference("<blob name>");

        // Set the CacheControl property to expire in 1 hour (3600 seconds)
        blob.Properties.CacheControl = "public, max-age=3600";

        // Update the blob's properties in the cloud
        blob.SetProperties();
    }
}

```

#### TIP

There are many more .NET code samples available in the [Azure Blob Storage Samples for .NET](#).

## Other methods

- [Azure Command-Line Interface](#)

When uploading the blob, set the *cacheControl* property using the `-p` switch. This example sets the TTL to one hour (3600 seconds).

```

azure storage blob upload -c <connectionstring> -p cacheControl="public, max-age=3600" .\test.txt
myContainer test.txt

```

- [Azure Storage Services REST API](#)

Explicitly set the *x-ms-blob-cache-control* property on a [Put Blob](#), [Put Block List](#), or [Set Blob Properties](#) request.

- Third-party storage management tools

Some third-party Azure Storage management tools allow you to set the *CacheControl* property on blobs.

## Testing the *Cache-Control* header

You can easily verify the TTL of your blobs. Using your browser's [developer tools](#), test that your blob is including the *Cache-Control* response header. You can also use a tool like **wget**, [Postman](#), or [Fiddler](#) to examine the response headers.

## Next Steps

- [Read about the \*Cache-Control\* header](#)
- [Learn how to manage expiration of Cloud Service content in Azure CDN](#)



# Map Azure CDN content to a custom domain

4/28/2017 • 6 min to read • [Edit Online](#)

You can map a custom domain to a CDN endpoint in order to use your own domain name in URLs to cached content rather than using a subdomain of azureedge.net.

There are two ways to map your custom domain to a CDN endpoint:

1. [Create a CNAME record with your domain registrar and map your custom domain and subdomain to the CDN endpoint.](#)

A CNAME record is a DNS feature that maps a source domain, like `www.contosocdn.com` or `cdn.contoso.com`, to a destination domain. In this case, the source domain is your custom domain and subdomain (a subdomain, like **www** or **cdn** is always required). The destination domain is your CDN endpoint.

The process of mapping your custom domain to your CDN endpoint can, however, result in a brief period of downtime for the domain while you are registering the domain in the Azure Portal.

2. [Add an intermediate registration step with \*\*cdnverify\*\*](#)

If your custom domain is currently supporting an application with a service-level agreement (SLA) that requires that there be no downtime, then you can use the Azure **cdnverify** subdomain to provide an intermediate registration step so that users will be able to access your domain while the DNS mapping takes place.

After you register your custom domain using one of the above procedures, you will want to [verify that the custom subdomain references your CDN endpoint](#).

## NOTE

You must create a CNAME record with your domain registrar to map your domain to the CDN endpoint. CNAME records map specific subdomains such as `www.contoso.com` or `cdn.contoso.com`. It is not possible to map a CNAME record to a root domain, such as `contoso.com`.

A subdomain can only be associated with one CDN endpoint. The CNAME record that you create will route all traffic addressed to the subdomain to the specified endpoint. For example, if you associate `www.contoso.com` with your CDN endpoint, then you cannot associate it with other Azure endpoints, such as a storage account endpoint or a cloud service endpoint. However, you can use different subdomains from the same domain for different service endpoints. You can also map different subdomains to the same CDN endpoint.

For **Azure CDN from Verizon** (Standard and Premium) endpoints, note that it takes up to **90 minutes** for custom domain changes to propagate to CDN edge nodes.

## Register a custom domain for an Azure CDN endpoint

1. Log into the [Azure Portal](#).
2. Click **Browse**, then **CDN Profiles**, then the CDN profile with the endpoint you want to map to a custom domain.
3. In the **CDN Profile** blade, click the CDN endpoint with which you want to associate the subdomain.
4. At the top of the endpoint blade, click the **Add Custom Domain** button. In the **Add a custom domain** blade, you'll see the endpoint host name, derived from your CDN endpoint, to use in creating a new CNAME record. The format of the host name address will appear as **<EndpointName>.azureedge.net**. You can copy this

host name to use in creating the CNAME record.

5. Navigate to your domain registrar's web site, and locate the section for creating DNS records. You might find this in a section such as **Domain Name, DNS**, or **Name Server Management**.
6. Find the section for managing CNAMEs. You may have to go to an advanced settings page and look for the words CNAME, Alias, or Subdomains.
7. Create a new CNAME record that maps your chosen subdomain (for example, **www** or **cdn**) to the host name provided in the **Add a custom domain** blade.
8. Return to the **Add a custom domain** blade, and enter your custom domain, including the subdomain, in the dialog box. For example, enter the domain name in the format `www.contoso.com` or `cdn.contoso.com`.

Azure will verify that the CNAME record exists for the domain name you have entered. If the CNAME is correct, your custom domain is validated. For **Azure CDN from Verizon** (Standard and Premium) endpoints, it may take up to 90 minutes for custom domain settings to propagate to all CDN edge nodes, however.

Note that in some cases it can take time for the CNAME record to propagate to name servers on the Internet. If your domain is not validated immediately, and you believe the CNAME record is correct, then wait a few minutes and try again.

## Register a custom domain for an Azure CDN endpoint using the intermediary cdnverify subdomain

1. Log into the [Azure Portal](#).
2. Click **Browse**, then **CDN Profiles**, then the CDN profile with the endpoint you want to map to a custom domain.
3. In the **CDN Profile** blade, click the CDN endpoint with which you want to associate the subdomain.
4. At the top of the endpoint blade, click the **Add Custom Domain** button. In the **Add a custom domain** blade, you'll see the endpoint host name, derived from your CDN endpoint, to use in creating a new CNAME record. The format of the host name address will appear as **<EndpointName>.azureedge.net**. You can copy this host name to use in creating the CNAME record.
5. Navigate to your domain registrar's web site, and locate the section for creating DNS records. You might find this in a section such as **Domain Name, DNS**, or **Name Server Management**.
6. Find the section for managing CNAMEs. You may have to go to an advanced settings page and look for the words **CNAME, Alias**, or **Subdomains**.
7. Create a new CNAME record, and provide a subdomain alias that includes the **cdnverify** subdomain. For example, the subdomain that you specify will be in the format **cdnverify.www** or **cdnverify.cdn**. Then provide the host name, which is your CDN endpoint, in the format **cdnverify.<EndpointName>.azureedge.net**. Your DNS mapping should look like:

```
cdnverify.www.consoto.com CNAME cdnverify.consoto.azureedge.net
```

8. Return to the **Add a custom domain** blade, and enter your custom domain, including the subdomain, in the dialog box. For example, enter the domain name in the format `www.contoso.com` or `cdn.contoso.com`. Note that in this step, you do not need to preface the subdomain with **cdnverify**.

Azure will verify that the CNAME record exists for the cdnverify domain name you have entered.

9. At this point, your custom domain has been verified by Azure, but traffic to your domain is not yet being routed to your CDN endpoint. After waiting long enough to allow the custom domain settings to propagate to the CDN edge nodes (90 minutes for **Azure CDN from Verizon**, 1-2 minutes for **Azure CDN from Akamai**), return to your DNS registrar's web site and create another CNAME record that maps your subdomain to your CDN endpoint. For example, specify the subdomain as **www** or **cdn**, and the hostname as **<EndpointName>.azureedge.net**. With this step, the registration of your custom domain is complete.
10. Finally, you can delete the CNAME record you created using **cdnverify**, as it was necessary only as an

intermediary step.

## Verify that the custom subdomain references your CDN endpoint

- After you have completed the registration of your custom domain, you can access content that is cached at your CDN endpoint using the custom domain. First, ensure that you have public content that is cached at the endpoint. For example, if your CDN endpoint is associated with a storage account, the CDN caches content in public blob containers. To test the custom domain, ensure that your container is set to allow public access and that it contains at least one blob.
- In your browser, navigate to the address of the blob using the custom domain. For example, if your custom domain is `cdn.contoso.com`, the URL to a cached blob will be similar to the following URL:  
<http://cdn.contoso.com/mypubliccontainer/acachedblob.jpg>

## See Also

[How to Enable the Content Delivery Network \(CDN\) for Azure](#)



# Restrict Azure CDN content by country

1/25/2017 • 2 min to read • [Edit Online](#)

## Overview

When a user requests your content, by default, the content is served regardless of where the user made this request from. In some cases, you may want to restrict access to your content by country. This topic explains how to use the **Geo-Filtering** feature in order to configure the service to allow or block access by country.

### IMPORTANT

The Verizon and Akamai products provide the same geo-filtering functionality but have a small difference in the country codes they support. See Step 3 for a link to the differences.

For information about considerations that apply to configuring this type of restriction, see the [Considerations](#) section at the end of the topic.

Save Discard

Add a set of paths below to block or allow the content in the selected countries ⓘ

PATH	ACTION	COUNTRY CODES
/myfolder/	Allow	US ...
/myfolder/file.exe	Allow	US,CA ...

/pictures/ or /pictures/city. Allow 0 selected ...

## Step 1: Define the directory path

Select your endpoint within the portal, and find the Geo-Filtering tab on the left-hand navigation to find this feature.

When configuring a country filter, you must specify the relative path to the location to which users will be allowed or denied access. You can apply geo-filtering for all your files with "/" or selected folders by specifying directory paths "/pictures/". You can also apply geo-filtering to a single file by specifying the file, and leaving out the trailing slash "/pictures/city.png".

Example directory path filter:

```
/
/Photos/
/Photos/Strasbourg/
/Photos/Strasbourg/city.png
```

## Step 2: Define the action: block or allow

**Block:** Users from the specified countries will be denied access to assets requested from that recursive path. If no other country filtering options have been configured for that location, then all other users will be allowed access.

**Allow:** Only users from the specified countries will be allowed access to assets requested from that recursive path.

## Step 3: Define the countries

Select the countries that you want to block or allow for the path.

For example, the rule of blocking /Photos/Strasbourg/ will filter files including:

```
http://<endpoint>.azureedge.net/Photos/Strasbourg/1000.jpg  
http://<endpoint>.azureedge.net/Photos/Strasbourg/Cathedral/1000.jpg
```

### Country codes

The **Geo-Filtering** feature uses country codes to define the countries from which a request will be allowed or blocked for a secured directory. You will find the country codes in [Azure CDN Country Codes](#).

## Considerations

- It may take up to 90 minutes for Verizon, or a couple minutes with Akamai, for changes to your country filtering configuration to take effect.
- This feature does not support wildcard characters (for example, '\*').
- The geo-filtering configuration associated with the relative path will be applied recursively to that path.
- Only one rule can be applied to the same relative path (you cannot create multiple country filters that point to the same relative path. However, a folder may have multiple country filters. This is due to the recursive nature of country filters. In other words, a subfolder of a previously configured folder can be assigned a different country filter.

# Improve performance by compressing files in Azure CDN

1/25/2017 • 3 min to read • [Edit Online](#)

Compression is a simple and effective method to improve file transfer speed and increase page load performance by reducing file size before it is sent from the server. It reduces bandwidth costs and provides a more responsive experience for your users.

There are two ways to enable compression:

- You can enable compression on your origin server, in which case the CDN will pass through the compressed files and deliver compressed files to clients that request them.
- You can enable compression directly on CDN edge servers, in which case the CDN will compress the files and serve it to end users, even if they are not compressed by the origin server.

## IMPORTANT

CDN configuration changes take some time to propagate through the network. For **Azure CDN from Akamai** profiles, propagation usually completes in under one minute. For **Azure CDN from Verizon** profiles, you will usually see your changes apply within 90 minutes. If this is the first time you've set up compression for your CDN endpoint, you should consider waiting 1-2 hours to be sure the compression settings have propagated to the POPs before troubleshooting

## Enabling compression

### NOTE

The Standard and Premium CDN tiers provide the same compression functionality, but the user interface differs. For more information about the differences between Standard and Premium CDN tiers, see [Azure CDN Overview](#).

### Standard tier

### NOTE

This section applies to **Azure CDN Standard from Verizon** and **Azure CDN Standard from Akamai** profiles.

1. From the CDN profile blade, click the CDN endpoint you wish to manage.

The screenshot shows the 'CdnDemo' CDN profile in the Azure portal. The 'Endpoints' section displays a table with four endpoints, all of which are 'Running' and support 'HTTP, HTTPS' protocols. A red box highlights the entire table.

HOSTNAME	STATUS	PROTOCOL
cdndocdemo.azureedge.net	Running	HTTP, HTTPS
cdndocdemoarm.azureedge...	Running	HTTP, HTTPS
cdndocdemoarma.azureedg...	Running	HTTP, HTTPS
cdndocdemoarmb.azureedg...	Running	HTTP, HTTPS

The CDN endpoint blade opens.

- Click the **Configure** button.

The screenshot shows the 'cdndocdemo.azureedge.net' endpoint blade. The 'Configure' button, represented by a gear icon, is highlighted with a red box. Other buttons visible include 'Custom domain', 'Stop', 'Purge', 'Load', and 'Delete'.

The CDN Configuration blade opens.

- Turn on **Compression**.

The screenshot shows the 'Compression' configuration blade. The 'On' toggle is selected. Below it, a list of 'Formats to compress' is shown, including 'application/x-javascript', 'text/css', 'text/html', 'text/javascript', and 'text/plain'. Each format has a three-dot menu icon to its right.

4. Use the default types, or modify the list by removing or adding file types.

**TIP**

While possible, it is not recommended to apply compression to compressed formats, such as ZIP, MP3, MP4, JPG, etc.

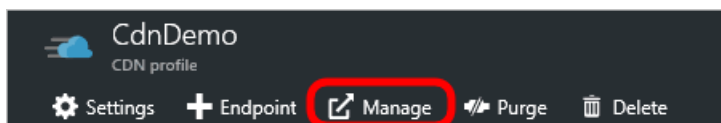
5. After making your changes, click the **Save** button.

### Premium tier

**NOTE**

This section applies to **Azure CDN Premium from Verizon** profiles.

1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **HTTP Large** tab, then hover over the **Cache Settings** flyout. Click on **Compression**.

Compression options are displayed.

A screenshot of the 'Cache Settings' flyout in the Azure CDN management portal. It shows two radio buttons: 'Compression Disabled' and 'Compression Enabled', with the latter being selected. Below the radio buttons is a 'File Types' section with a text input field containing the text 'text/plain,text/html,text/css,application/x-javascript,text/javascript'. At the bottom of the flyout, it shows 'Last Updated: 4/20/2016 8:45:59 PM' and 'Expected Complete Time (1 Hour): 4/20/2016 9:45:59 PM'.

3. Enable compression by clicking the **Compression Enabled** radio button. Enter the MIME types you wish to compress as a comma-delimited list (no spaces) in the **File Types** textbox.

**TIP**

While possible, it is not recommended to apply compression to compressed formats, such as ZIP, MP3, MP4, JPG, etc.

4. After making your changes, click the **Update** button.

## Compression rules

These tables describe Azure CDN compression behavior for every scenario.

## IMPORTANT

For **Azure CDN from Verizon** (Standard and Premium), only eligible files are compressed. To be eligible for compression, a file must:

- Be larger than 128 bytes.
- Be smaller than 1 MB.

For **Azure CDN from Akamai**, all files are eligible for compression.

For all Azure CDN products, a file must be a MIME type that has been [configured for compression](#).

**Azure CDN from Verizon** profiles (Standard and Premium) support **gzip**, **deflate**, or **bzip2** encoding. **Azure CDN from Akamai** profiles only support **gzip** encoding.

**Azure CDN from Akamai** endpoints always request **gzip** encoded files from the origin, regardless of the client request.

## Compression disabled or file is ineligible for compression

CLIENT REQUESTED FORMAT (VIA ACCEPT-ENCODING HEADER)	CACHED FILE FORMAT	CDN RESPONSE TO THE CLIENT	NOTES
Compressed	Compressed	Compressed	
Compressed	Uncompressed	Uncompressed	
Compressed	Not cached	Compressed or Uncompressed	Depends on origin response
Uncompressed	Compressed	Uncompressed	
Uncompressed	Uncompressed	Uncompressed	
Uncompressed	Not cached	Uncompressed	

## Compression enabled and file is eligible for compression

CLIENT REQUESTED FORMAT (VIA ACCEPT-ENCODING HEADER)	CACHED FILE FORMAT	CDN RESPONSE TO THE CLIENT	NOTES
Compressed	Compressed	Compressed	CDN transcodes between supported formats
Compressed	Uncompressed	Compressed	CDN performs compression
Compressed	Not cached	Compressed	CDN performs compression if origin returns uncompressed. <b>Azure CDN from Verizon</b> will pass the uncompressed file on the first request and then compress and cache the file for subsequent requests. Files with <code>Cache-Control: no-cache</code> header will never be compressed.

CLIENT REQUESTED FORMAT (VIA ACCEPT-ENCODING HEADER)	CACHED FILE FORMAT	CDN RESPONSE TO THE CLIENT	NOTES
Uncompressed	Compressed	Uncompressed	CDN performs decompression
Uncompressed	Uncompressed	Uncompressed	
Uncompressed	Not cached	Uncompressed	

## Media Services CDN Compression

For Media Services CDN enabled streaming endpoints, compression is enabled by default for the following content types: application/vnd.ms-sstr+xml, application/dash+xml,application/vnd.apple.mpegurl, application/f4m+xml. You cannot enable/disable compression for the mentioned types using the Azure portal.

## See also

- [Troubleshooting CDN file compression](#)

# Control Azure CDN caching behavior with query strings

1/25/2017 • 1 min to read • [Edit Online](#)

## Overview

Query string caching controls how files are to be cached when they contain query strings.

### IMPORTANT

The Standard and Premium CDN products provide the same query string caching functionality, but the user interface differs. This document describes the interface for **Azure CDN Standard from Akamai** and **Azure CDN Standard from Verizon**. For query string caching with **Azure CDN Premium from Verizon**, see [Controlling caching behavior of CDN requests with query strings - Premium](#).

Three modes are available:

- **Ignore query strings:** This is the default mode. The CDN edge node will pass the query string from the requestor to the origin on the first request and cache the asset. All subsequent requests for that asset that are served from the edge node will ignore the query string until the cached asset expires.
- **Bypass caching for URL with query strings:** In this mode, requests with query strings are not cached at the CDN edge node. The edge node retrieves the asset directly from the origin and passes it to the requestor with each request.
- **Cache every unique URL:** This mode treats each request with a query string as a unique asset with its own cache. For example, the response from the origin for a request for *foo.ashx?q=bar* would be cached at the edge node and returned for subsequent caches with that same query string. A request for *foo.ashx?q=somethingelse* would be cached as a separate asset with its own time to live.

## Changing query string caching settings for standard CDN profiles

1. From the CDN profile blade, click the CDN endpoint you wish to manage.



**CdnDemo**  
CDN profile

Settings Endpoint Manage Purge Delete

Essentials ^

Resource group: CdnDemoRG  
Status: Active  
Location: Central US  
Subscription name: msdn  
Subscription ID: <Subscription ID>

Pricing tier: Standard Verizon

All settings →

Add tiles (+)

Endpoints

4

HOSTNAME	STATUS	PROTOCOL
cdndocdemo.azureedge.net	Running	HTTP, HTTPS
cdndocdemoarm.azureedge...	Running	HTTP, HTTPS
cdndocdemoarma.azureedg...	Running	HTTP, HTTPS
cdndocdemoarmb.azureedg...	Running	HTTP, HTTPS

The CDN endpoint blade opens.

- Click the **Configure** button.

**cdndocdemo.azureedge.net**  
Endpoint

+ Custom domain **Configure** Stop Purge Load Delete

The CDN Configuration blade opens.

- Select a setting from the **Query string caching behavior** dropdown.

Configure

Save Discard

Compression ⓘ

Off On

Query string caching behavior ⓘ

Ignore query strings  
Bypass caching for query strings  
Cache every unique URL

Protocols ⓘ

☒ HTTP  
☒ HTTPS

Origin host header ⓘ

cdndocdemo.blob.core.windows.net

Origin path ⓘ

/Path

4. After making your selection, click the **Save** button.

#### IMPORTANT

The settings changes may not be immediately visible, as it takes time for the registration to propagate through the CDN. For **Azure CDN from Akamai** profiles, propagation will usually complete within one minute. For **Azure CDN from Verizon** profiles, propagation will usually complete within 90 minutes, but in some cases can take longer.

# Control Azure CDN caching behavior with query strings - Premium

1/25/2017 • 1 min to read • [Edit Online](#)

## Overview

Query string caching controls how files are to be cached when they contain query strings.

### IMPORTANT

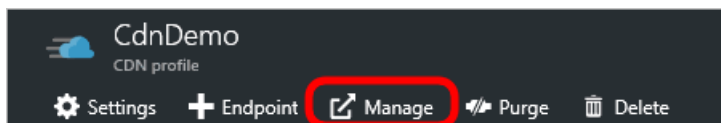
The Standard and Premium CDN products provide the same query string caching functionality, but the user interface differs. This document describes the interface for **Azure CDN Premium from Verizon**. For query string caching with **Azure CDN Standard from Akamai** and **Azure CDN Standard from Verizon**, see [Controlling caching behavior of CDN requests with query strings](#).

Three modes are available:

- **standard-cache**: This is the default mode. The CDN edge node will pass the query string from the requestor to the origin on the first request and cache the asset. All subsequent requests for that asset that are served from the edge node will ignore the query string until the cached asset expires.
- **no-cache**: In this mode, requests with query strings are not cached at the CDN edge node. The edge node retrieves the asset directly from the origin and passes it to the requestor with each request.
- **unique-cache**: This mode treats each request with a query string as a unique asset with its own cache. For example, the response from the origin for a request for *foo.ashx?q=bar* would be cached at the edge node and returned for subsequent caches with that same query string. A request for *foo.ashx?q=somethingelse* would be cached as a separate asset with its own time to live.

## Changing query string caching settings for premium CDN profiles

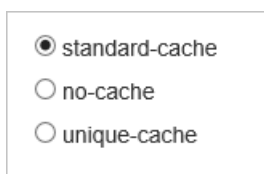
1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **HTTP Large** tab, then hover over the **Cache Settings** flyout. Click on **Query-String Caching**.

Query string caching options are displayed.



3. After making your selection, click the **Update** button.

#### IMPORTANT

The settings changes may not be immediately visible, as it takes time for the registration to propagate through the CDN. For **Azure CDN from Verizon** profiles, propagation will usually complete within 90 minutes, but in some cases can take longer.

# Purge an Azure CDN endpoint

2/13/2017 • 2 min to read • [Edit Online](#)

## Overview

Azure CDN edge nodes will cache assets until the asset's time-to-live (TTL) expires. After the asset's TTL expires, when a client requests the asset from the edge node, the edge node will retrieve a new updated copy of the asset to serve the client request and store refresh the cache.

The best practice to make sure your users always obtain the latest copy of your assets is to version your assets for each update and publish them as new URLs. CDN will immediately retrieve the new assets for the next client requests. Sometimes you may wish to purge cached content from all edge nodes and force them all to retrieve new updated assets. This might be due to updates to your web application, or to quickly update assets that contain incorrect information.

### TIP

Note that purging only clears the cached content on the CDN edge servers. Any downstream caches, such as proxy servers and local browser caches, may still hold a cached copy of the file. It's important to remember this when you set a file's time-to-live. You can force a downstream client to request the latest version of your file by giving it a unique name every time you update it, or by taking advantage of [query string caching](#).

This tutorial walks you through purging assets from all edge nodes of an endpoint.

## Walkthrough

1. In the [Azure Portal](#), browse to the CDN profile containing the endpoint you wish to purge.
2. From the CDN profile blade, click the purge button.



The Purge blade opens.

3. On the Purge blade, select the service address you wish to purge from the URL dropdown.

#### NOTE

You can also get to the Purge blade by clicking the **Purge** button on the CDN endpoint blade. In that case, the **URL** field will be pre-populated with the service address of that specific endpoint.

4. Select what assets you wish to purge from the edge nodes. If you wish to clear all assets, click the **Purge all** checkbox. Otherwise, type the path of each asset you wish to purge in the **Path** textbox. Below formats are supported in the path.
  - a. **Single URL purge:** Purge individual asset by specifying the full URL, with or without the file extension, e.g., `/pictures/strasbourg.png` ; `/pictures/strasbourg`
  - b. **Wildcard purge:** Asterisk (\*) may be used as a wildcard. Purge all folders, sub-folders and files under an endpoint with `/*` in the path or purge all sub-folders and files under a specific folder by specifying the folder followed by `/*`, e.g., `/pictures/*` . Note that wildcard purge is not supported by Azure CDN from Akamai currently.
  - c. **Root domain purge:** Purge the root of the endpoint with `/` in the path.

#### TIP

Paths must be specified for purge and must be a relative URL that fit the following [regular expression](#). **Purge all** and **Wildcard purge** not supported by **Azure CDN from Akamai** currently.

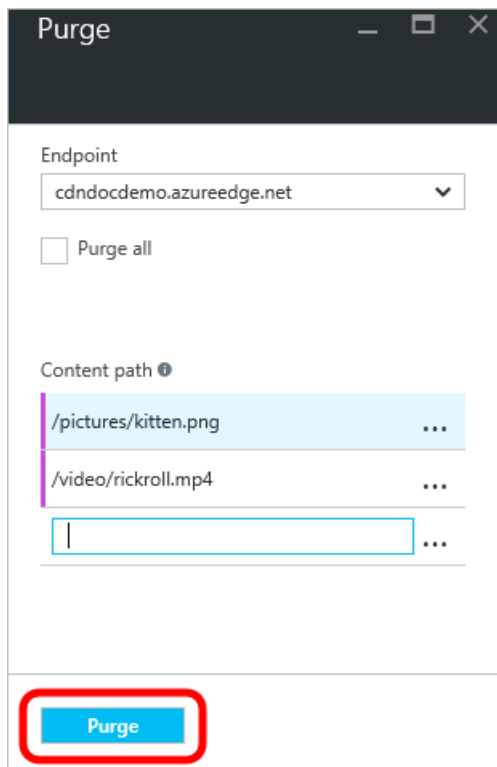
Single URL purge `@"^\/(?:[a-zA-Z0-9-_.%=\(\)\u0020]+\/?)*$";`

Query string `@"^(?:\?[-\@_a-zA-Z0-9\/%:;=!\.\\+'&\(\)\u0020]*)?$";`

Wildcard purge `@"^\/(?:[a-zA-Z0-9-_.%=\(\)\u0020]+\/?)*\*$";` .

More **Path** textboxes will appear after you enter text to allow you to build a list of multiple assets. You can delete assets from the list by clicking the ellipsis (...) button.

5. Click the **Purge** button.



Purge

Endpoint  
cdndocdemo.azureedge.net

☐ Purge all

Content path ⓘ

- /pictures/kitten.png ...
- /video/rickroll.mp4 ...
- ...

Purge

#### IMPORTANT

Purge requests take approximately 2-3 minutes to process with **Azure CDN from Verizon** (Standard and Premium), and approximately 7 minutes with **Azure CDN from Akamai**. Azure CDN has a limit of 50 concurrent purge requests at any given time.

## See also

- [Pre-load assets on an Azure CDN endpoint](#)
- [Azure CDN REST API reference - Purge or Pre-Load an Endpoint](#)

# Pre-load assets on an Azure CDN endpoint

3/3/2017 • 1 min to read • [Edit Online](#)

## IMPORTANT

This feature is available with **Azure CDN from Verizon** products (Standard and Premium). It is not supported on **Azure CDN from Akamai**. For a comparison of CDN features, see [Azure CDN Overview](#).

By default, assets are first cached as they are requested. This means that the first request from each region may take longer, since the edge servers will not have the content cached and will need to forward the request to the origin server. Pre-loading content avoids this first hit latency.

In addition to providing a better customer experience, pre-loading your cached assets can also reduce network traffic on the origin server.

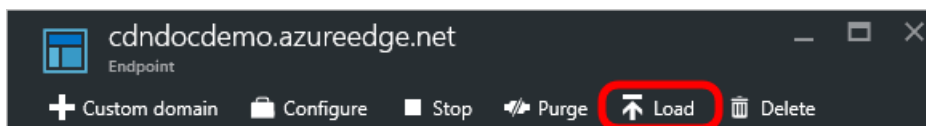
## NOTE

Pre-loading assets is useful for large events or content that becomes simultaneously available to a large number of users, such as a new movie release or a software update.

This tutorial walks you through pre-loading cached content on all Azure CDN edge nodes.

## Walkthrough

1. In the [Azure Portal](#), browse to the CDN profile containing the endpoint you wish to pre-load. The profile blade opens.
2. Click the endpoint in the list. The endpoint blade opens.
3. From the CDN endpoint blade, click the load button.



The Load blade opens.



Load

Endpoint  
cdndocdemo.azureedge.net

Content path ⓘ  
...

Load

- Enter the full path of each asset you wish to load (e.g., `/pictures/kitten.png`) in the **Path** textbox.

#### TIP

More **Path** textboxes will appear after you enter text to allow you to build a list of multiple assets. You can delete assets from the list by clicking the ellipsis (...) button.

Paths must be a relative URL that fits the following [regular expression](#):

Load a single file path `@\"^(?:\\/[a-zA-Z0-9-_.%=\u0020]+)$\" ;`

Load a single file with query string `@\"^(?:\\?[-_a-zA-Z0-9\\/%:;=!,.\\+ '&\u0020]*)?$\" ;`

Each asset must have its own path. There is no wildcard functionality for pre-loading assets.

Path ⓘ

/pictures/kitten.png ...

/video/rickroll.mp4 ...

...

- Click the **Load** button.

Load

#### NOTE

There is a limitation of 10 load requests per minute per CDN profile.

## See also

- [Purge an Azure CDN endpoint](#)
- [Azure CDN REST API reference - Purge or Pre-Load an Endpoint](#)

# Securing Azure CDN assets with token authentication

3/28/2017 • 5 min to read • [Edit Online](#)

## IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

## Overview

Token authentication is a mechanism that allows you to prevent Azure CDN from serving assets to unauthorized clients. This is typically done to prevent "hotlinking" of content, where a different website, often a message board, uses your assets without permission. This can have an impact on your content delivery costs. By enabling this feature on CDN, requests will be authenticated by CDN edge POPs before delivering the content.

## How it works

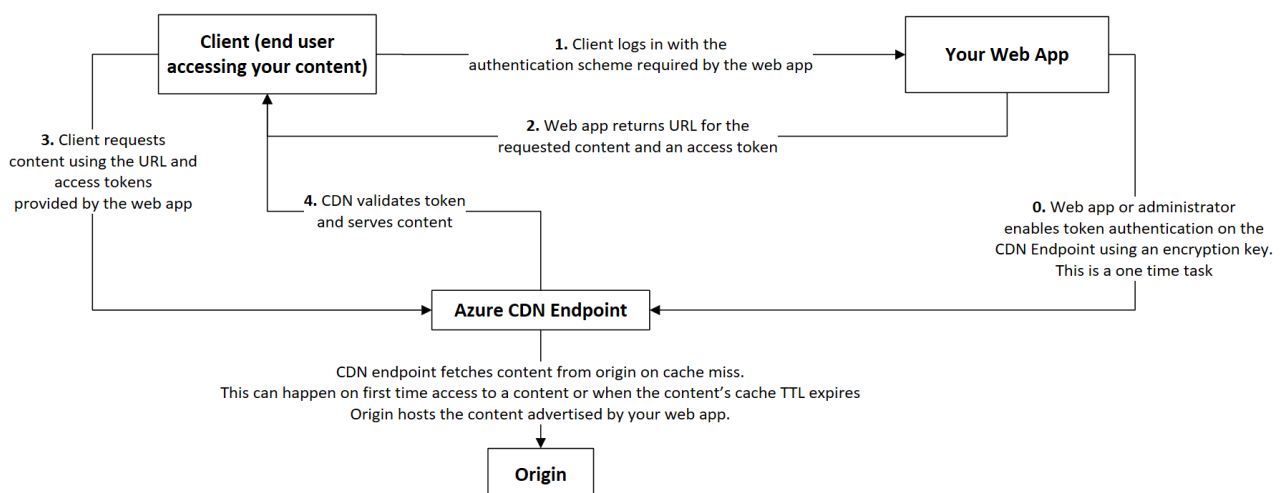
Token authentication verifies requests are generated by a trusted site by requiring requests to contain a token value containing encoded information about the requester. Content will only be served to requester when the encoded information meet the requirements, otherwise requests will be denied. You can set up the requirement using one or multiple parameters below.

- Country: allow or deny requests that originated from specified countries. [List of valid country codes](#).
- URL: only allow specified asset or path to request.
- Host: allow or deny requests using specified hosts in the request header.
- Referrer: allow or deny specified referrer to request.
- IP address: only allow requests that originated from specific IP address or IP subnet.
- Protocol: allow or block requests based on the protocol used to request the content.
- Expiration time: assign a date and time period to ensure that a link only remains valid for a limited time period.

See detailed configuration example of each parameter.

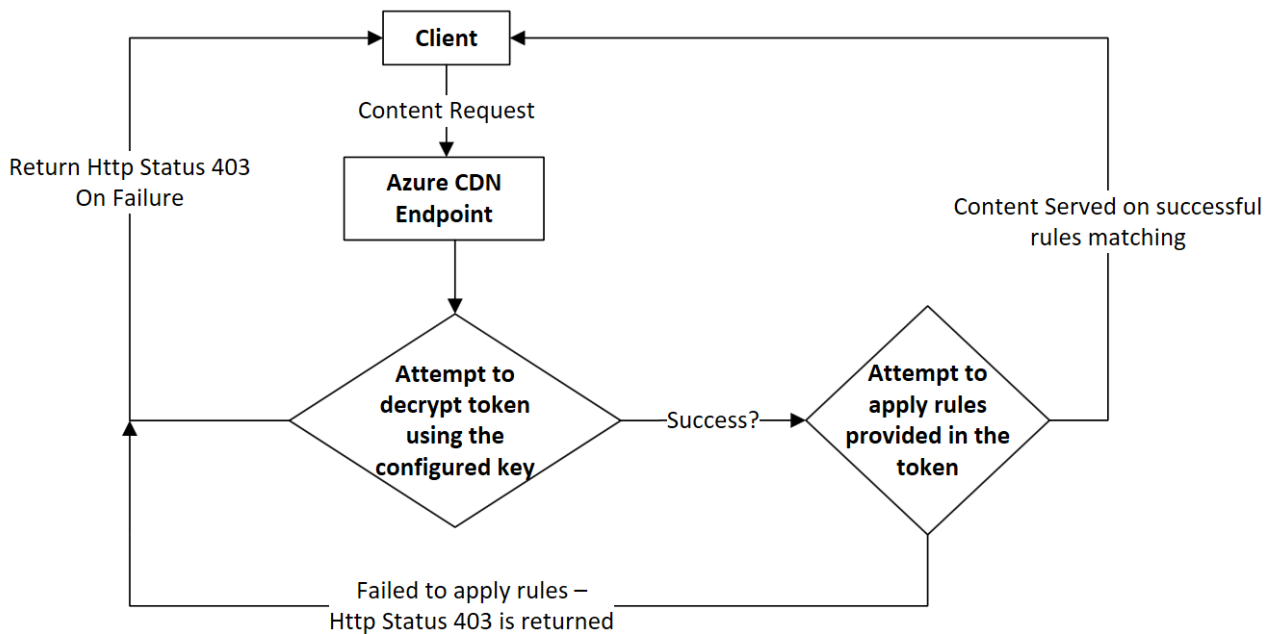
## Reference architecture

See below a reference architecture of setting up token authentication on CDN to work with your Web App.



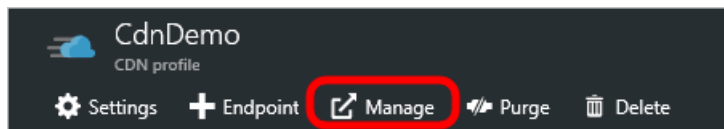
# Token validation logic on CDN endpoint

This chart describes how Azure CDN validates client request when token authentication is configured on CDN endpoint.



## Setting up token authentication

1. From the [Azure portal](#), browse to your CDN profile, and then click the **Manage** button to launch the supplemental portal.



2. Hover over **HTTP Large**, and then click **Token Auth** in the flyout. You will set up encryption key and encryption parameters in this tab.

- a. Enter a unique encryption key for **Primary Key**. Enter another for **Backup Key**

The screenshot shows the 'HTTP Large Object Token-Based Authentication' configuration page. It includes fields for 'Primary Key' (containing '12343') and 'Backup Key'. There are also dropdown menus for 'Minimum Encryption Version' (set to 'V3' for Primary and 'V2' for Backup). An 'Update' button is located at the bottom right.

- b. Set up encryption parameters with encryption tool (allow or deny requests based on expiration time, country, referrer, protocol, client IP. You can use any combination.)

- **ec-expire:** assigns an expiration time of a token after a specified time period. Requests submitted after the expiration time will be denied. This parameter uses Unix timestamp (based on seconds since standard epoch of 1/1/1970 00:00:00 GMT. You can use online tools to provide conversion between standard time and Unix time.) For example, If you want to set up the token to be expired at 12/31/2016 12:00:00 GMT, use the Unix time:1483185600 as below:

**ec\_expire:**

- **ec-url-allow:** allows you to tailor tokens to a particular asset or path. It restricts access to requests whose URL start with a specific relative path. You can input multiple paths separating each path with a comma. URLs are case-sensitive. Depending on the requirement, you can set up different value to provide different level of access. Below are a couple of scenarios:

If you have a URL: <http://www.mydomain.com/pictures/city/strasbourg.png>. See input value "" and its access level accordingly

- Input value "/": all requests will be allowed
- Input value "/pictures": all the following requests will be allow
  - <http://www.mydomain.com/pictures.png>
  - <http://www.mydomain.com/pictures/city/strasbourg.png>
  - <http://www.mydomain.com/picturesnew/city/strasbourgh.png>
- Input value "/pictures/": only requests for /pictures/ will be allowed
- Input value "/pictures/city/strasbourg.png": only allows request for this asset

**ec\_proto\_allow:**

- **ec-country-allow:** only allows requests that originate from one or more specified countries. Requests that originate from all other countries will be denied. Use country code to set up the parameters and separating each country code with a comma. For example, If you want to allow access from United States and France, input US, FR in the column as below.

**ec\_country\_allow:**

- **ec-country-deny:** denies requests that originated from one or more specified countries. Requests that originate from all other countries will be allowed. Use country code to set up the parameters and separating each country code with a comma. For example, If you want to deny access from United States and France, input US, FR in the column.
- **ec-ref-allow:** only allows requests from specified referrer. A referrer identifies the URL of the web page that linked to the resource being requested. The referrer parameter value shouldn't include the protocol. You can input a hostname and/or a particular path on that hostname. You can also add multiple referrers within a single parameter separating each one with a comma. If

you have specified a referrer value, but the referrer information is not sent in the request due to some browser configuration, these requests will be denied by default. You can assign "Missing" or a blank value in the parameter to allow these requests with missing referrer information. You can also use "\*.consoto.com" to allow all subdomains of consoto.com. For example, if you want to allow access for requests from www.consoto.com, all subdomains under consoto2.com and erquests with blank or missing reffers, input value below:

ec\_ref\_allow:

- ec-ref-deny: denies requests from specified referrer. Refer to details and example in "ec-ref-allow" parameter.
- ec-proto-allow: only allows requests from specified protocol. For example, http or https.

ec\_proto\_allow:

- ec-proto-deny: denies requests from specified protocol. For example, http or https.
- ec-clientip: restricts access to specified requester's IP address. Both IPV4 and IPV6 are supported. You can specify single request IP address or IP subnet.

ec\_clientip:

c. You can test your token with the decryption tool.

d. You can also customize the type of response that will be returned to user when request is denied. By default we use 403.

3. Now click **Rules Engine** tab under **HTTP Large**. You will use this tab to define paths to apply the feature, enable the token authentication feature, and enable additional token authentication related capabilities.

- Use "IF" column to define asset or path that you want to apply token authentication.
- Click to add "Token Auth" from the feature dropdown to enable token authentication.

The screenshot shows a configuration interface for a rule. At the top, there's a section with a red 'x' icon, a blue 'i' icon, and a dropdown menu set to 'IF'. Below this is a row of fields: 'URL Path Directory' (dropdown), 'Matches' (dropdown), 'Value' (text input with '/pics/'), 'Ignore Case' (checkbox), and 'Relative To' (dropdown set to 'Origin'). Below this row is a 'Features' section with a '+' button, a red 'x' icon, a blue 'i' icon, a dropdown menu set to 'Token Auth', and an 'Enabled' dropdown set to 'Enabled'.

4. In the **Rules Engine** tab, there are a few additional capabilities you can enable.

- Token auth denial code: determines the type of response that will be returned to user when a request is denied. Rules set up here will override the denial code setting in the token auth tab.
- Token auth ignore: determines whether URL used to validate token will be case sensitive.
- Token auth parameter: rename the token auth query string parameter showing in the requested URL.

The screenshot shows the same configuration interface as before, but with additional settings. Below the 'Token Auth' feature, there are four more rows, each with a blue 'i' icon, a dropdown menu, and an 'Enabled' dropdown. The first row is 'Token Auth Denial Code' with a value of '301' and an 'Optional Header Name' of 'WWW-Authenticate' and an 'Optional Header Value' of 'abctest'. The second row is 'Token Auth Ignore URL Case' with a value of 'Enabled'. The third row is 'Token Auth Parameter' with a value of 'Token' and an 'Enabled' dropdown set to 'Enabled'.

5. You can customize your token which is an application that generates token for Token-based authentication features. Source code can be accessed here in [GitHub](#). Available languages include:

- C
- C#
- PHP
- Perl

- Java
- Python

## Azure CDN features and provider pricing

See the [CDN Overview](#) topic.

# Monitor the health of Azure CDN resources

1/25/2017 • 1 min to read • [Edit Online](#)

Azure CDN Resource health is a subset of [Azure resource health](#). You can use Azure resource health to monitor the health of CDN resources and receive actionable guidance to troubleshoot problems.

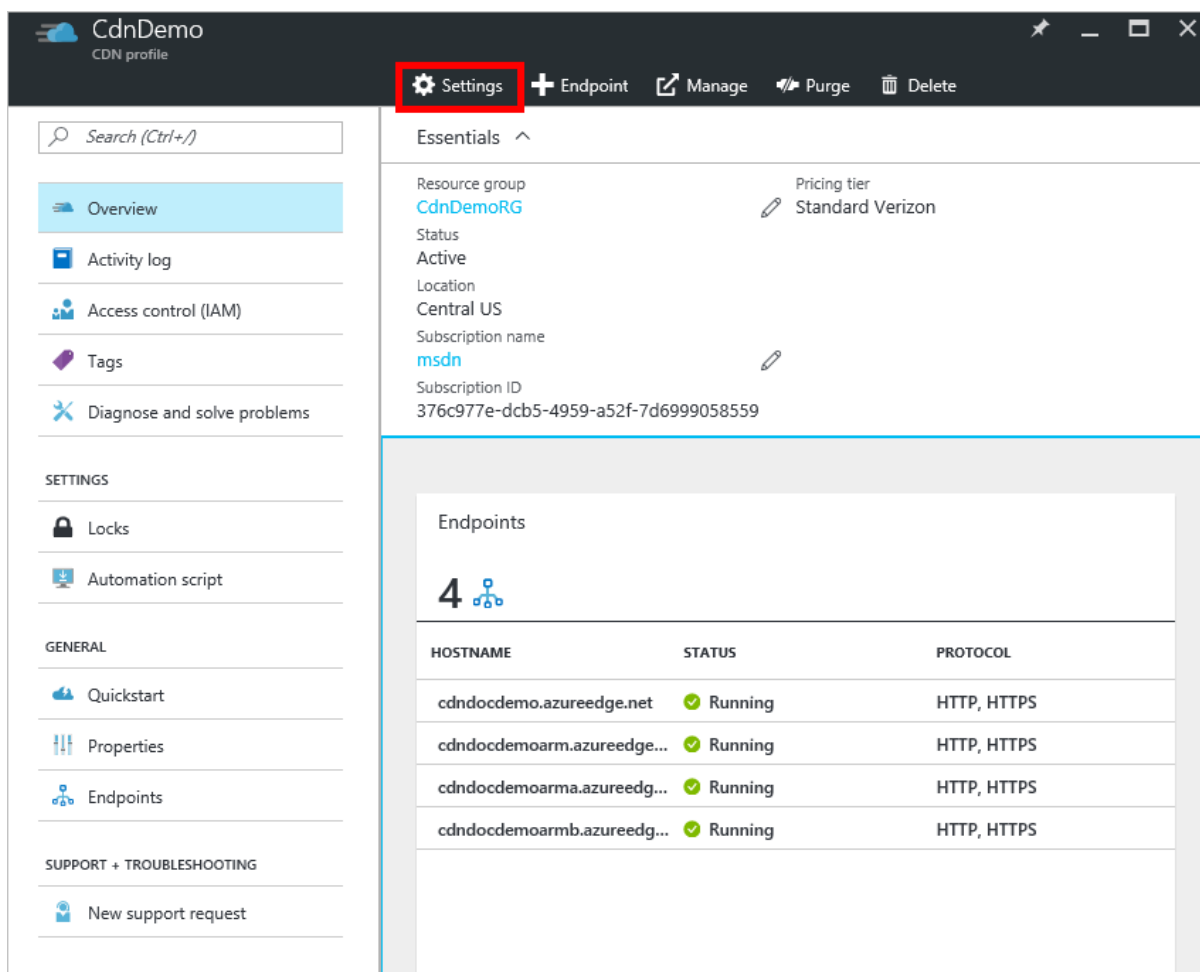
## IMPORTANT

Azure CDN resource health only currently accounts for the health of global CDN delivery and API capabilities. Azure CDN resource health does not verify individual CDN endpoints.

The signals that feed Azure CDN resource health may be up to 15 minutes delayed.

## How to find Azure CDN resource health

1. In the [Azure portal](#), browse to your CDN profile.
2. Click the **Settings** button.



The screenshot shows the Azure portal interface for a CDN profile named 'CdnDemo'. The 'Settings' button in the top navigation bar is highlighted with a red box. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Locks, Automation script, Quickstart, Properties, Endpoints, and New support request. The main content area displays the 'Essentials' section with various settings and a table of endpoints.

HOSTNAME	STATUS	PROTOCOL
cdndocdemo.azureedge.net	Running	HTTP, HTTPS
cdndocdemoarm.azureedge...	Running	HTTP, HTTPS
cdndocdemoarma.azureedg...	Running	HTTP, HTTPS
cdndocdemoarmb.azureedg...	Running	HTTP, HTTPS

3. Under *Support + troubleshooting*, click **Resource health**.




 Search (Ctrl+J)

 Overview

 Activity log


 Access control (IAM)

 Tags


 Diagnose and solve problems


#### SETTINGS

 Locks

 Automation script


#### GENERAL


 Quickstart

 Properties

 Endpoints

#### SUPPORT + TROUBLESHOOTING

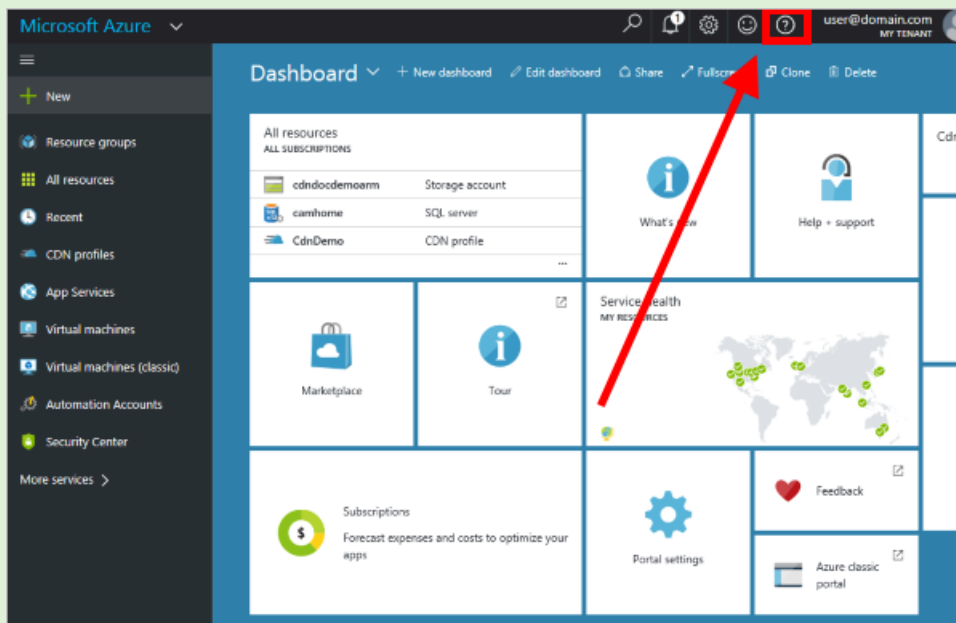
 Resource health

 New support request



## TIP

You can also find CDN resources listed in the *Resource health* tile in the *Help + support* blade. You can quickly get to *Help + support* by clicking the circled ? in the upper right corner of the portal.



## Azure CDN-specific messages

Statuses related to Azure CDN resource health can be found below.

MESSAGE	RECOMMENDED ACTION
You may have stopped, removed, or misconfigured one or more of your CDN endpoints	You may have stopped, removed, or misconfigured one or more of your CDN endpoints.
We are sorry, the CDN management service is currently unavailable	Check back here for status updates; If your problem persists after the expected resolution time, contact support.
We're sorry, your CDN endpoints may be impacted by ongoing issues with some of our CDN providers	Check back here for status updates; Use the Troubleshoot tool to learn how to test your origin and CDN endpoint; If your problem persists after the expected resolution time, contact support.
We're sorry, CDN endpoint configuration changes are experiencing propagation delays	Check back here for status updates; If your configuration changes are not fully propagated in the expected time, contact support.
We're sorry, we are experiencing issues loading the supplemental portal	Check back here for status updates; If your problem persists after the expected resolution time, contact support.
We are sorry, we are experiencing issues with some of our CDN providers	Check back here for status updates; If your problem persists after the expected resolution time, contact support.

## Next steps

- [Read an overview of Azure resource health](#)
- [Troubleshoot issues with CDN compression](#)
- [Troubleshoot issues with 404 errors](#)



# Override HTTP behavior using the Azure CDN rules engine

1/25/2017 • 2 min to read • [Edit Online](#)

## IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

## Overview

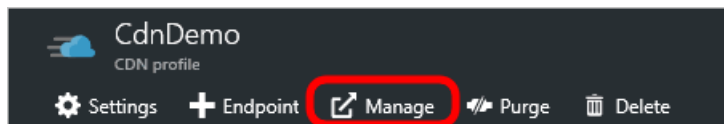
The rules engine allows you to customize how HTTP requests are handled, such as blocking the delivery of certain types of content, defining a caching policy, and modifying HTTP headers. This tutorial will demonstrate creating a rule that will change the caching behavior of CDN assets. There's also video content available in the "See also" section.

## TIP

For a reference to the syntax in detail, see [Rules Engine Reference](#).

## Tutorial

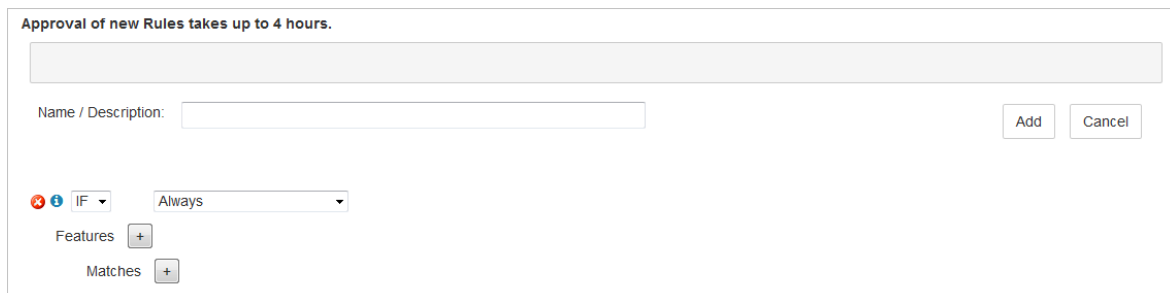
1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Click on the **HTTP Large** tab, followed by **Rules Engine**.

Options for a new rule are displayed.

A screenshot of a dialog box titled 'Approval of new Rules takes up to 4 hours.' It contains a text input field for 'Name / Description:' with 'Add' and 'Cancel' buttons to its right. Below this is a section for rule configuration: a red 'X' icon, a blue 'i' icon, an 'IF' dropdown menu, and a 'Always' dropdown menu. At the bottom, there are two sections: 'Features' with a '+' button and 'Matches' with a '+' button.

## IMPORTANT

The order in which multiple rules are listed affects how they are handled. A subsequent rule may override the actions specified by a previous rule.

3. Enter a name in the **Name / Description** textbox.

4. Identify the type of requests the rule will apply to. By default, the **Always** match condition is selected. You'll use **Always** for this tutorial, so leave that selected.



**TIP**

There are many types of match conditions available in the dropdown. Clicking on the blue informational icon to the left of the match condition will explain the currently selected condition in detail.

For the full list of conditional expressions in detail, see [Rules Engine Conditional Expressions](#).

For the full list of match conditions in detail, see [Rules Engine Match Conditions](#).

5. Click the **+** button next to **Features** to add a new feature. In the dropdown on the left, select **Force Internal Max-Age**. In the textbox that appears, enter **300**. Leave the remaining default values.



**NOTE**

As with match conditions, clicking the blue informational icon to the left of the new feature will display details about this feature. In the case of **Force Internal Max-Age**, we are overriding the asset's **Cache-Control** and **Expires** headers to control when the CDN edge node will refresh the asset from the origin. Our example of 300 seconds means the CDN edge node will cache the asset for 5 minutes before refreshing the asset from its origin.

For the full list of features in detail, see [Rules Engine Feature Details](#).

6. Click the **Add** button to save the new rule. The new rule is now awaiting approval. Once it has been approved, the status will change from **Pending XML** to **Active XML**.

**IMPORTANT**

Rules changes may take up to 90 minutes to propagate through the CDN.

## See also

- [Azure CDN Overview](#)
- [Rules Engine Reference](#)
- [Rules Engine Match Conditions](#)
- [Rules Engine Conditional Expressions](#)
- [Rules Engine Features](#)
- [Overriding default HTTP behavior using the rules engine](#)
- [Azure Fridays: Azure CDN's powerful new Premium Features](#) (video)

# Real-time alerts in Microsoft Azure CDN

1/25/2017 • 3 min to read • [Edit Online](#)

## IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

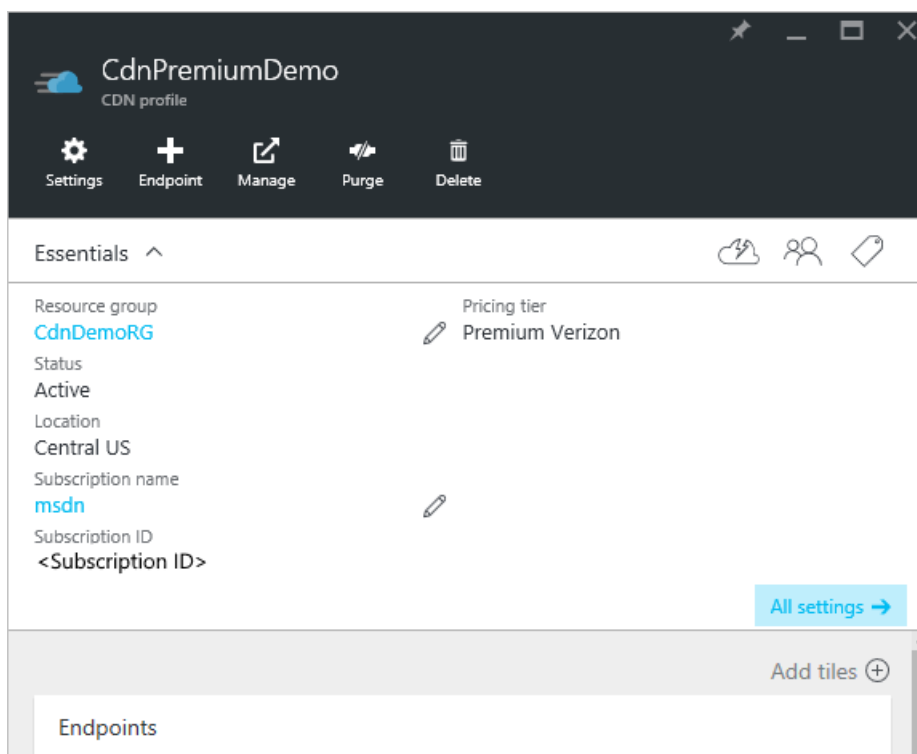
## Overview

This document explains real-time alerts in Microsoft Azure CDN. This functionality provides real-time notifications about the performance of the endpoints in your CDN profile. You can set up email or HTTP alerts based on:

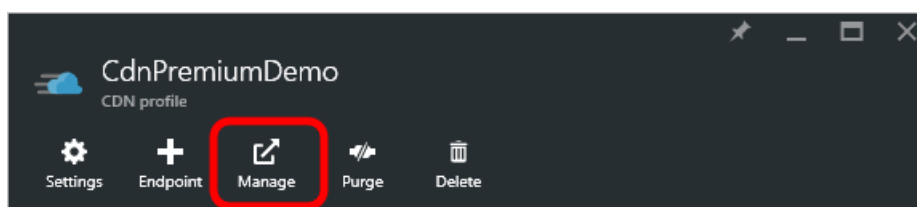
- Bandwidth
- Status Codes
- Cache Statuses
- Connections

## Creating a real-time alert

1. In the [Azure Portal](#), browse to your CDN profile.

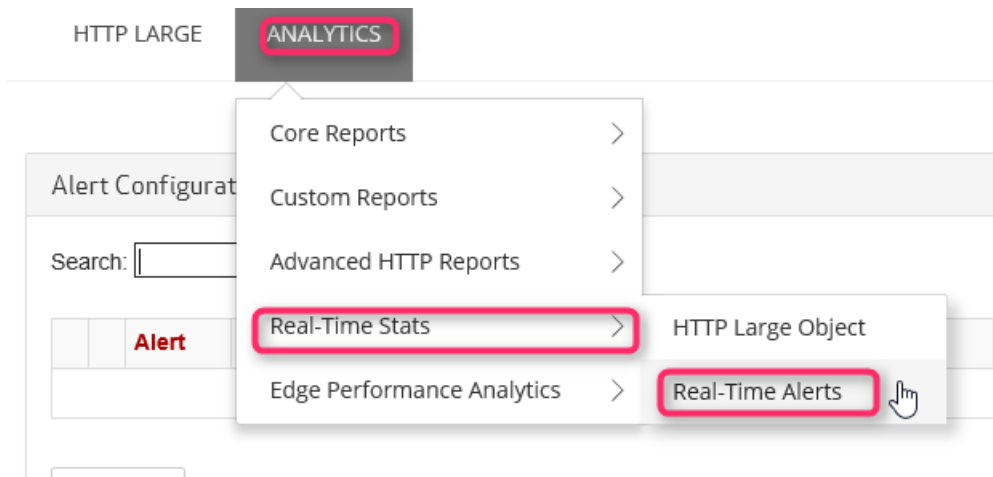


2. From the CDN profile blade, click the **Manage** button.



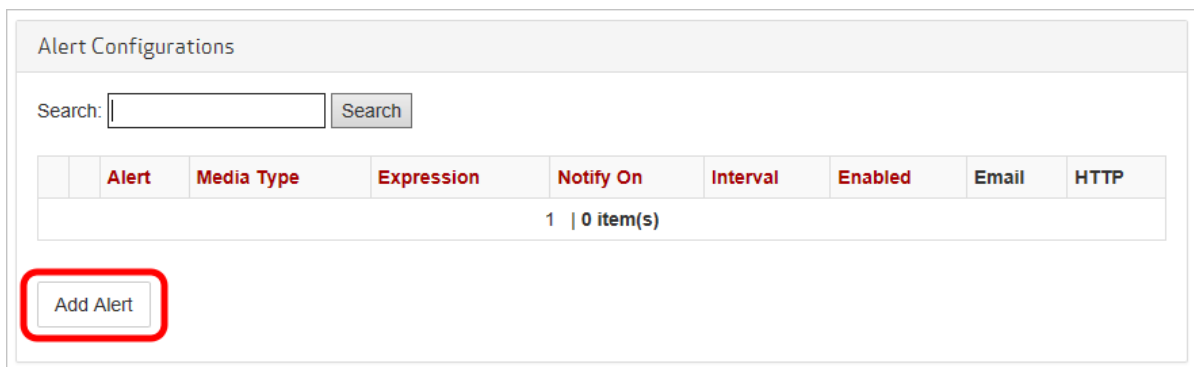
The CDN management portal opens.

3. Hover over the **Analytics** tab, then hover over the **Real-Time Stats** flyout. Click on **Real-Time Alerts**.



The list of existing alert configurations (if any) is displayed.

4. Click the **Add Alert** button.



A form for creating a new alert is displayed.

A screenshot of the form for creating a new alert. The form is titled 'Alert Configurations'. It includes a checkbox for 'Alert Enabled'. Below this are fields for 'Name:', 'Media Type:' (with a dropdown menu showing 'Flash (Live or On-Demand)'), 'Expression:' (with a dropdown menu showing 'Bandwidth Mbps', a dropdown menu showing '>', and a text input field showing '1'), 'Interval:' (with a dropdown menu showing 'Every 5 minutes'), and 'Notify on:' (with a dropdown menu showing 'Condition Start'). At the bottom, there are two expandable sections: 'Notify by Email' and 'Notify by HTTP Post'. At the very bottom, there are 'Save' and 'Cancel' buttons.

5. If you want this alert to be active when you click **Save**, check the **Alert Enabled** checkbox.
6. Enter a descriptive name for your alert in the **Name** field.
7. In the **Media Type** dropdown, select **HTTP Large Object**.

Media Type: HTTP Large Object ▼

**IMPORTANT**

You must select **HTTP Large Object** as the **Media Type**. The other choices are not used by **Azure CDN from Verizon**. Failure to select **HTTP Large Object** will cause your alert to never be triggered.

8. Create an **Expression** to monitor by selecting a **Metric**, **Operator**, and **Trigger value**.
  - For **Metric**, select the type of condition you want monitored. **Bandwidth Mbps** is the amount of bandwidth usage in megabits per second. **Total Connections** is the number of concurrent HTTP connections to our edge servers. For definitions of the various cache statuses and status codes, see [Azure CDN Cache Status Codes](#) and [Azure CDN HTTP Status Codes](#)
  - **Operator** is the mathematical operator that establishes the relationship between the metric and the trigger value.
  - **Trigger Value** is the threshold value that must be met before a notification will be sent out.

In the below example, the expression I have created indicates that I would like to be notified when the number of 404 status codes is greater than 25.

Expression: Status Code : 404 per second ▼ > 25  
Metric Operator Trigger value

9. For **Interval**, enter how frequently you would like the expression evaluated.
10. In the **Notify on** dropdown, select when you would like to be notified when the expression is true.
  - **Condition Start** indicates that a notification will be sent when the specified condition is first detected.
  - **Condition End** indicates that a notification will be sent when the specified condition is no longer detected. This notification can only be triggered after our network monitoring system detected that the specified condition occurred.
  - **Continuous** indicates that a notification will be sent each time that the network monitoring system detects the specified condition. Keep in mind that the network monitoring system will only check once per interval for the specified condition.
  - **Condition Start and End** indicates that a notification will be sent the first time that the specified condition is detected and once again when the condition is no longer detected.
11. If you want to receive notifications by email, check the **Notify by Email** checkbox.

☒ **Notify by Email**

From:

To:  \* Required

Subject:

Body:

Available keywords:

- [Name]
- [MediaType]
- [Expression]
- [Interval]
- [NotificationCondition]
- [CurrentValue]
- [TriggerValue]
- [Metric]
- [Operator]
- [AlertDuration]

In the **To** field, enter the email address you want notifications sent. For **Subject** and **Body**, you may leave the default, or you may customize the message using the **Available keywords** list to dynamically insert alert data when the message is sent.

#### NOTE

You can test the email notification by clicking the **Test Notification** button, but only after the alert configuration has been saved.

- If you want notifications to be posted to a web server, check the **Notify by HTTP Post** checkbox.

☒ **Notify by HTTP Post**

Uri:  \* Required

Headers:

Body:

Available keywords:

- [Name]
- [MediaType]
- [Expression]
- [Interval]
- [NotificationCondition]
- [CurrentValue]
- [TriggerValue]
- [Metric]
- [Operator]
- [AlertDuration]



In the **Url** field, enter the URL you want the HTTP message posted. In the **Headers** textbox, enter the HTTP headers to be sent in the request. For **Body** you may customize the message using the **Available keywords** list to dynamically insert alert data when the message is sent. **Headers** and **Body** default to an XML payload similar to the below example.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">
  <![CDATA[Expression=Status Code : 404 per second > 25&Metric=Status Code : 404 per
second&CurrentValue=[CurrentValue]&NotificationCondition=Condition Start]]>
</string>
```

#### NOTE

You can test the HTTP Post notification by clicking the **Test Notification** button, but only after the alert configuration has been saved.

13. Click the **Save** button to save your alert configuration. If you checked **Alert Enabled** in step 5, your alert is now active.

## Next Steps

- Analyze [Real-time stats in Azure CDN](#)
- Dig deeper with [advanced HTTP reports](#)
- Analyze [usage patterns](#)

# HTTP/2 Support in Azure CDN

5/5/2017 • 1 min to read • [Edit Online](#)

HTTP/2 is a major revision to HTTP/1.1. It provides faster web performance, reduced response time, and improved user experience, while maintaining the familiar HTTP methods, status codes, and semantics. Though HTTP/2 is designed to work with HTTP and HTTPS, many client web browsers only support HTTP/2 over TLS.

## HTTP/2 Benefits

The benefits of HTTP/2 include:

- **Multiplexing and concurrency**

Using HTTP 1.1, multiple making multiple resource requests requires multiple TCP connections, and each connection has performance overhead associated with it. HTTP/2 allows multiple resources to be requested on a single TCP connection.

- **Header compression**

By compressing the HTTP headers for served resources, time on the wire is reduced significantly.

- **Stream dependencies**

Stream dependencies allow the client to indicate to the server which of resources have priority.

## HTTP/2 Browser Support

All of the major browsers have implemented HTTP/2 support in their current versions. Non-supported browsers will automatically fallback to HTTP/1.1.

BROWSER	MINIMUM VERSION
Microsoft Edge	12
Google Chrome	43
Mozilla Firefox	38
Opera	32
Safari	9

## Enabling HTTP/2 Support in Azure CDN

Currently HTTP/2 support is active for **Azure CDN from Akamai** and **Azure CDN from Verizon** profiles. No further action is required from customers.

## Next Steps

To see the benefits of HTTP/2 in action, see [this demo from Akamai](#).

To learn more about HTTP/2, visit the following resources:

- [HTTP/2 specification homepage](#)

- [Official HTTP/2 FAQ](#)
- [Akamai HTTP/2 information](#)

To learn more about Azure CDN's available features, see the [Azure CDN Overview](#).

# Analyze Azure CDN usage patterns

5/11/2017 • 7 min to read • [Edit Online](#)

## IMPORTANT

This feature is available with **Azure CDN from Verizon** products (Standard and Premium). It is not supported on **Azure CDN from Akamai**. For a comparison of CDN features, see [Azure CDN Overview](#).

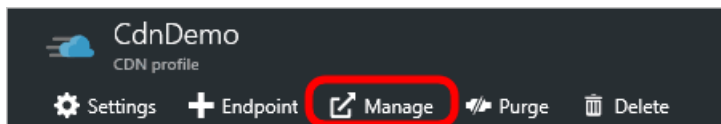
The guide below goes through the steps to view the core reports via the Manage portal for Verizon profiles. You may also export core analytics data to storage, event hub, or log analytics (oms) for both Verizon and Akamai profiles [through the azure portal](#).

You can view usage patterns for your CDN using the following reports:

- Bandwidth
- Data Transferred
- Hits
- Cache Statuses
- Cache Hit Ratio
- IPV4/IPV6 Data Transferred

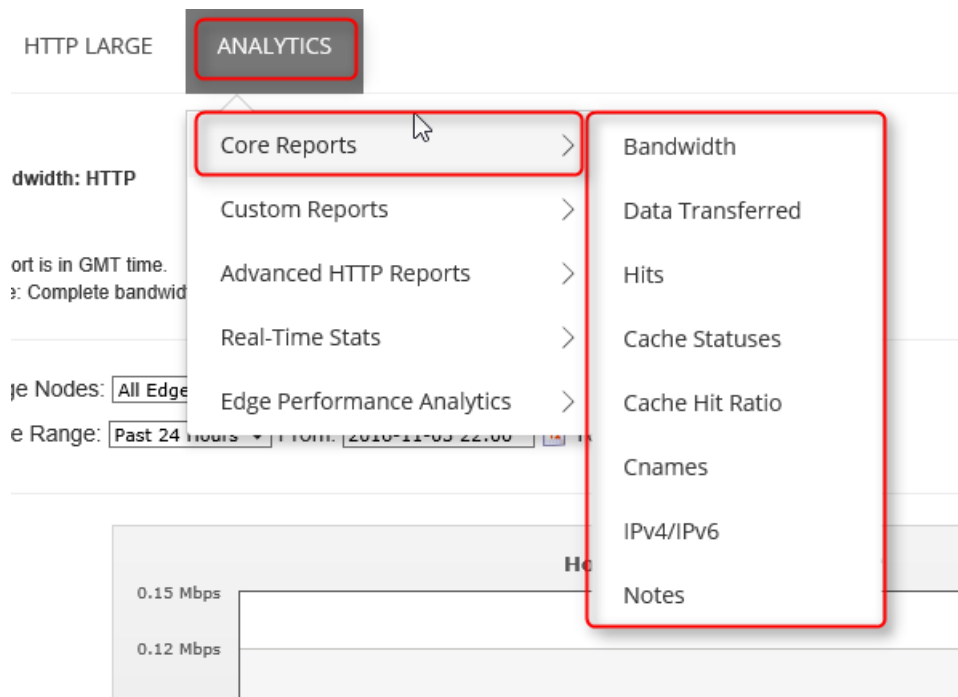
## Accessing Core Reports

1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **Analytics** tab, then hover over the **Core Reports** flyout. Click on the desired report in the menu.

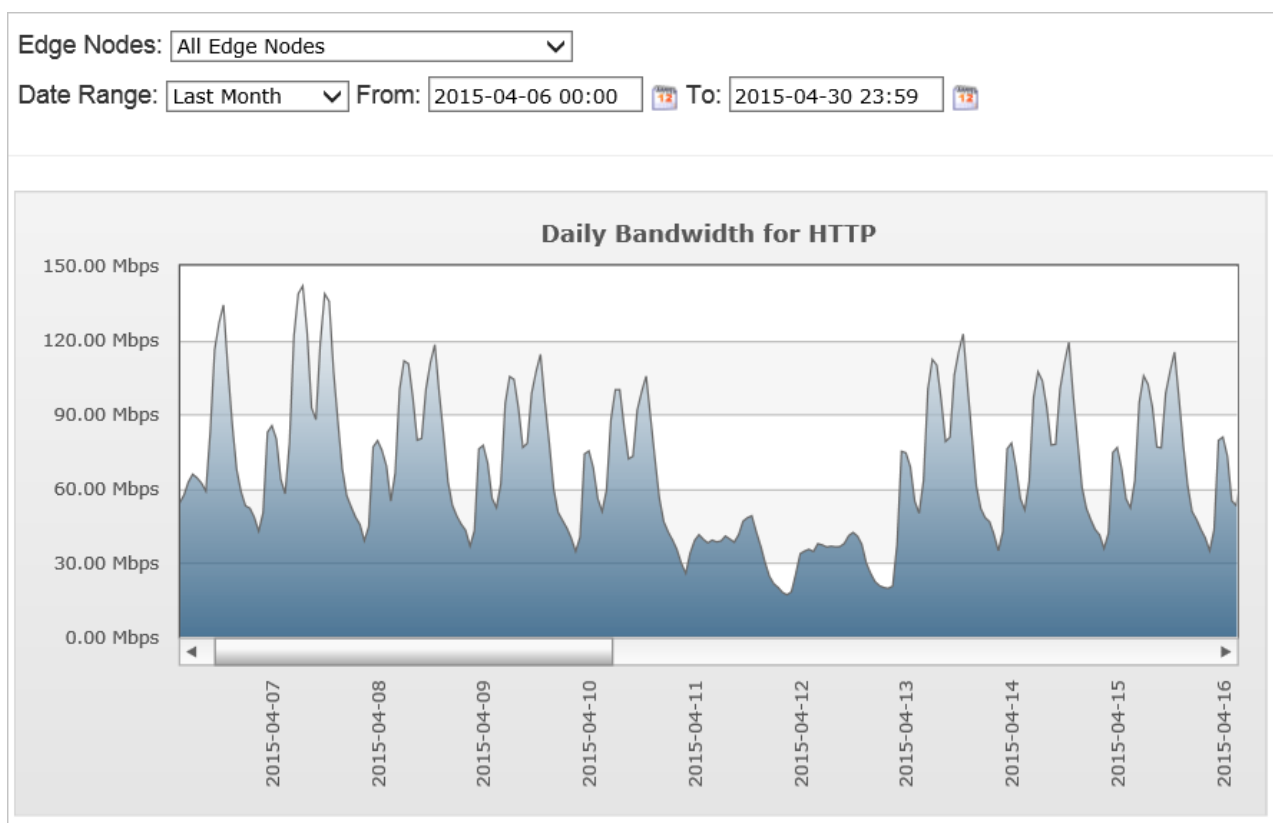


## Bandwidth

The bandwidth report consists of a graph and data table indicating the bandwidth usage for HTTP and HTTPS over a particular time period. You can view the bandwidth usage across all CDN POPs or a particular POP. This allows you to view the traffic spikes and distribution across CDN POPs in Mbps.

- Select All Edge Nodes to see traffic from all nodes or choose a specific region/node from the dropdown list.
- Select Date range to view data for today/this week/this month, etc. or enter custom dates, then click "go" to make sure your selection is updated.
- You can export and download the data by clicking the excel sheet icon located next to "go".

The report is updated every 5 minutes.

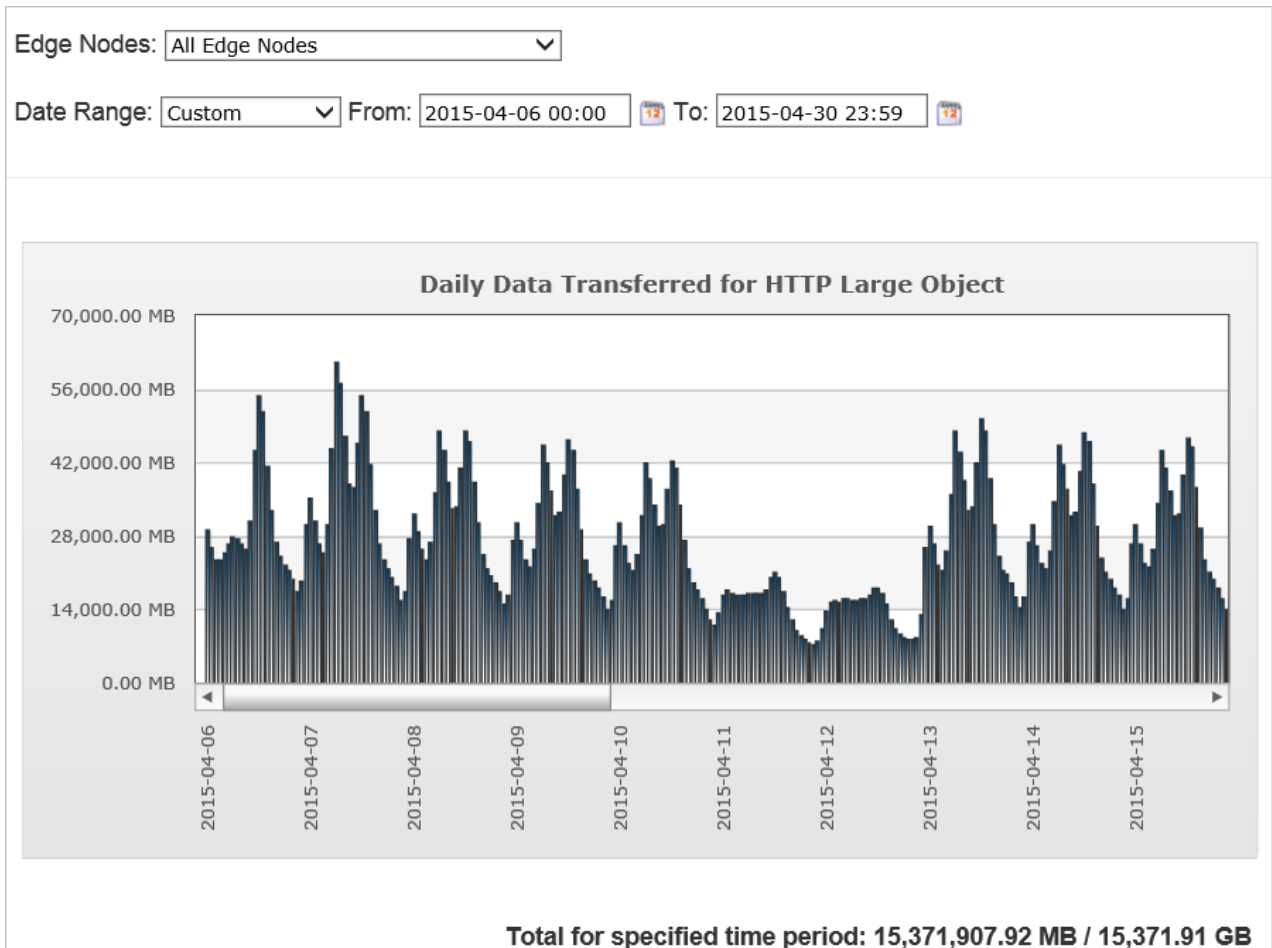


## Data transferred

This report consists of a graph and data table indicating the traffic usage for HTTP and HTTPS over a particular time period. You can view the traffic usage across all CDN POPs or a particular POP. This allows you to view the traffic spikes and distribution across CDN POPs in GB.

- Select All Edge Nodes to see traffic from all notes or choose a specific region/node from the dropdown list.
- Select Date range to view data for today/this week/this month, etc. or enter custom dates, then click "go" to make sure your selection is updated.
- You can export and download the data by clicking the excel sheet icon located next to "go" .

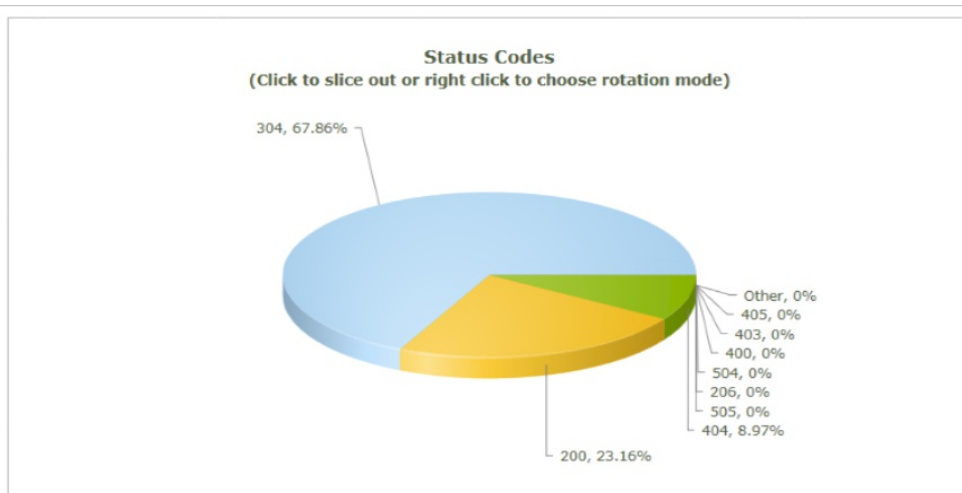
The report is updated every 5 minutes.



## Hits (status codes)

This report describes the distribution of request status codes for your content. Every request for content will generate an HTTP status code. The status code describes how edge POPs handled the request. For example, 2xx status codes indicate that the request was successfully served to a client, while a 4xx status code indicates an error occurred. For more details about HTTP status code, see [status codes](#).

- Select Date range to view data for today/this week/this month, etc. or enter custom dates, then click "go" to make sure your selection is updated.
- You can export and download the data by clicking the excel sheet located next to "go".

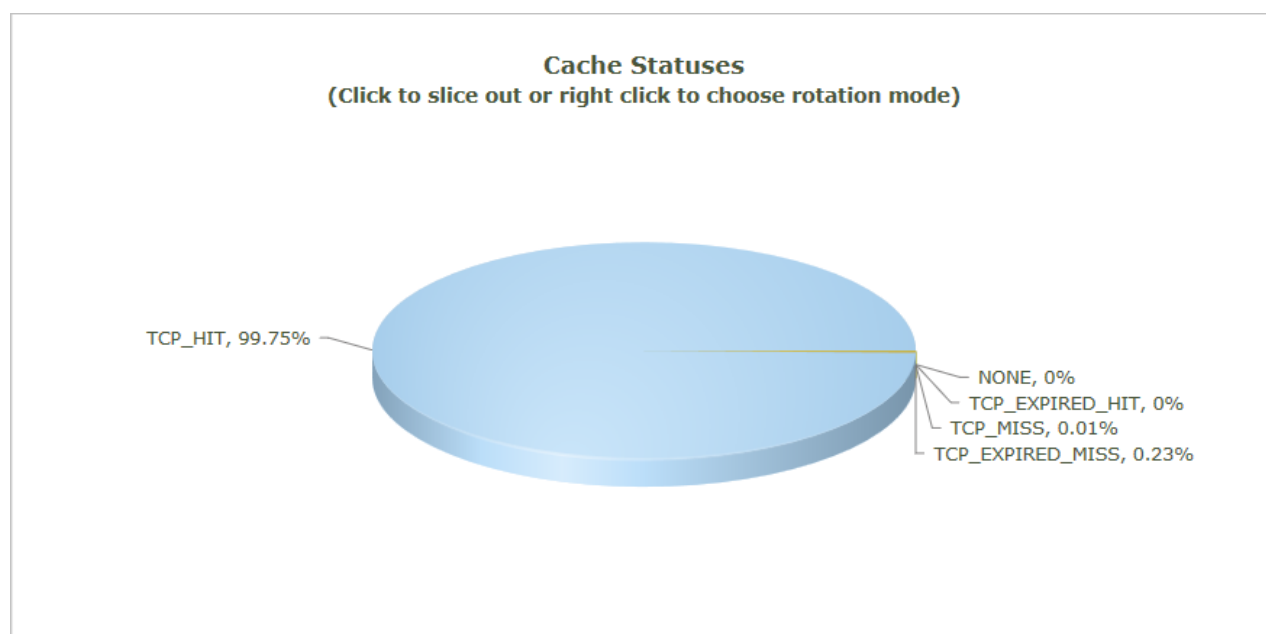


Status Codes: HTTP Large Object

Status Codes	Description	Total Hits	% of Total Hits
304	Not Modified	6553777699	67.86%
200	Ok	2236726432	23.16%

## Cache statuses

This report describes the distribution of cache hits and cache misses for client request. Since the fastest performance comes from cache hits, you can optimize data delivery speeds by minimizing cache misses and expired cache hits. Cache misses can be reduced by configuring your origin server to avoid assigning "no-cache" response headers, by avoiding query-string caching except where strictly needed, and by avoiding non-cacheable response codes. Expired cache hits can be avoided by making an asset's max-age as long as possible to minimize the number of requests to the origin server.



### Main cache statuses include:

- **TCP\_HIT:** Served from Edge. The object was in cache and had not exceeded its max-age.
- **TCP\_MISS:** Served from Origin. The object was not in cache and the response was back to origin.
- **TCP\_EXPIRED\_MISS:** Served from origin after revalidation with origin. The object was in cache but had exceeded its max-age. A revalidation with origin resulted in the cache object being replaced by a new response from origin.
- **TCP\_EXPIRED\_HIT:** Served from Edge after revalidation with origin. The object was in cache but had exceeded

its max-age. A revalidation with the origin server resulted in the cache object being unmodified.

- Select Date range to view data for today/this week/this month, etc. or enter custom dates, then click "go" to make sure your selection is updated.
- You can export and download the data by clicking the excel sheet icon located next to "go".

#### Full list of cache statuses

- **TCP\_HIT** - This status is reported when a request is served directly from the POP to the client. An asset is immediately served from a POP when it is cached on the POP closest to the client and it has a valid time-to-live, or TTL. TTL is determined by the following response headers:
  - Cache-Control: s-maxage
  - Cache-Control: max-age
  - Expires
- **TCP\_MISS** - This status indicates that a cached version of the requested asset was not found on the POP closest to the client. The asset will be requested from either an origin server or an origin shield server. If the origin server or the origin shield server returns an asset, it will be served to the client and cached on both the client and the edge server. Otherwise, a non-200 status code (e.g., 403 Forbidden, 404 Not Found, etc.) will be returned.
- **TCP\_EXPIRED\_HIT** - This status is reported when a request that targeted an asset with an expired TTL, such as when the asset's max-age has expired, was served directly from the POP to the client.

An expired request typically results in a revalidation request to the origin server. In order for a **TCP\_EXPIRED\_HIT** to occur, the origin server must indicate that a newer version of the asset does not exist. This type of situation will typically update that asset's Cache-Control and Expires headers.

- **TCP\_EXPIRED\_MISS** - This status is reported when a newer version of an expired cached asset is served from the POP to the client. This occurs when the TTL for a cached asset has expired (e.g., expired max-age) and the origin server returns a newer version of that asset. This new version of the asset will be served to the client instead of the cached version. Additionally, it will be cached on the edge server and the client.
- **CONFIG\_NOCACHE** - This status indicates that a customer-specific configuration on our edge POP prevented the asset from being cached.
- **NONE** - This status indicates that a cache content freshness check was not performed.
- **TCP\_CLIENT\_REFRESH\_MISS** - This status is reported when an HTTP client (e.g., browser) forces an edge POP to retrieve a new version of a stale asset from the origin server.

By default, our servers prevent an HTTP client from forcing our edge servers to retrieve a new version of the asset from the origin server.

- **TCP\_PARTIAL\_HIT** - This status is reported when a byte range request results in a hit for a partially cached asset. The requested byte range is immediately served from the POP to the client.
- **UNCACHEABLE** - This status is reported when an asset's Cache-Control and Expires headers indicate that it should not be cached on a POP or by the HTTP client. These types of requests are served from the origin server

## Cache Hit Ratio

This report indicates the percentage of cached requests that were served directly from cache.

The report provides the following details:

- The requested content was cached on the POP closest to the requester.
- The request was served directly from the edge of our network.
- The request did not require revalidation with the origin server.

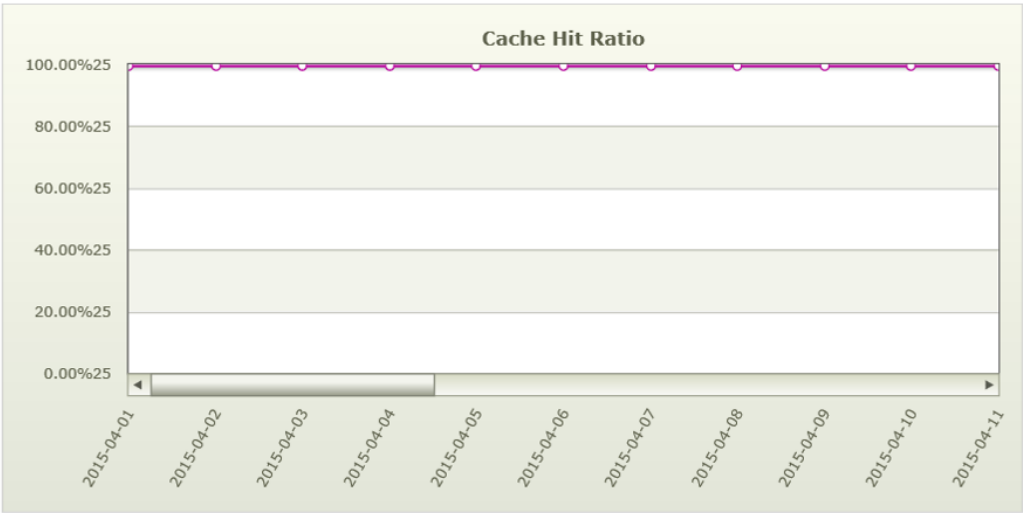
The report doesn't include:



- Requests that are denied due to country filtering options.
- Requests for assets whose headers indicate that they should not be cached. For example, Cache-Control: private, Cache-Control: no-cache, or Pragma: no-cache headers will prevent an asset from being cached.
- Byte range requests for partially cached content.

The formula is:  $(TCP\_HIT / (TCP\_HIT + TCP\_MISS)) * 100$

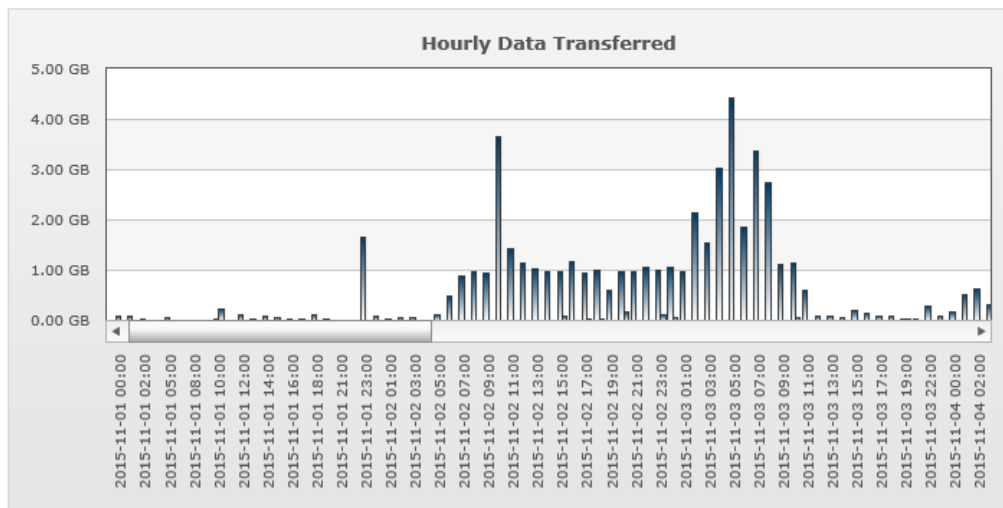
- Select Date range to view data for today/this week/this month, etc. or enter custom dates, then click "go" to make sure your selection is updated.
- You can export and download the data by clicking the excel sheet icon located next to "go" .



Cache Hit Ratio: HTTP Large Object			
Date	Hits	Misses	Cache Hit Percentage
2015-04-30	1935989881	2873417	99.85%
2015-04-29	1950331843	3008493	99.85%
2015-04-28	1991619929	2989130	99.85%

## IPV4/IPV6 Data transferred

This report shows the traffic usage distribution in IPV4 vs IPV6.



Total for specified time period: 88.12 GB

#### IPv4/IPv6 Data Transferred

Date	IPv4 (GB)	IPv6 (GB)	Total (GB)
2015-11-01 00:00	0.07 GB	0.00 GB	0.07 GB
2015-11-01 01:00	0.06 GB	0.00 GB	0.06 GB
2015-11-01 02:00	0.02 GB	0.00 GB	0.02 GB

- Select Date range to view data for today/this week/this month, etc. or enter custom dates.
- Then, click "go" to make sure your selection is updated.

## Considerations

Reports can only be generated within the last 18 months.

# Analyze usage statistics with Azure CDN advanced HTTP reports

1/24/2017 • 15 min to read • [Edit Online](#)

## Overview

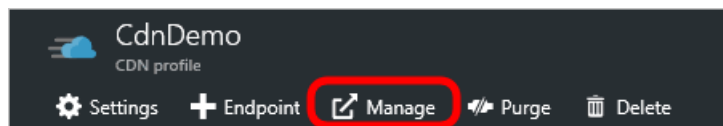
This document explains advanced HTTP reporting in Microsoft Azure CDN. These reports provide detailed information on CDN activity.

### IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

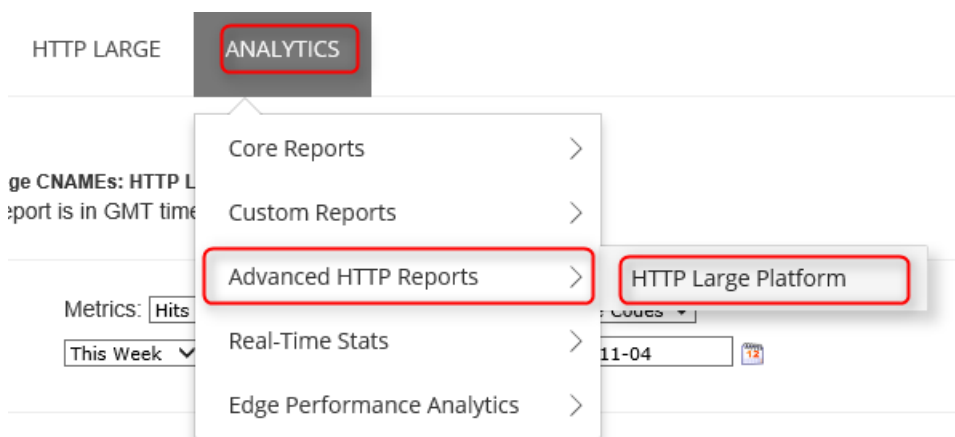
## Accessing advanced HTTP reports

1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **Analytics** tab, then hover over the **Advanced HTTP Reports** flyout. Click on **HTTP Large Platform**.



Report options are displayed.

## Geography Reports (Map-Based)

There are five reports that take advantage of a map to indicate the regions from which your content is being requested. These reports are World Map, United States Map, Canada Map, Europe Map, and Asia Pacific Map.

Each map-based report ranks geographic entities (i.e., countries, states, and provinces) according to the percentage of hits that originated from that region. Additionally, a map is provided to help you visualize the locations from which your content is being requested. It is able to do so by color-coding each region according to the amount of

demand experienced in that region. Lighter shaded regions indicate lower demand for your content, while darker regions indicate higher levels of demand for your content.

Detailed traffic and bandwidth information for each region is provided directly below the map. This allows you to view the total number of hits, the percentage of hits, the total amount of data transferred (in gigabytes), and the percentage of data transferred for each region. View a description for each of these metrics. Finally, when you hover over a region (i.e., country, state, or province), the name and the percentage of hits that occurred in the region will be displayed as a tooltip.

A brief description is provided below for each type of map-based geography report.

REPORT NAME	DESCRIPTION
World Map	This report allows you to view the worldwide demand for your CDN content. Each country is color-coded on the world map to indicate the percentage of hits that originated from that region.
United States Map	This report allows you to view the demand for your CDN content in the United States. Each state is color-coded on this map to indicate the percentage of hits that originated from that region.
Canada Map	This report allows you to view the demand for your CDN content in Canada. Each province is color-coded on this map to indicate the percentage of hits that originated from that region.
Europe Map	This report allows you to view the demand for your CDN content in Europe. Each country is color-coded on this map to indicate the percentage of hits that originated from that region.
Asia Pacific Map	This report allows you to view the demand for your CDN content in Asia. Each country is color-coded on this map to indicate the percentage of hits that originated from that region.

## Geography Reports (Bar Charts)

There are two additional reports that provide statistical information according to geography, which are Top Cities and Top Countries. These reports rank cities and countries, respectively, according to the number of hits that originated from those regions. Upon generating this type of report, a bar chart will indicate the top 10 cities or countries that requested content over a specific platform. This bar chart allows you to quickly assess the regions that generate the highest number of requests for your content.

The left-hand side of the graph (y-axis) indicates how many hits occurred in the specified region. Directly below the graph (x-axis), you will find a label for each of the top 10 regions.

### Using the bar charts

- If you hover over a bar, the name and the total number of hits that occurred in the region will be displayed as a tooltip.
- The tooltip for the Top Cities report identifies a city by its name, state/province, and country abbreviation.
- If the city or region (i.e., state/province) from which a request originated could not be determined, then it will indicate that they are unknown. If the country is unknown, then two question marks (i.e., ??) will be displayed.
- A report may include metrics for "Europe" or the "Asia/Pacific Region." Those items are not meant to provide

statistical information on all IP addresses in those regions. Rather, they only apply to requests that originate from IP addresses that are spread out over Europe or Asia/Pacific instead of to a specific city or country.

The data that was used to generate the bar chart can be viewed below it. There you will find the total number of hits, the percentage of hits, the amount of data transferred (in gigabytes), and the percentage of data transferred for the top 250 regions. View a description for each of these metrics.

A brief description is provided for both types of reports below.

REPORT NAME	DESCRIPTION
Top Cities	This report ranks cities according to the number of hits that originated from that region.
Top Countries	This report ranks countries according to the number of hits that originated from that region.

## Daily Summary

The Daily Summary report allows you to view the total number of hits and data transferred over a particular platform on a daily basis. This information can be used to quickly discern CDN activity patterns. For example, this report can help you detect which days experienced higher or lower than expected traffic.

Upon generating this type of report, a bar chart will provide a visual indication as to the amount of platform-specific demand experienced on a daily basis over the time period covered by the report. It will do so by displaying a bar for each day in the report. For example, selecting the time period called "Last Week" will generate a bar chart with seven bars. Each bar will indicate the total number of hits experienced on that day.

The left-hand side of the graph (y-axis) indicates how many hits occurred on the specified date. Directly below the graph (x-axis), you will find a label that indicates the date (Format: YYYY-MM-DD) for each day included in the report.

### TIP

If you hover over a bar, the total number of hits that occurred on that date will be displayed as a tooltip.

The data that was used to generate the bar chart can be viewed below it. There you will find the total number of hits and the amount of data transferred (in gigabytes) for each day covered by the report.

## By Hour

The By Hour report allows you to view the total number of hits and data transferred over a particular platform on an hourly basis. This information can be used to quickly discern CDN activity patterns. For example, this report can help you detect the time periods during the day that experience higher or lower than expected traffic.

Upon generating this type of report, a bar chart will provide a visual indication as to the amount of platform-specific demand experienced on an hourly basis over the time period covered by the report. It will do so by displaying a bar for each hour covered by the report. For example, selecting a 24 hour time period will generate a bar chart with twenty four bars. Each bar will indicate the total number of hits experienced during that hour.

The left-hand side of the graph (y-axis) indicates how many hits occurred on the specified hour. Directly below the graph (x-axis), you will find a label that indicates the date/time (Format: YYYY-MM-DD hh:mm) for each hour included in the report. Time is reported using 24 hour format and it is specified using the UTC/GMT time zone.

**TIP**

If you hover over a bar, the total number of hits that occurred during that hour will be displayed as a tooltip.

The data that was used to generate the bar chart can be viewed below it. There you will find the total number of hits and the amount of data transferred (in gigabytes) for each hour covered by the report.

## By File

The By File report allows you to view the amount of demand and the traffic incurred over a particular platform for the most requested assets. Upon generating this type of report, a bar chart will be generated on the top 10 most requested assets over the specified time period.

**NOTE**

For the purposes of this report, edge CNAME URLs are converted to their equivalent CDN URLs. This allows an accurate tally for the total number of hits associated with an asset regardless of the CDN or edge CNAME URL used to request it.

The left-hand side of the graph (y-axis) indicates the number of requests for each asset over the specified time period. Directly below the graph (x-axis), you will find a label that indicates the file name for each of the top 10 requested assets.

The data that was used to generate the bar chart can be viewed below it. There you will find the following information for each of the top 250 requested assets: relative path, the total number of hits, the percentage of hits, the amount of data transferred (in gigabytes), and the percentage of data transferred.

## By File Detail

The By File Detail report allows you to view the amount of demand and the traffic incurred over a particular platform for a specific asset. At the very top of this report is the File Details For option. This option provides a list of your most requested assets on the selected platform. In order to generate a By File Detail report, you will need to select the desired asset from the File Details For option. After which, a bar chart will indicate the amount of daily demand that it generated over the specified time period.

The left-hand side of the graph (y-axis) indicates the total number of requests that an asset experienced on a particular day. Directly below the graph (x-axis), you will find a label that indicates the date (Format: YYYY-MM-DD) for which CDN demand for the asset was reported.

The data that was used to generate the bar chart can be viewed below it. There you will find the total number of hits and the amount of data transferred (in gigabytes) for each day covered by the report.

## By File Type

The By File Type report allows you to view the amount of demand and the traffic incurred by file type. Upon generating this type of report, a donut chart will indicate the percentage of hits generated by the top 10 file types.

**TIP**

If you hover over a slice in the donut chart, the Internet media type of that file type will be displayed as a tooltip.

The data that was used to generate the donut chart can be viewed below it. There you will find the file name extension/Internet media type, the total number of hits, the percentage of hits, the amount of data transferred (in gigabytes), and the percentage of data transferred for each of the top 250 file types.

## By Directory

The By Directory report allows you to view the amount of demand and the traffic incurred over a particular platform for content from a specific directory. Upon generating this type of report, a bar chart will indicate the total number of hits generated by content in the top 10 directories.

### Using the bar chart

- Hover over a bar to view the relative path to the corresponding directory.
- Content stored in a subfolder of a directory does not count when calculating demand by directory. This calculation relies solely on the number of requests generated for content stored in the actual directory.
- For the purposes of this report, edge CNAME URLs are converted to their equivalent CDN URLs. This allows an accurate tally for all statistics associated with an asset regardless of the CDN or edge CNAME URL used to request it.

The left-hand side of the graph (y-axis) indicates the total number of requests for the content stored in your top 10 directories. Each bar on the chart represents a directory. Use the color-coding scheme to match up a bar to a directory listed in the Top 250 Full Directories section.

The data that was used to generate the bar chart can be viewed below it. There you will find the following information for each of the top 250 directories: relative path, the total number of hits, the percentage of hits, the amount of data transferred (in gigabytes), and the percentage of data transferred.

## By Browser

The By Browser report allows you to view which browsers were used to request content. Upon generating this type of report, a pie chart will indicate the percentage of requests handled by the top 10 browsers.

### Using the pie chart

- Hover over a slice in the pie chart to view a browser's name and version.
- For the purposes of this report, each unique browser/version combination is considered a different browser.
- The slice called "Other" indicates the percentage of requests handled by all other browsers and versions.

The data that was used to generate the pie chart can be viewed below it. There you will find the browser type/version number, the total number of hits and the percentage of hits for each of the top 250 browsers.

## By Referrer

The By Referrer report allows you to view the top referrers to content on the selected platform. A referrer indicates the hostname from which a request was generated. Upon generating this type of report, a bar chart will indicate the amount of demand (i.e., hits) generated by the top 10 referrers.

The left-hand side of the graph (y-axis) indicates the total number of requests that an asset experienced for each referrer. Each bar on the chart represents a referrer. Use the color-coding scheme to match up a bar to a referrer listed in the Top 250 Referrer section.

The data that was used to generate the bar chart can be viewed below it. There you will find the URL, the total number of hits, and the percentage of hits generated from each of the top 250 referrers.

## By Download

The By Download report allows you to analyze download patterns for your most requested content. The top of the report contains a bar chart that compares attempted downloads with completed downloads for the top 10 requested assets. Each bar is color-coded according to whether it is an attempted download (blue) or a completed download (green).

#### NOTE

For the purposes of this report, edge CNAME URLs are converted to their equivalent CDN URLs. This allows an accurate tally for all statistics associated with an asset regardless of the CDN or edge CNAME URL used to request it.

The left-hand side of the graph (y-axis) indicates the file name for each of the top 10 requested assets. Directly below the graph (x-axis), you will find labels that indicate the total number of attempted/completed downloads.

Directly below the bar chart, the following information will be listed for the top 250 requested assets: relative path (including file name), the number of times that it was downloaded to completion, the number of times that it was requested, and the percentage of requests that resulted in a complete download.

#### TIP

Our CDN is not informed by an HTTP client (i.e. browser) when an asset has been completely downloaded. As a result, we have to calculate whether an asset has been completely downloaded according to status codes and byte-range requests. The first thing we look for when making this calculation is whether the request results in a 200 OK status code. If so, then we look at byte-range requests to ensure that they cover the entire asset. Finally, we compare the amount of data transferred to the size of the requested asset. If the data transferred is equal to or greater than the file size and the byte-range requests are appropriate for that asset, then the hit will be counted as a complete download.

Due to the interpretive nature of this report, you should keep in mind the following points that may alter the consistency and accuracy of this report.

- Traffic patterns cannot be accurately predicted when user-agents behave differently. This may produce completed download results that are greater than 100%.
- Assets that take advantage of HTTP Progressive Download may not be accurately represented by this report. This is due to users seeking to different positions in a video.

## By 404 Errors

The By 404 Errors report allows you to identify the type of content that generates the most number of 404 Not Found status codes. The top of the report contains a bar chart for the top 10 assets for which a 404 Not Found status code was returned. This bar chart compares the total number of requests with requests that resulted in a 404 Not Found status code for those assets. Each bar is color-coded. A yellow bar is used to indicate that the request resulted in a 404 Not Found status code. A red bar is used to indicate the total number of requests for the asset.

#### NOTE

For the purposes of this report, note the following:

- A hit represents any request for an asset regardless of status code.
- Edge CNAME URLs are converted to their equivalent CDN URLs. This allows an accurate tally for all statistics associated with an asset regardless of the CDN or edge CNAME URL used to request it.

The left-hand side of the graph (y-axis) indicates the file name for each of the top 10 requested assets that resulted in a 404 Not Found status code. Directly below the graph (x-axis), you will find labels that indicate the total number of requests and the number of requests that resulted in a 404 Not Found status code.

Directly below the bar chart, the following information will be listed for the top 250 requested assets: relative path (including file name), the number of requests that resulted in a 404 Not Found status code, the total number of times that the asset was requested, and the percentage of requests that resulted in a 404 Not Found status code.



## See also

- [Azure CDN Overview](#)
- [Real-time stats in Microsoft Azure CDN](#)
- [Overriding default HTTP behavior using the rules engine](#)
- [Analyze Edge Performance](#)

# Real-time stats in Microsoft Azure CDN

1/25/2017 • 3 min to read • [Edit Online](#)

## IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

## Overview

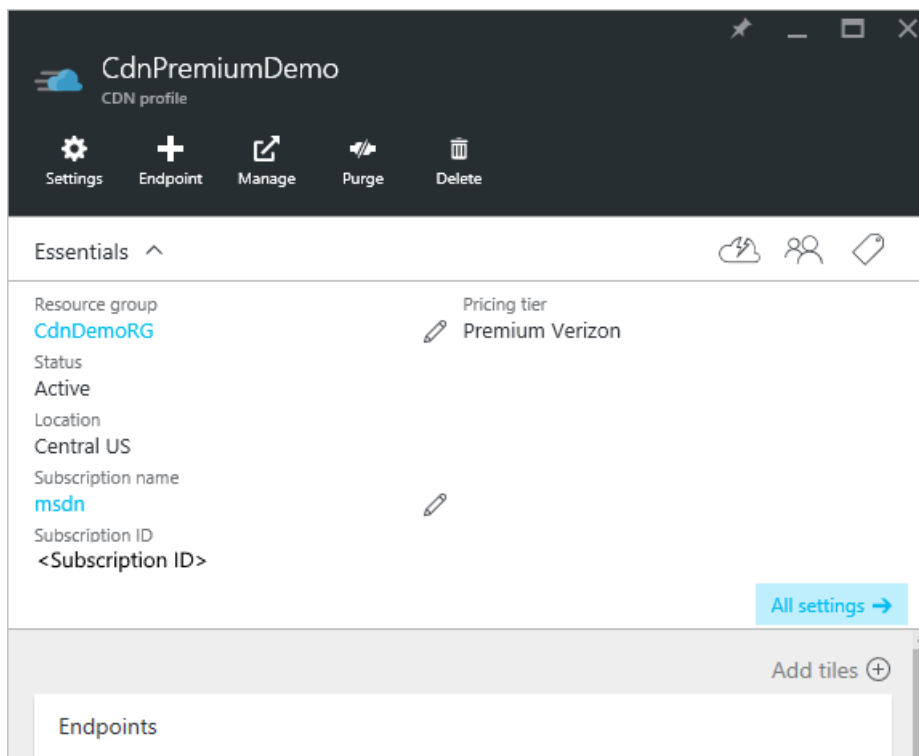
This document explains real-time stats in Microsoft Azure CDN. This functionality provides real-time data, such as bandwidth, cache statuses, and concurrent connections to your CDN profile when delivering content to your clients. This enables continuous monitoring of the health of your service at any time, including go-live events.

The following graphs are available:

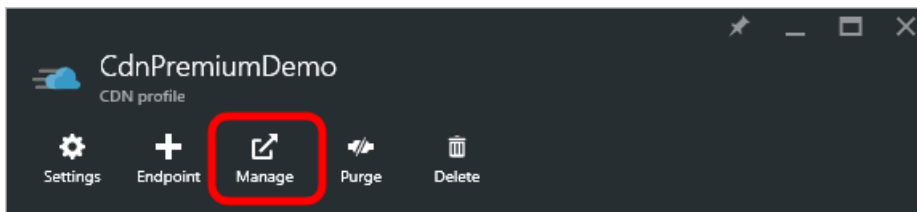
- [Bandwidth](#)
- [Status Codes](#)
- [Cache Statuses](#)
- [Connections](#)

## Accessing real-time stats

1. In the [Azure Portal](#), browse to your CDN profile.

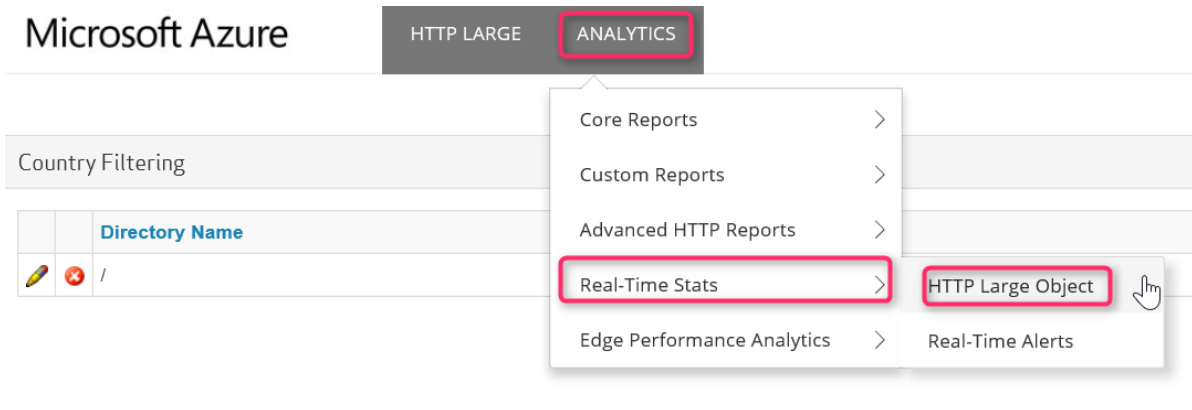


2. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

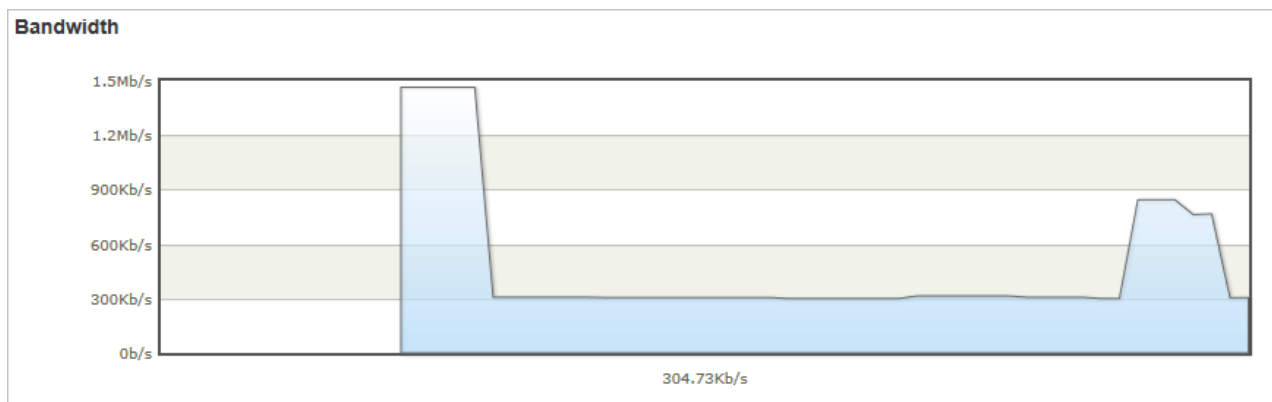
3. Hover over the **Analytics** tab, then hover over the **Real-Time Stats** flyout. Click on **HTTP Large Object**.



The real-time stats graphs are displayed.

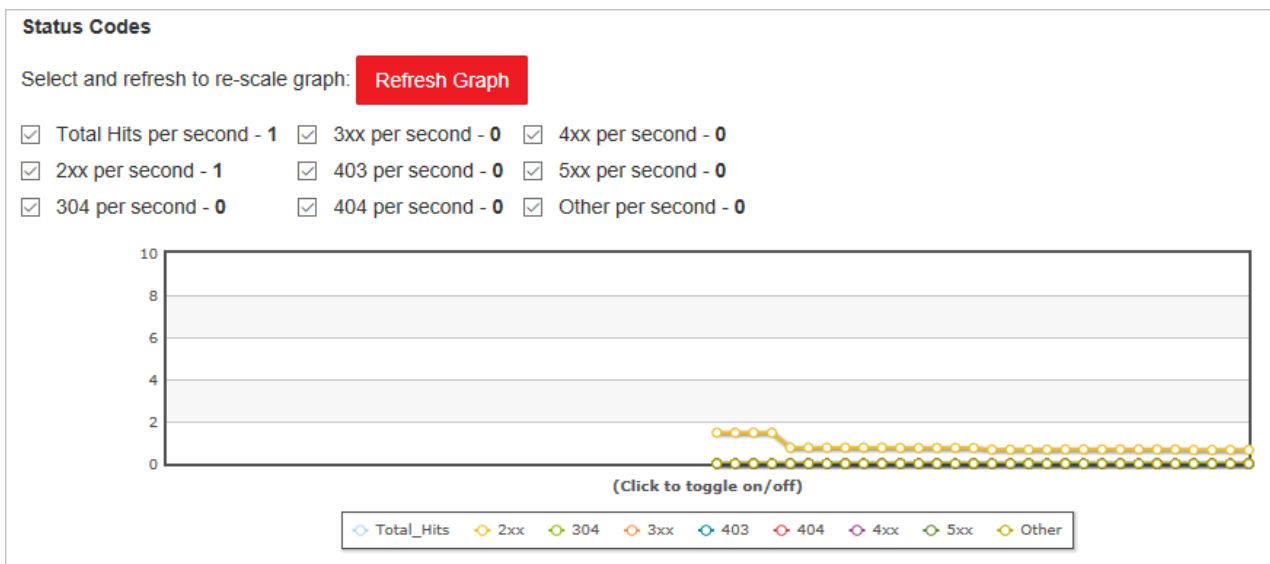
Each of the graphs displays real-time statistics for the selected time span, starting when the page loads. The graphs update automatically every few seconds. The **Refresh Graph** button, if present, will clear the graph, after which it will only display the selected data.

## Bandwidth



The **Bandwidth** graph displays the amount of bandwidth used for the current platform over the selected time span. The shaded portion of the graph indicates bandwidth usage. The exact amount of bandwidth currently being used is displayed directly below the line graph.

## Status Codes



The **Status Codes** graph indicates how often certain HTTP response codes are occurring over the selected time span.

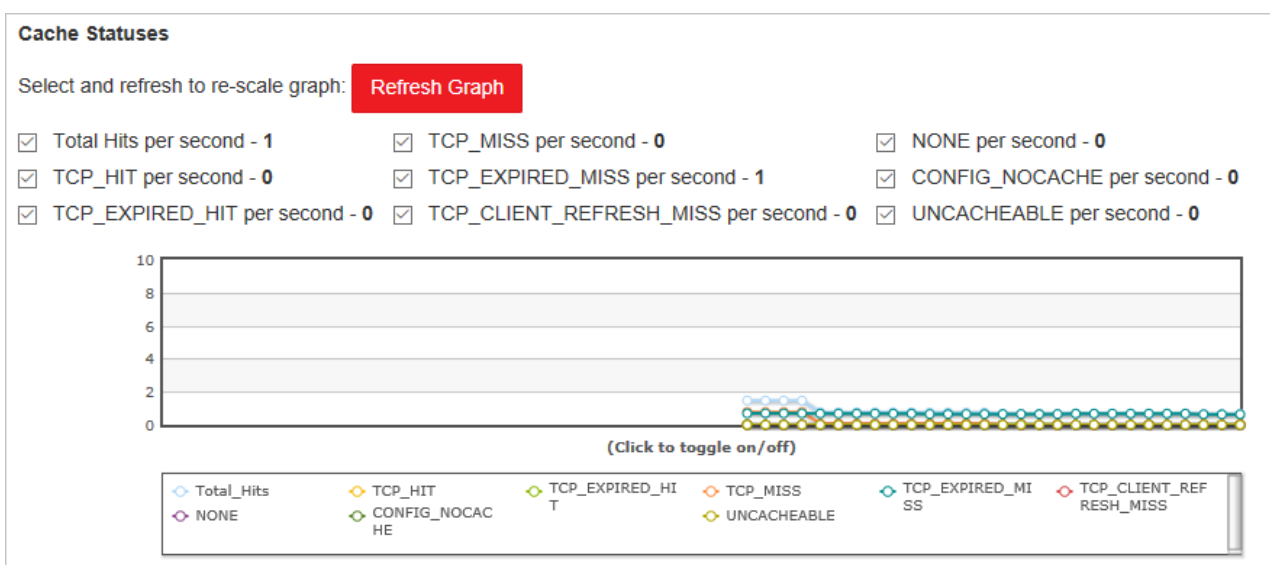
#### TIP

For a description of each HTTP status code option, see [Azure CDN HTTP Status Codes](#).

A list of HTTP status codes is displayed directly above the graph. This list indicates each status code that can be included in the line graph and the current number of occurrences per second for that status code. By default, a line is displayed for each of these status codes in the graph. However, you can choose to only monitor the status codes that have special significance for your CDN configuration. To do this, check the desired status codes and clear all other options, then click **Refresh Graph**.

You can temporarily hide logged data for a particular status code. From the legend directly below the graph, click the status code you want to hide. The status code will be immediately hidden from the graph. Clicking that status code again will cause that option to be displayed again.

## Cache Statuses



The **Cache Statuses** graph indicates how often certain types of cache statuses are occurring over the selected time span.

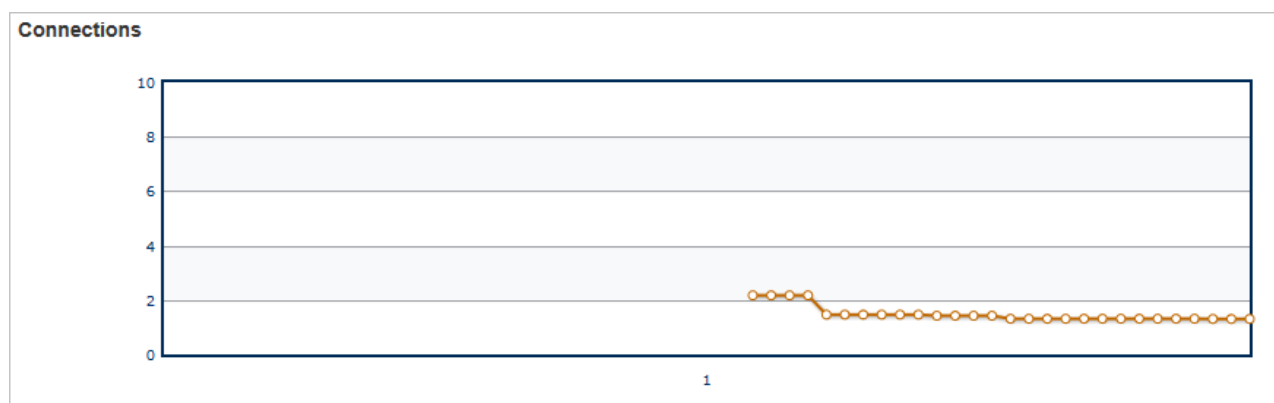
### TIP

For a description of each cache status code option, see [Azure CDN Cache Status Codes](#).

A list of cache status codes is displayed directly above the graph. This list indicates each status code that can be included in the line graph and the current number of occurrences per second for that status code. By default, a line is displayed for each of these status codes in the graph. However, you can choose to only monitor the status codes that have special significance for your CDN configuration. To do this, check the desired status codes and clear all other options, then click **Refresh Graph**.

You can temporarily hide logged data for a particular status code. From the legend directly below the graph, click the status code you want to hide. The status code will be immediately hidden from the graph. Clicking that status code again will cause that option to be displayed again.

## Connections



This graph indicates how many connections have been established to your edge servers. Each request for an asset that passes through our CDN results in a connection.

## Next Steps

- Get notified with [Real-time alerts in Azure CDN](#)
- Dig deeper with [advanced HTTP reports](#)
- Analyze [usage patterns](#)

# Analyze edge node performance in Microsoft Azure CDN

1/24/2017 • 14 min to read • [Edit Online](#)

## IMPORTANT

This is a feature of **Azure CDN Premium from Verizon**, and is not available with **Azure CDN Standard** products. For a comparison of CDN features, see [Azure CDN Overview](#).

## Overview

Edge performance analytics provides granular information traffic and bandwidth usage for the CDN. This information can then be used to generate trending statistics, which allow you to gain insight on how your assets are being cached and delivered to your clients. In turn, this allows you to form a strategy on how to optimize the delivery of your content and to determine what issues should be tackled to better leverage the CDN. As a result, not only will you be able to improve data delivery performance, but you will also be able to reduce your CDN costs.

## NOTE

All reports use UTC/GMT notation when specifying a date/time.

## Reports and log collection

CDN activity data must be collected by the Edge Performance Analytics module before it can generate reports on it. This collection process occurs once a day and it covers the activity that took place during the previous day. This means that a report's statistics represent a sample of the day's statistics at the time it was processed, and do not necessarily contain the complete set of data for the current day. The primary function of these reports is to assess performance. They should not be used for billing purposes or exact numeric statistics.

## NOTE

The raw data from which Edge Performance Analytic reports are generated is available for at least 90 days.

## Dashboard

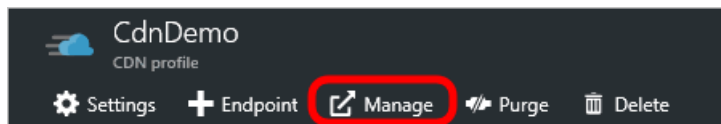
The Edge Performance Analytics dashboard tracks current and historical CDN traffic through a chart and statistics. Use this dashboard to detect recent and long-term trends on the performance of CDN traffic for your account.

This dashboard consists of:

- An interactive chart that allows the visualization of key metrics and trends.
- A timeline that provides a sense of long term patterns for key metrics and trends.
- Key metrics and statistical information on how our CDN network improves site traffic as measured by overall performance, usage, and efficiency.

### Accessing the edge performance dashboard

1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **Analytics** tab, then hover over the **Edge Performance Analytics** flyout. Click on **Dashboard**.

The edge node analytics dashboard is displayed.

## Chart

The dashboard contains a chart that tracks a metric over the time period selected in the timeline that appears directly below it. A timeline that graphs up to the last two years of CDN activity is displayed directly below the chart.

### Using the chart

- By default, the cache efficiency rate for the last 30 days will be charted.
- This chart is generated from data collated on a daily basis.
- Hovering over a day on the line graph will indicate a date and the value of the metric on that date.
- Click Highlight Weekends to toggle an overlay of light gray vertical bars that represent weekends onto the chart. This type of overlay is useful for identifying traffic patterns over weekends.
- Click View One Year Ago to toggle an overlay of the previous year's activity over the same time period onto the chart. This type of comparison provides insight into long-term CDN usage patterns. The upper-right hand corner of the chart contains a legend that indicates the color code for each line graph.

### Updating the chart

- Time Range: Perform one of the following:
  - Select the desired region in the timeline. The chart will be updated with data that corresponds to the selected time period.
  - Double-click the chart to display all available historical data up to a maximum of two years.
- Metric: Click the chart icon that appears next to the desired metric. The chart and the timeline will be refreshed with data for the corresponding metric.

## Key metrics and statistics

### Efficiency metrics

The purpose of these metrics is to see whether cache efficiency can be improved. The main benefits derived from cache efficiency are:

- Reduced load on the origin server which may lead to:
  - Better web server performance.
  - Reduced operational costs.
- Improved data delivery acceleration since more requests will be served directly from the CDN.

FIELD	DESCRIPTION
Cache Efficiency	Indicates the percentage of data transferred that was served from cache. This metric measures when a cached version of the requested content was served directly from the CDN (edge servers) to requesters (e.g., web browser)
Hit Rate	Indicates the percentage of requests that were served from cache. This metric measures when a cached version of the requested content was served directly from the CDN (edge servers) to requesters (e.g., web browser).

FIELD	DESCRIPTION
% of Remote Bytes - No Cache Config	Indicates the percentage of traffic that was served from origin servers to the CDN (edge servers) that will not be cached as a result of the Bypass Cache feature (HTTP Rules Engine).
% of Remote Bytes - Expired Cache	Indicates the percentage of traffic that was served from origin servers to the CDN (edge servers) as a result of stale content revalidation.

#### Usage metrics

The purpose of these metrics is to provide insight into the following cost-cutting measures:

- Minimizing operational costs through the CDN.
- Reducing CDN expenditures through cache efficiency and compression.

#### NOTE

Traffic volume numbers represent traffic that was used in calculations of ratios and percentages, and may only show a portion of the total traffic for high-volume customers.

FIELD	DESCRIPTION
Ave Bytes Out	Indicates the average number of bytes transferred for each request served from the CDN (edge servers) to the requester (e.g., web browser).
No Cache Config Byte Rate	Indicates the percentage of traffic served from the CDN (edge servers) to the requester (e.g., web browser) that will not be cached due to the Bypass Cache feature.
Compressed Byte Rate	Indicates the percentage of traffic sent from the CDN (edge servers) to requesters (e.g., web browser) in a compressed format.
Bytes Out	Indicates the amount of data, in bytes, that were delivered from the CDN (edge servers) to the requester (e.g., web browser).
Bytes In	Indicates the amount of data, in bytes, sent from requesters (e.g., web browser) to the CDN (edge servers).
Bytes Remote	Indicates the amount of data, in bytes, sent from CDN and customer origin servers to the CDN (edge servers).

#### Performance Metrics

The purpose of these metrics is to track overall CDN performance for your traffic.

FIELD	DESCRIPTION
Transfer Rate	Indicates the average rate at which content was transferred from the CDN to a requester.
Duration	Indicates the average time, in milliseconds, it took to deliver an asset to a requester (e.g., web browser).



FIELD	DESCRIPTION
Compressed Request Rate	Indicates the percentage of hits that were delivered from the CDN (edge servers) to the requester (e.g., web browser) in a compressed format.
4xx Error Rate	Indicates the percentage of hits that generated a 4xx status code.
5xx Error Rate	Indicates the percentage of hits that generated a 5xx status code.
Hits	Indicates the number of requests for CDN content.

### Secure Traffic Metrics

The purpose of these metrics is to track CDN performance for HTTPS traffic.

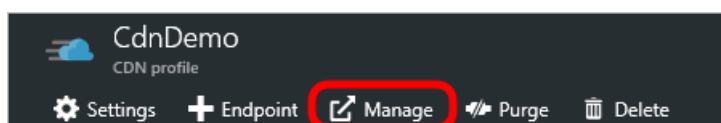
FIELD	DESCRIPTION
Secure Cache Efficiency	Indicates the percentage of data transferred for HTTPS requests that were served from cache. This metric measures when a cached version of the requested content was served directly from the CDN (edge servers) to requesters (e.g., web browser) over HTTPS.
Secure Transfer Rate	Indicates the average rate at which content was transferred from the CDN (edge servers) to requesters (e.g., web servers) over HTTPS.
Average Secure Duration	Indicates the average time, in milliseconds, it took to deliver an asset to a requester (e.g., web browser) over HTTPS.
Secure Hits	Indicates the number of HTTPS requests for CDN content.
Secure Bytes Out	Indicates the amount of HTTPS traffic, in bytes, that were delivered from the CDN (edge servers) to the requester (e.g., web browser).

## Reports

Each report in this module contains a chart and statistics on bandwidth and traffic usage for different types of metrics (e.g., HTTP status codes, cache status codes, request URL, etc.). This information may be used to delve deeper into how content is being served to your clients and to fine-tune CDN behavior to improve data delivery performance.

### Accessing the edge performance reports

1. From the CDN profile blade, click the **Manage** button.



The CDN management portal opens.

2. Hover over the **Analytics** tab, then hover over the **Edge Performance Analytics** flyout. Click on **HTTP Large Object**.

The edge node analytics reports screen is displayed.

REPORT	DESCRIPTION
Daily Summary	Allows you to view daily traffic trends over a specified time period. Each bar on this graph represents a particular date. The size of the bar indicates the total number of hits that occurred on that date.
Hourly Summary	Allows you to view hourly traffic trends over a specified time period. Each bar on this graph represents a single hour on a particular date. The size of the bar indicates the total number of hits that occurred during that hour.
Protocols	Displays the breakdown of traffic between the HTTP and HTTPS protocols. A donut chart indicates the percentage of hits that occurred for each type of protocol.
HTTP Methods	Allows you to get a quick sense of which HTTP methods are being used to request your data. Typically, the most common HTTP request methods are GET, HEAD, and POST. A donut chart indicates the percentage of hits that occurred for each type of HTTP request method.
URLs	Contains a graph that displays the top 10 requested URLs. A bar is displayed for each URL. The height of the bar indicates how many hits that particular URL has generated over the time span covered by the report. Statistics for the top 100 requested URLs are displayed directly below this graph.
CNAMEs	Contains a graph that displays the top 10 CNAMEs used to request assets over the time span of a report. Statistics for the top 100 requested CNAMEs are displayed directly below this graph.
Origins	Contains a graph that displays the top 10 CDN or customer origin servers from which assets were requested over a specified period of time. Statistics for the top 100 requested CDN or customer origin servers are displayed directly below this graph. Customer origin servers are identified by the name defined in the Directory Name option.
Geo POPs	Shows how much of your traffic is being routed through a particular point-of-presence (POP). The three-letter abbreviation represents a POP in our CDN network.
Clients	Contains a graph that displays the top 10 clients that requested assets over a specified period of time. For the purposes of this report, all requests that originate from the same IP address are considered to be from the same client. Statistics for the top 100 clients are displayed directly below this graph. This report is useful for determining download activity patterns for your top clients.
Cache Statuses	Gives a detailed breakdown of cache behavior, which may reveal approaches for improving the overall end-user experience. Since the fastest performance comes from cache hits, you can optimize data delivery speeds by minimizing cache misses and expired cache hits.

REPORT	DESCRIPTION
NONE Details	Contains a graph that displays the top 10 URLs for assets for which cache content freshness was not checked over a specified period of time. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
CONFIG_NOCACHE Details	Contains a graph that displays the top 10 URLs for assets that were not cached due to the customer's CDN configuration. These types of assets were served directly from the origin server. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
UNCACHEABLE Details	Contains a graph that displays the top 10 URLs for assets that could not be cached due to request header data. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
TCP_HIT Details	Contains a graph that displays the top 10 URLs for assets that are served immediately from cache. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
TCP_MISS Details	Contains a graph that displays the top 10 URLs for assets that have a cache status of TCP_MISS. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
TCP_EXPIRED_HIT Details	Contains a graph that displays the top 10 URLs for stale assets that were served directly from the POP. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
TCP_EXPIRED_MISS Details	Contains a graph that displays the top 10 URLs for stale assets for which a new version had to be retrieved from the origin server. Statistics for the top 100 URLs for these types of assets are displayed directly below this graph.
TCP_CLIENT_REFRESH_MISS Details	Contains a bar chart that displays the top 10 URLs for assets were retrieved from an origin server due to a no-cache request from the client. Statistics for the top 100 URLs for these types of requests are displayed directly below this chart.
Client Request Types	Indicates the type of requests that were made by HTTP clients (e.g., browsers). This report includes a donut chart that provides a sense as to how requests are being handled. Bandwidth and traffic information for each request type is displayed below the chart.
User Agent	Contains a bar graph displaying the top 10 user agents to request your content through our CDN. Typically, a user agent is a web browser, media player, or a mobile phone browser. Statistics for the top 100 user agents are displayed directly below this chart.

REPORT	DESCRIPTION
Referrers	Contains a bar graph displaying the top 10 referrers to content accessed through our CDN. Typically, a referrer is the URL of the web page or resource that links to your content. Detailed information is provided below the graph for the top 100 referrers.
Compression Types	Contains a donut chart that breaks down requested assets by whether they were compressed by our edge servers. The percentage of compressed assets is broken down by the type of compression used. Detailed information is provided below the graph for each compression type and status.
File Types	Contains a bar graph that displays the top 10 file types that have been requested through our CDN for your account. For the purposes of this report, a file type is defined by the asset's file name extension and Internet media type (e.g., .html [text/html], .htm [text/html], .aspx [text/html], etc.). Detailed information is provided below the graph for the top 100 file types.
Unique Files	Contains a graph that plots the total number of unique assets that were requested on a particular day over a specified period of time.
Token Auth Summary	Contains a pie chart that provides a quick overview on whether requested assets were protected by Token-Based Authentication. Protected assets are displayed in the chart according to the results of their attempted authentication.
Token Auth Deny Details	Contains a bar graph that allows you to view the top 10 requests that were denied due to Token-Based Authentication.
HTTP Response Codes	Provides a breakdown of the HTTP status codes (e.g., 200 OK, 403 Forbidden, 404 Not Found, etc.) that were delivered to your HTTP clients by our edge servers. A pie chart allows you to quickly assess how your assets were served. Detailed statistical data is provided for each response code below the graph.
404 Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a 404 Not Found response code.
403 Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a 403 Forbidden response code. A 403 Forbidden response code occurs when a request is denied by a customer origin server or an edge server on our POP.
4xx Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a response code in the 400 range. Excluded from this report are 403 Not Found and 404 Forbidden response codes. Typically, a 4xx response code occurs when a request is denied as a result of a client error.

REPORT	DESCRIPTION
504 Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a 504 Gateway Timeout response code. A 504 Gateway Timeout response code occurs when a timeout occurs when an HTTP proxy is trying to communicate with another server. In the case of our CDN, a 504 Gateway Timeout response code typically occurs when an edge server is unable to establish communication with a customer origin server.
502 Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a 502 Bad Gateway response code. A 502 Bad Gateway response code occurs when an HTTP protocol failure occurs between a server and an HTTP proxy. In the case of our CDN, a 502 Bad Gateway response code typically occurs when a customer origin server returns an invalid response to an edge server. A response is invalid if it cannot be parsed or if it is incomplete.
5xx Errors	Contains a bar graph that allows you to view the top 10 requests that resulted in a response code in the 500 range. Excluded from this report are 502 Bad Gateway and 504 Gateway Timeout response codes.

## See also

- [Azure CDN Overview](#)
- [Real-time stats in Microsoft Azure CDN](#)
- [Overriding default HTTP behavior using the rules engine](#)
- [Advanced HTTP Reports](#)

# Diagnostics Logs for Azure CDN

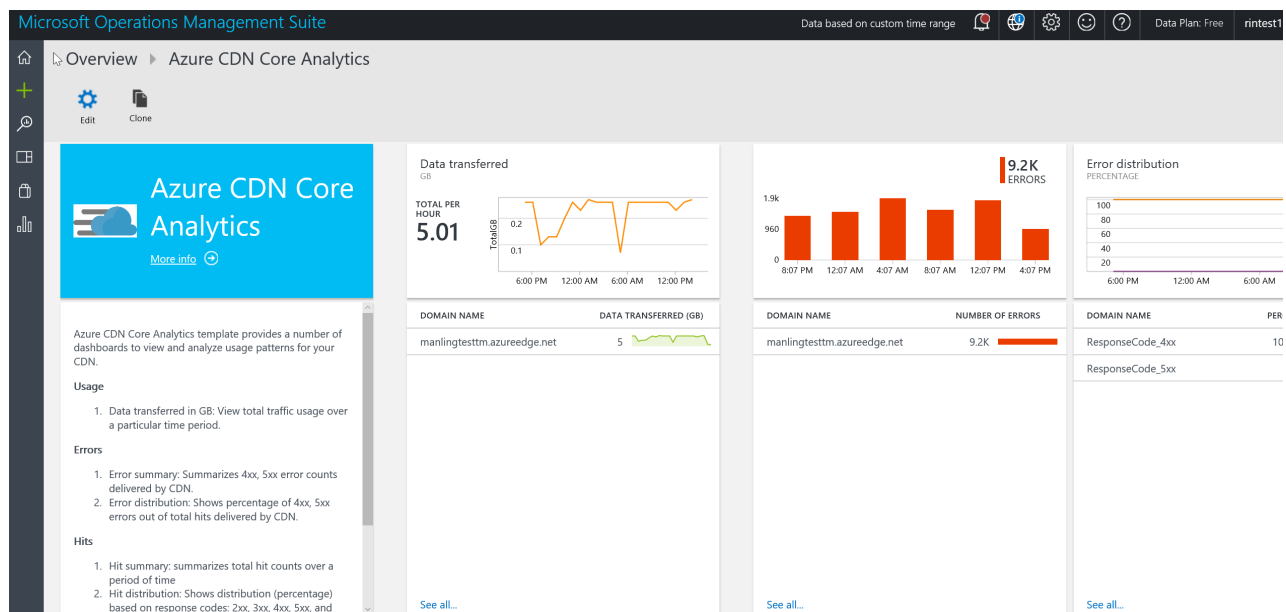
5/11/2017 • 9 min to read • [Edit Online](#)

After enabling CDN for your application, you will likely want to monitor the CDN usage, check the health of your delivery, and troubleshoot potential issues. Azure CDN provides these capabilities with [core analytics](#).

As a current Azure CDN user with Verizon standard or premium profile, you are already able to view core analytics in the supplemental portal accessible via the "Manage" option from the Azure portal. With this new [Diagnostic Logs](#) feature, you can now view core analytics and save them into one or more destinations including Azure Storage account, Azure Event hub, and/or [Log Analytics \(OMS\) workspace](#). This feature is available for all CDN endpoints belonging to Verizon (Standard & Premium) and Akamai (Standard) CDN Profiles.

Diagnostics logs allows you to export basic usage metrics from your CDN endpoint to a variety of sources so that you can consume them in a customized way. For example you can do the following:

- Export data to blob storage, export to CSV, and generate graphs in excel.
- Export data to event hubs and correlate with data from other azure services.
- Export data to log analytics and view data in your own OMS work space



The walkthrough below will go through the schema of the core analytics data, steps involved in enabling the feature and delivering them to various destinations, and consuming from these destinations.

## Enable logging with Azure portal

The diagnostics logs are turned **off** by default. Follow the steps below to enable them:

Sign in to the [Azure portal](#). If you don't already have CDN enabled for your workflow, [Enable Azure CDN](#) before you continue.

1. In the portal, navigate to **CDN profile**.
2. Select a CDN profile, then select the CDN endpoint that you want to enable **Diagnostics Logs**.

Endpoint: Manage Purge Move Delete

Essentials

Resource group (change) **ManlingAzureCDN** Pricing tier **Standard Verizon**

Status **Active**

Location **West US**

Subscription name (change) **AZURE-CDN-TEST**

Subscription ID

Endpoints

2

HOSTNAME	STATUS	PROTOCOL	ORIGIN T...	CUSTOM DOMAINS
fridaydustydogg.azureedge.net	Running	HTTP, H...	Storage	awesomefriday.dustydoggpetcare.us
saturdaydustydogg.azureedge.net	Running	HTTP, H...	Storage	awesomesaturday.dustydoggpetcare.us

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Origin

Custom domains

Compression

Cache

Geo-filtering

Locks

Automation script

MONITORING

Diagnostics logs

- Go to **Diagnostics Logs** blade Under **Monitoring** section, then change the status to **On**.

fridaydustydogg - Diagnostics logs

Endpoint

Save Discard

Status

**On** Off

☒ Archive to a storage account

☐ Stream to an event hub

☐ Send to Log Analytics

LOG

☒ CoreAnalytics

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Origin

Custom domains

Compression

Cache

Geo-filtering

Locks

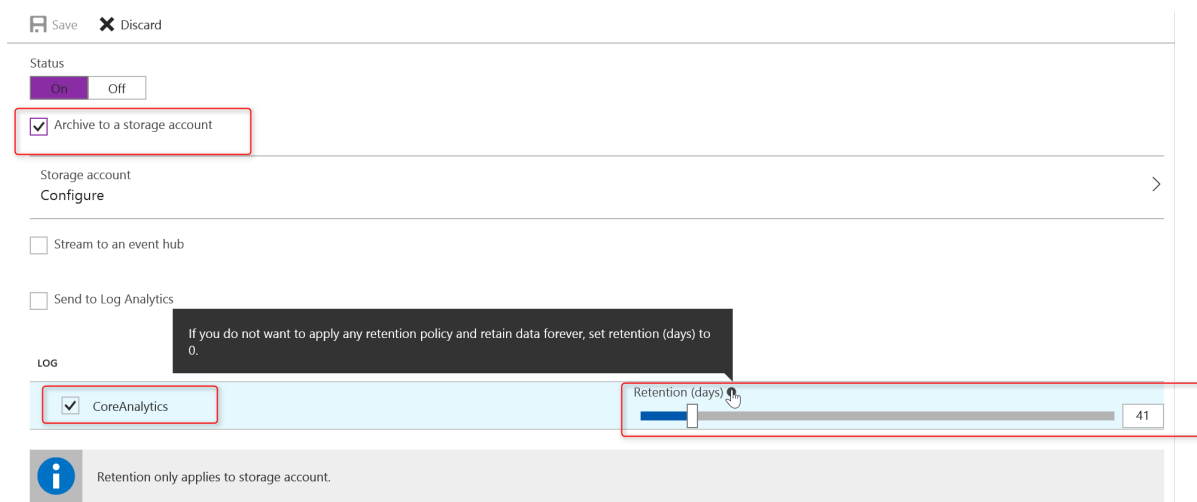
Automation script

MONITORING

Diagnostics logs

- Select and configure the desired archival target (Storage Account, Event Hub, Log Analytics).

In this example, Azure Storage is used to store the logs, select **Archive to a storage account**, select retention days, and click **CoreAnalytics** under **Log**. We only offer **CoreAnalytics** currently, but more log types will be coming in the future. See below for schema, aggregation, and delay information on CoreAnalytics.



5. Save the new diagnostics configuration.

Verizon log data is 1 hour delayed, and take up to 2 hours to start appearing after endpoint propagation completion. Akamai log data is 24 hours delayed, and takes up to 2 hours to start appearing if it was created more than 24 hours ago. If it was just created, it can take up to 25 hours for the logs to start appearing.

## Enable logging with PowerShell

Below are two examples on how to enable and get Diagnostic Logs via the Azure PowerShell Cmdlets.

### Enabling Diagnostic Logs in a Storage Account

To Enable Diagnostic Logs in a Storage Account, use this command:

```
Set-AzureRmDiagnosticSetting -ResourceId
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Cdn/profiles/{profileName}/endpoints/{endpointName}" -StorageAccountId
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ClassicStorage/storageAccounts/{storageAccountName}" -Enabled $true -Categories CoreAnalytics
```

## Consuming logs from storage

This section describes the schema of the CDN core analytics, how these are organized inside of an Azure Storage Account and provides sample code to download the logs into a CSV file.

### Using Microsoft Azure Storage Explorer

Before you can access the core analytics data from the Azure Storage Account, you first need a tool to access the contents in a storage account. While there are several tools available in the market, the one that we recommend is the Microsoft Azure Storage Explorer. You can download the tool from [here](#). After downloading and installing the software, please configure to use the same Azure Storage Account that was configured as a destination to the CDN Diagnostics Logs.

1. Open **Microsoft Azure Storage Explorer**
2. Locate the storage account
3. Go to the **"Blob Containers"** node under this storage account and expand the node
4. Select the container named **"insights-logs-coreanalytics"** and double click on it
5. Results show up on the right-hand pane starting with the first level which looks like **"resourceId="**. Continue clicking all the way until you see the file **PT1H.json**. See note below for explanation of the path.
6. Each blob **PT1H.json** represents the analytics logs for one hour for a specific CDN endpoint or its custom domain.
7. The schema of the contents of this JSON file is described in the section Schema of the Core Analytics Logs



## NOTE

### Blob path format

Core Analytics logs are generated every hour. All data for an hour are collected and stored inside a single Azure Blob as a JSON payload. The path to this Azure Blob appears as if there is a hierarchical structure. This is because the Storage explorer tool interprets '/' as a directory separator and shows the hierarchy for convenience. Actually, the whole path just represents the blob name. This name of the blob follows the following naming convention

```
resourceId=/SUBSCRIPTIONS/{Subscription Id}/RESOURCEGROUPS/{Resource Group Name}/PROVIDERS/MICROSOFT.CDN/PROFILES/{Profile Name}/ENDPOINTS/{Endpoint Name}/ y={Year}/m={Month}/d={Day}/h={Hour}/m={Minutes}/PT1H.json
```

### Description of fields:

VALUE	DESCRIPTION
Subscription Id	Id of the Azure Subscription. This is in the Guid format.
Resource	Group Name Name of the resource group to which the CDN resources belong.
Profile Name	Name of the CDN Profile
Endpoint Name	Name of the CDN Endpoint
Year	4 digit representation of the year e.g. 2017
Month	2 digit representation of the month number. 01=January.. 12=December
Day	2 digit representation of the day of the month
PT1H.json	Actual JSON file where the analytics data is stored

### Exporting the Core Analytics Data to a CSV File

To make it easy to access the Core Analytics, we provide a sample code for a tool, which allows downloading the JSON files into a flat comma separated file format which can be used to easily create charts or other aggregations.

Here is how you can use the tool:

1. Visit the github link: <https://github.com/Azure-Samples/azure-cdn-samples/tree/master/CoreAnalytics-ExportToCsv>
2. Download the code
3. Follow instructions to compile and configure
4. Run the tool
5. Resulting CSV file shows the analytics data in a simple flat hierarchy.

## Diagnostic logs types

We currently offer only the Core Analytics logs which contains metrics showing HTTP response statistics and egress statistics as seen from the CDN POPs/edges. Over time, we will add additional types of logs.

## Core Analytics Metrics Details

Below is a list of metrics available in the Core Analytics logs. Not all metrics are available from all providers, although such differences are minimal. The table below also shows if a given metric is available from a provider. Please note that the metrics are available for only those CDN endpoints that have traffic on them.

METRIC	DESCRIPTION	VERIZON	AKAMAI
RequestCountTotal	Total number of request hits during this period	Yes	Yes
RequestCountHttpStatus2xx	Count of all requests that resulted in a 2xx HTTP code (e.g. 200, 202)	Yes	Yes
RequestCountHttpStatus3xx	Count of all requests that resulted in a 3xx HTTP code (e.g. 300, 302)	Yes	Yes
RequestCountHttpStatus4xx	Count of all requests that resulted in a 4xx HTTP code (e.g. 400, 404)	Yes	Yes
RequestCountHttpStatus5xx	Count of all requests that resulted in a 5xx HTTP code (e.g. 500, 504)	Yes	Yes
RequestCountHttpStatusOthers	Count of all other HTTP codes (outside of 2xx-5xx)	Yes	Yes
RequestCountHttpStatus200	Count of all requests that resulted in a 200 HTTP code response	No	Yes
RequestCountHttpStatus206	Count of all requests that resulted in a 206 HTTP code response	No	Yes
RequestCountHttpStatus302	Count of all requests that resulted in a 302 HTTP code response	No	Yes
RequestCountHttpStatus304	Count of all requests that resulted in a 304 HTTP code response	No	Yes
RequestCountHttpStatus404	Count of all requests that resulted in a 404 HTTP code response	No	Yes
RequestCountCacheHit	Count of all requests that resulted in a Cache Hit. This means the asset was served directly from the POP to the Client.	Yes	No

METRIC	DESCRIPTION	VERIZON	AKAMAI
RequestCountCacheMiss	Count of all requests that resulted in a Cache Miss. This means the asset was not found on the POP closest to the client, and therefore was retrieved from the Origin.	Yes	No
RequestCountCacheNoCache	Count of all requests to an asset that are prevented from being cached due to a user configuration on the edge.	Yes	No
RequestCountCacheUncachable	Count of all requests to assets that are prevented from being cached by the asset's Cache-Control and Expires headers, which indicate that it should not be cached on a POP or by the HTTP client	Yes	No
RequestCountCacheOthers	Count of all requests with cache status not covered by above.	Yes	No
EgressTotal	Outbound data transfer in GB	Yes	Yes
EgressHttpStatus2xx	Outbound data transfer* for responses with 2xx HTTP status codes in GB	Yes	No
EgressHttpStatus3xx	Outbound data transfer for responses with 3xx HTTP status codes in GB	Yes	No
EgressHttpStatus4xx	Outbound data transfer for responses with 4xx HTTP status codes in GB	Yes	No
EgressHttpStatus5xx	Outbound data transfer for responses with 5xx HTTP status codes in GB	Yes	No
EgressHttpStatusOthers	Outbound data transfer for responses with other HTTP status codes in GB	Yes	No
EgressCacheHit	Outbound data transfer for responses that were delivered directly from the CDN cache on the CDN POPs/Edges	Yes	No

METRIC	DESCRIPTION	VERIZON	AKAMAI
EgressCacheMiss	Outbound data transfer for responses that were not found on the nearest POP server, and retrieved from the origin server	Yes	No
EgressCacheNoCache	Outbound data transfer for assets that are prevented from being cached due to a user configuration on the edge.	Yes	No
EgressCacheUncacheable	Outbound data transfer for assets that are prevented from being cached by the asset's Cache-Control and/or Expires headers, which indicate that it should not be cached on a POP or by the HTTP client	Yes	No
EgressCacheOthers	Outbound data transfers for other cache scenarios.	Yes	No

\*Outbound data transfer refers to traffic delivered from CDN POP servers to the client.

### Schema of the Core Analytics Logs

All logs are stored in JSON format and each entry has string fields following the below schema:

```

"records": [
  {
    "time": "2017-04-27T01:00:00",
    "resourceId": "<ARM Resource Id of the CDN Endpoint>",
    "operationName": "Microsoft.Cdn/profiles/endpoints/contentDelivery",
    "category": "CoreAnalytics",
    "properties": {
      "DomainName": "<Name of the domain for which the statistics is reported>",
      "RequestCountTotal": integer value,
      "RequestCountHttpStatus2xx": integer value,
      "RequestCountHttpStatus3xx": integer value,
      "RequestCountHttpStatus4xx": integer value,
      "RequestCountHttpStatus5xx": integer value,
      "RequestCountHttpStatusOthers": integer value,
      "RequestCountHttpStatus200": integer value,
      "RequestCountHttpStatus206": integer value,
      "RequestCountHttpStatus302": integer value,
      "RequestCountHttpStatus304": integer value,
      "RequestCountHttpStatus404": integer value,
      "RequestCountCacheHit": integer value,
      "RequestCountCacheMiss": integer value,
      "RequestCountCacheNoCache": integer value,
      "RequestCountCacheUncacheable": integer value,
      "RequestCountCacheOthers": integer value,
      "EgressTotal": double value,
      "EgressHttpStatus2xx": double value,
      "EgressHttpStatus3xx": double value,
      "EgressHttpStatus4xx": double value,
      "EgressHttpStatus5xx": double value,
      "EgressHttpStatusOthers": double value,
      "EgressCacheHit": double value,
      "EgressCacheMiss": double value,
      "EgressCacheNoCache": double value,
      "EgressCacheUncacheable": double value,
      "EgressCacheOthers": double value,
    }
  }
]
}

```

Where the 'time' represents the start time of the hour boundary for which the statistics is reported. Please note that when a metric is not supported by a CDN provider, instead of a double or integer value, there will be a null value. This null value indicates the absence of a metric, and this is different from a 0 value. Also note that there will be one set of these metrics per domain configured on the endpoint.

Example properties below:

```
{
  "DomainName": "manlingakamaitest2.azureedge.net",
  "RequestCountTotal": 480,
  "RequestCountHttpStatus2xx": 480,
  "RequestCountHttpStatus3xx": 0,
  "RequestCountHttpStatus4xx": 0,
  "RequestCountHttpStatus5xx": 0,
  "RequestCountHttpStatusOthers": 0,
  "RequestCountHttpStatus200": 480,
  "RequestCountHttpStatus206": 0,
  "RequestCountHttpStatus302": 0,
  "RequestCountHttpStatus304": 0,
  "RequestCountHttpStatus404": 0,
  "RequestCountCacheHit": null,
  "RequestCountCacheMiss": null,
  "RequestCountCacheNoCache": null,
  "RequestCountCacheUncacheable": null,
  "RequestCountCacheOthers": null,
  "EgressTotal": 0.09,
  "EgressHttpStatus2xx": null,
  "EgressHttpStatus3xx": null,
  "EgressHttpStatus4xx": null,
  "EgressHttpStatus5xx": null,
  "EgressHttpStatusOthers": null,
  "EgressCacheHit": null,
  "EgressCacheMiss": null,
  "EgressCacheNoCache": null,
  "EgressCacheUncacheable": null,
  "EgressCacheOthers": null
}
```

## Additional resources

- [Azure Diagnostic logs](#)
- [Core analytics via Azure CDN supplemental portal](#)

# Get started with Azure CDN development

1/24/2017 • 9 min to read • [Edit Online](#)

You can use the [Azure CDN Library for .NET](#) to automate creation and management of CDN profiles and endpoints. This tutorial walks through the creation of a simple .NET console application that demonstrates several of the available operations. This tutorial is not intended to describe all aspects of the Azure CDN Library for .NET in detail.

You need Visual Studio 2015 to complete this tutorial. [Visual Studio Community 2015](#) is freely available for download.

## TIP

The [completed project from this tutorial](#) is available for download on MSDN.

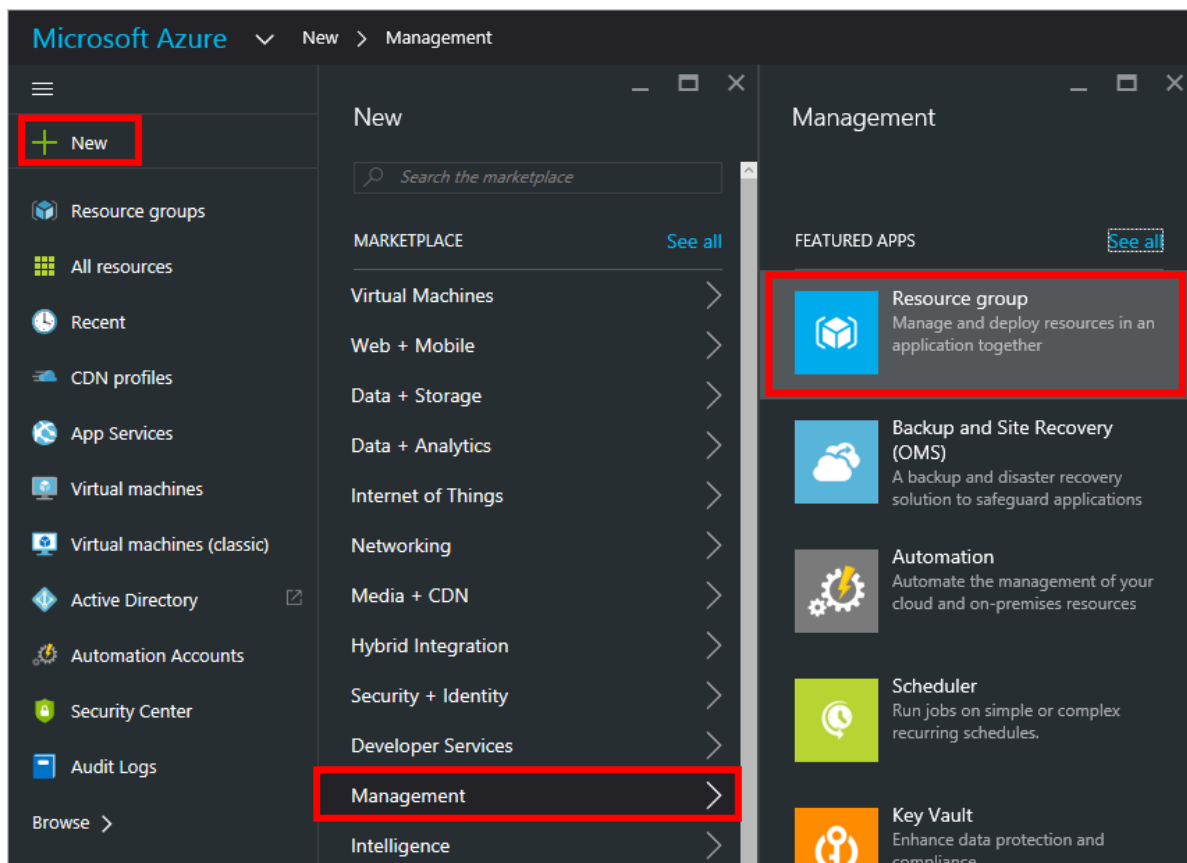
## Prerequisites

Before we can write CDN management code, we need to do some preparation to enable our code to interact with the Azure Resource Manager. To do this, you'll need to:

- Create a resource group to contain the CDN profile we create in this tutorial
- Configure Azure Active Directory to provide authentication for our application
- Apply permissions to the resource group so that only authorized users from our Azure AD tenant can interact with our CDN profile

### Creating the resource group

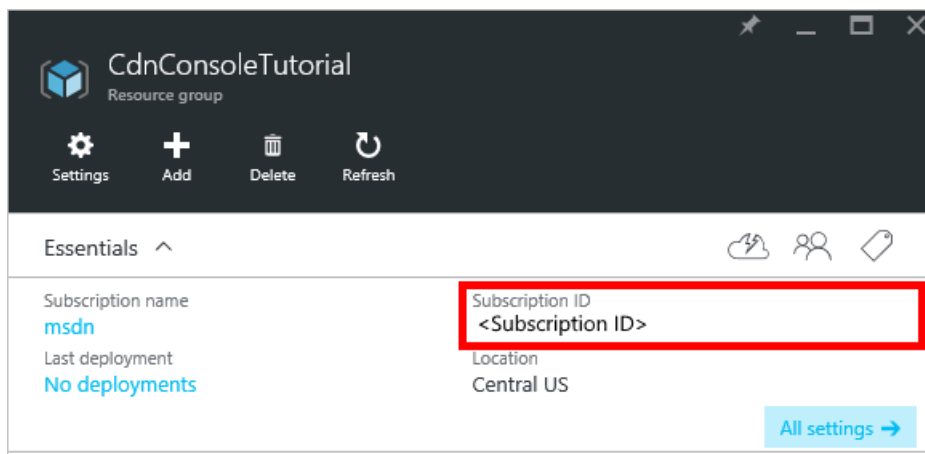
1. Log into the [Azure Portal](#).
2. Click the **New** button in the upper left, and then **Management**, and **Resource Group**.



3. Call your resource group *CdnConsoleTutorial*. Select your subscription and choose a location near you. If you wish, you may click the **Pin to dashboard** checkbox to pin the resource group to the dashboard in the portal. This will make it easier to find later. After you've made your selections, click **Create**.

4. After the resource group is created, if you didn't pin it to your dashboard, you can find it by clicking **Browse**, then **Resource Groups**. Click the resource group to open it. Make a note of your **Subscription ID**. We'll need it later.





## Creating the Azure AD application and applying permissions

There are two approaches to app authentication with Azure Active Directory: Individual users or a service principal. A service principal is similar to a service account in Windows. Instead of granting a particular user permissions to interact with the CDN profiles, we instead grant the permissions to the service principal. Service principals are generally used for automated, non-interactive processes. Even though this tutorial is writing an interactive console app, we'll focus on the service principal approach.

Creating a service principal consists of several steps, including creating an Azure Active Directory application. To do this, we're going to [follow this tutorial](#).

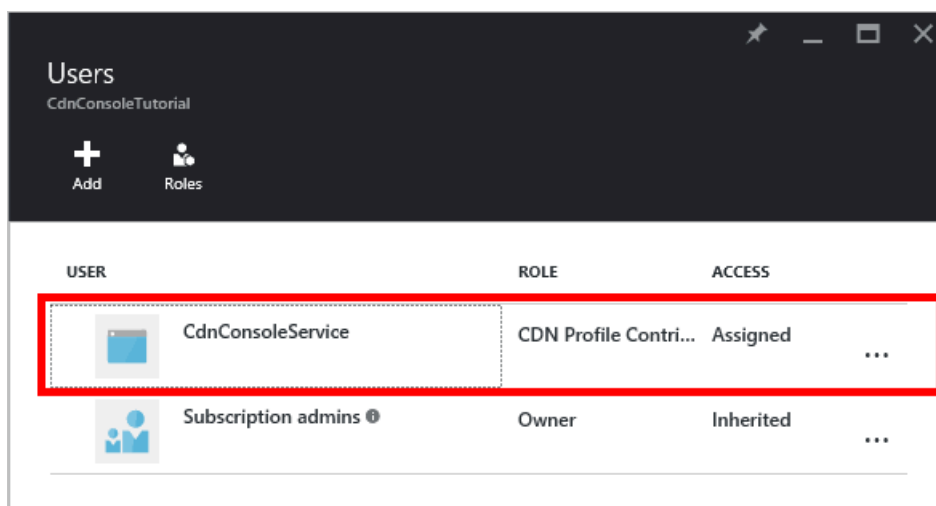
### IMPORTANT

Be sure to follow all the steps in the [linked tutorial](#). It is *extremely important* that you complete it exactly as described. Make sure to note your **tenant ID**, **tenant domain name** (commonly a `.onmicrosoft.com` domain unless you've specified a custom domain), **client ID**, and **client authentication key**, as we will need these later. Be very careful to guard your **client ID** and **client authentication key**, as these credentials can be used by anyone to execute operations as the service principal.

When you get to the step named Configure multi-tenant application, select **No**.

When you get to the step [Assign application to role](#), use the resource group we created earlier, *CdnConsoleTutorial*, but instead of the **Reader** role, assign the **CDN Profile Contributor** role. After you assign the application the **CDN Profile Contributor** role on your resource group, return to this tutorial.

Once you've created your service principal and assigned the **CDN Profile Contributor** role, the **Users** blade for your resource group should look similar to this.



## Interactive user authentication

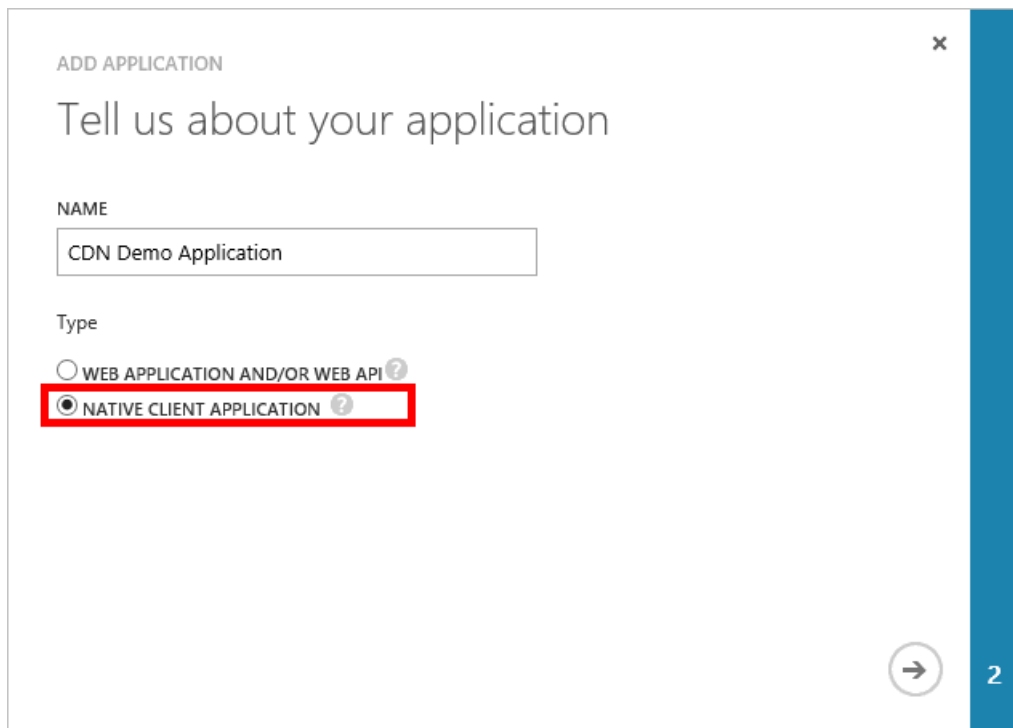
If, instead of a service principal, you'd rather have interactive individual user authentication, the process is very

similar to that for a service principal. In fact, you will need to follow the same procedure, but make a few minor changes.

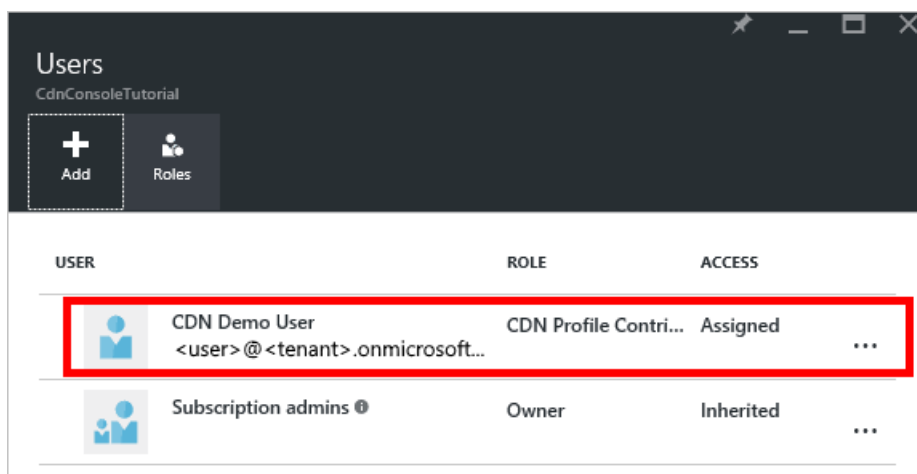
### IMPORTANT



Only follow these next steps if you are choosing to use individual user authentication instead of a service principal.

1. When creating your application, instead of **Web Application**, choose **Native application**.



2. On the next page, you will be prompted for a **redirect URI**. The URI won't be validated, but remember what you entered. You'll need it later.
3. There is no need to create a **client authentication key**.
4. Instead of assigning a service principal to the **CDN Profile Contributor** role, we're going to assign individual users or groups. In this example, you can see that I've assigned *CDN Demo User* to the **CDN Profile Contributor** role.

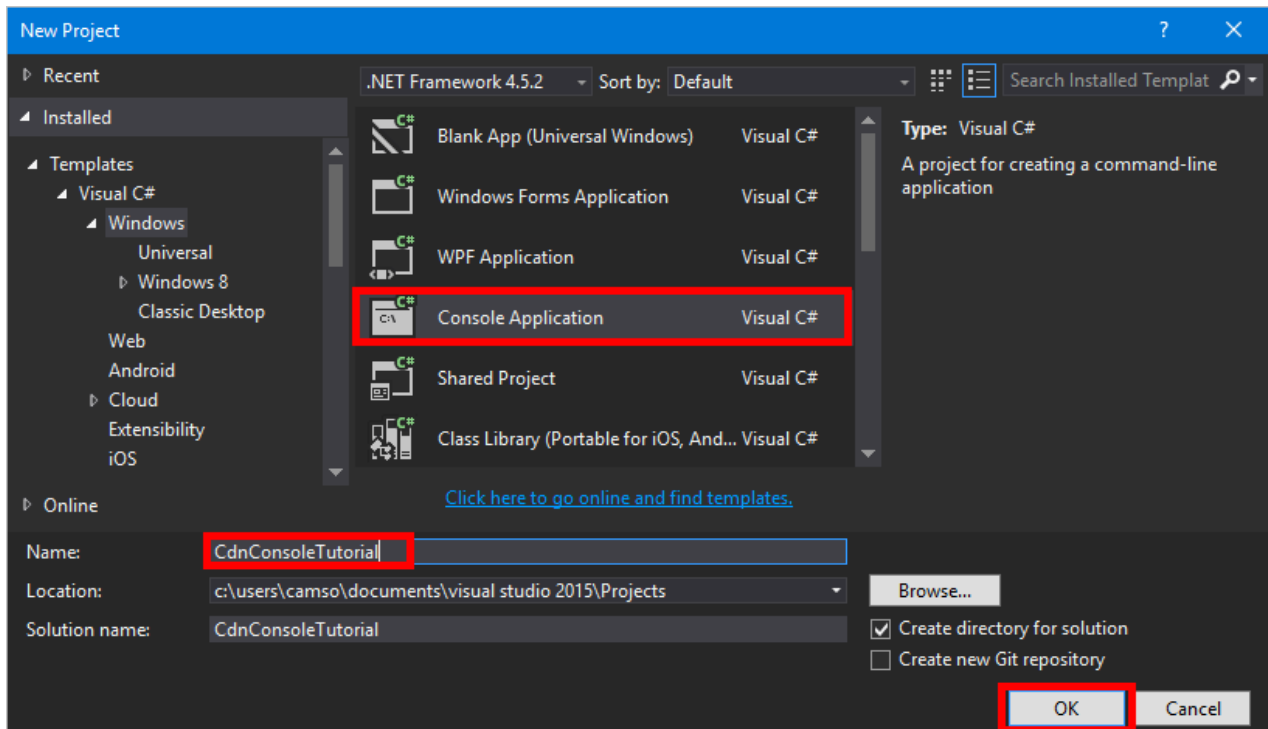


USER	ROLE	ACCESS
 CDN Demo User <user>@<tenant>.onmicrosoft...	CDN Profile Contri...	Assigned
 Subscription admins ⓘ	Owner	Inherited

## Create your project and add Nuget packages

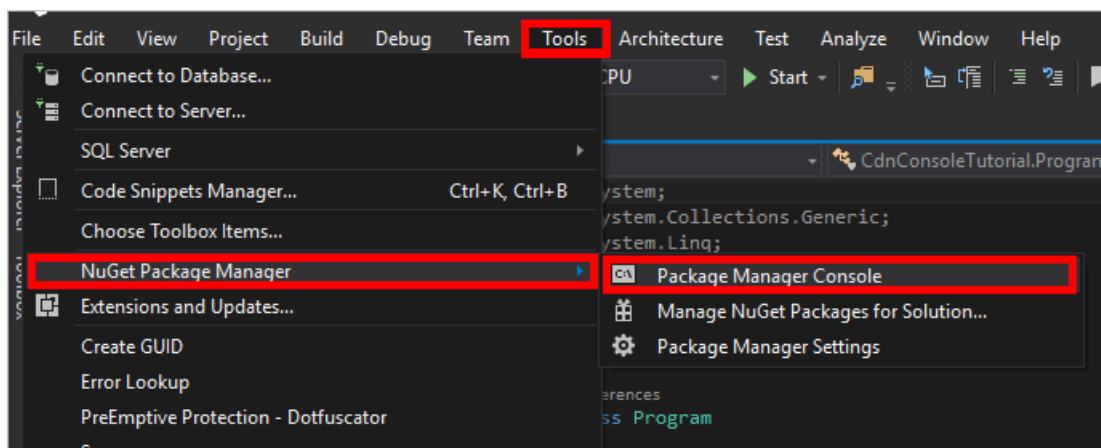
Now that we've created a resource group for our CDN profiles and given our Azure AD application permission to manage CDN profiles and endpoints within that group, we can start creating our application.

From within Visual Studio 2015, click **File, New, Project...** to open the new project dialog. Expand **Visual C#**, then select **Windows** in the pane on the left. Click **Console Application** in the center pane. Name your project, then click **OK**.



Our project is going to use some Azure libraries contained in Nuget packages. Let's add those to the project.

1. Click the **Tools** menu, **Nuget Package Manager**, then **Package Manager Console**.



2. In the Package Manager Console, execute the following command to install the **Active Directory Authentication Library (ADAL)**:

```
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

3. Execute the following to install the **Azure CDN Management Library**:

```
Install-Package Microsoft.Azure.Management.Cdn
```

## Directives, constants, main method, and helper methods

Let's get the basic structure of our program written.

1. Back in the Program.cs tab, replace the `using` directives at the top with the following:

```
using System;
using System.Collections.Generic;
using Microsoft.Azure.Management.Cdn;
using Microsoft.Azure.Management.Cdn.Models;
using Microsoft.Azure.Management.Resources;
using Microsoft.Azure.Management.Resources.Models;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.Rest;
```

2. We need to define some constants our methods will use. In the `Program` class, but before the `Main` method, add the following. Be sure to replace the placeholders, including the **<angle brackets>**, with your own values as needed.

```
//Tenant app constants
private const string clientId = "<YOUR CLIENT ID>";
private const string clientSecret = "<YOUR CLIENT AUTHENTICATION KEY>"; //Only for service principals
private const string authority = "https://login.microsoftonline.com/<YOUR TENANT ID>/<YOUR TENANT DOMAIN NAME>";

//Application constants
private const string subscriptionId = "<YOUR SUBSCRIPTION ID>";
private const string profileName = "CdnConsoleApp";
private const string endpointName = "<A UNIQUE NAME FOR YOUR CDN ENDPOINT>";
private const string resourceGroupName = "CdnConsoleTutorial";
private const string resourceLocation = "<YOUR PREFERRED AZURE LOCATION, SUCH AS Central US>";
```

3. Also at the class level, define these two variables. We'll use these later to determine if our profile and endpoint already exist.

```
static bool profileAlreadyExists = false;
static bool endpointAlreadyExists = false;
```

4. Replace the `Main` method as follows:

```

static void Main(string[] args)
{
    //Get a token
    AuthenticationResult authResult = GetAccessToken();

    // Create CDN client
    CdnManagementClient cdn = new CdnManagementClient(new TokenCredentials(authResult.AccessToken))
        { SubscriptionId = subscriptionId };

    ListProfilesAndEndpoints(cdn);

    // Create CDN Profile
    CreateCdnProfile(cdn);

    // Create CDN Endpoint
    CreateCdnEndpoint(cdn);

    Console.WriteLine();

    // Purge CDN Endpoint
    PromptPurgeCdnEndpoint(cdn);

    // Delete CDN Endpoint
    PromptDeleteCdnEndpoint(cdn);

    // Delete CDN Profile
    PromptDeleteCdnProfile(cdn);

    Console.WriteLine("Press Enter to end program.");
    Console.ReadLine();
}

```

5. Some of our other methods are going to prompt the user with "Yes/No" questions. Add the following method to make that a little easier:

```

private static bool PromptUser(string Question)
{
    Console.Write(Question + " (Y/N): ");
    var response = Console.ReadKey();
    Console.WriteLine();
    if (response.Key == ConsoleKey.Y)
    {
        return true;
    }
    else if (response.Key == ConsoleKey.N)
    {
        return false;
    }
    else
    {
        // They pressed something other than Y or N. Let's ask them again.
        return PromptUser(Question);
    }
}

```

Now that the basic structure of our program is written, we should create the methods called by the `Main` method.

## Authentication

Before we can use the Azure CDN Management Library, we need to authenticate our service principal and obtain an authentication token. This method uses ADAL to retrieve the token.

```
private static AuthenticationResult GetAccessToken()
{
    AuthenticationContext authContext = new AuthenticationContext(authority);
    ClientCredential credential = new ClientCredential(clientID, clientSecret);
    AuthenticationResult authResult =
        authContext.AcquireTokenAsync("https://management.core.windows.net/", credential).Result;

    return authResult;
}
```

If you are using individual user authentication, the `GetAccessToken` method will look slightly different.

### IMPORTANT

Only use this code sample if you are choosing to have individual user authentication instead of a service principal.

```
private static AuthenticationResult GetAccessToken()
{
    AuthenticationContext authContext = new AuthenticationContext(authority);
    AuthenticationResult authResult = authContext.AcquireTokenAsync("https://management.core.windows.net/",
        clientID, new Uri("http://<redirect URI>"), new
        PlatformParameters(PromptBehavior.RefreshSession)).Result;

    return authResult;
}
```

Be sure to replace `<redirect URI>` with the redirect URI you entered when you registered the application in Azure AD.

## List CDN profiles and endpoints

Now we're ready to perform CDN operations. The first thing our method does is list all the profiles and endpoints in our resource group, and if it finds a match for the profile and endpoint names specified in our constants, makes a note of that for later so we don't try to create duplicates.

```

private static void ListProfilesAndEndpoints(CdnManagementClient cdn)
{
    // List all the CDN profiles in this resource group
    var profileList = cdn.Profiles.ListByResourceGroup(resourceGroupName);
    foreach (Profile p in profileList)
    {
        Console.WriteLine("CDN profile {0}", p.Name);
        if (p.Name.Equals(profileName, StringComparison.OrdinalIgnoreCase))
        {
            // Hey, that's the name of the CDN profile we want to create!
            profileAlreadyExists = true;
        }

        //List all the CDN endpoints on this CDN profile
        Console.WriteLine("Endpoints:");
        var endpointList = cdn.Endpoints.ListByProfile(p.Name, resourceGroupName);
        foreach (Endpoint e in endpointList)
        {
            Console.WriteLine("-{0} ({1})", e.Name, e.HostName);
            if (e.Name.Equals(endpointName, StringComparison.OrdinalIgnoreCase))
            {
                // The unique endpoint name already exists.
                endpointAlreadyExists = true;
            }
        }
        Console.WriteLine();
    }
}

```

## Create CDN profiles and endpoints

Next, we'll create a profile.

```

private static void CreateCdnProfile(CdnManagementClient cdn)
{
    if (profileAlreadyExists)
    {
        Console.WriteLine("Profile {0} already exists.", profileName);
    }
    else
    {
        Console.WriteLine("Creating profile {0}.", profileName);
        ProfileCreateParameters profileParms =
            new ProfileCreateParameters() { Location = resourceLocation, Sku = new
Sku(SkuName.StandardVerizon) };
        cdn.Profiles.Create(profileName, profileParms, resourceGroupName);
    }
}

```

Once the profile is created, we'll create an endpoint.

```
private static void CreateCdnEndpoint(CdnManagementClient cdn)
{
    if (endpointAlreadyExists)
    {
        Console.WriteLine("Profile {0} already exists.", profileName);
    }
    else
    {
        Console.WriteLine("Creating endpoint {0} on profile {1}.", endpointName, profileName);
        EndpointCreateParameters endpointParms =
            new EndpointCreateParameters()
            {
                Origins = new List<DeepCreatedOrigin>() { new DeepCreatedOrigin("Contoso", "www.contoso.com")
            },
                IsHttpAllowed = true,
                IsHttpsAllowed = true,
                Location = resourceLocation
            };
        cdn.Endpoints.Create(endpointName, endpointParms, profileName, resourceGroupName);
    }
}
```

#### NOTE

The example above assigns the endpoint an origin named *Contoso* with a hostname `www.contoso.com`. You should change this to point to your own origin's hostname.

## Purge an endpoint

Assuming the endpoint has been created, one common task that we might want to perform in our program is purging the content in our endpoint.

```
private static void PromptPurgeCdnEndpoint(CdnManagementClient cdn)
{
    if (PromptUser(String.Format("Purge CDN endpoint {0}?", endpointName)))
    {
        Console.WriteLine("Purging endpoint. Please wait...");
        cdn.Endpoints.PurgeContent(endpointName, profileName, resourceGroupName, new List<string>() { "/" });
        Console.WriteLine("Done.");
        Console.WriteLine();
    }
}
```

#### NOTE

In the example above, the string `/` denotes that I want to purge everything in the root of the endpoint path. This is equivalent to checking **Purge All** in the Azure portal's "purge" dialog. In the `CreateCdnProfile` method, I created our profile as an **Azure CDN from Verizon** profile using the code `Sku = new Sku(SkuName.StandardVerizon)`, so this will be successful. However, **Azure CDN from Akamai** profiles do not support **Purge All**, so if I was using an Akamai profile for this tutorial, I would need to include specific paths to purge.

## Delete CDN profiles and endpoints

The last methods will delete our endpoint and profile.



```

private static void PromptDeleteCdnEndpoint(CdnManagementClient cdn)
{
    if(PromptUser(String.Format("Delete CDN endpoint {0} on profile {1}?", endpointName, profileName)))
    {
        Console.WriteLine("Deleting endpoint. Please wait...");
        cdn.Endpoints.DeleteIfExists(endpointName, profileName, resourceGroupName);
        Console.WriteLine("Done.");
        Console.WriteLine();
    }
}

private static void PromptDeleteCdnProfile(CdnManagementClient cdn)
{
    if(PromptUser(String.Format("Delete CDN profile {0}?", profileName)))
    {
        Console.WriteLine("Deleting profile. Please wait...");
        cdn.Profiles.DeleteIfExists(profileName, resourceGroupName);
        Console.WriteLine("Done.");
        Console.WriteLine();
    }
}
}

```

## Running the program

We can now compile and run the program by clicking the **Start** button in Visual Studio.

```

file:///c:/users/cams0/documents/visual studio 2015/Projects/CdnConsoleTutorial/
Creating profile CdnConsoleApp.
Creating endpoint CdnConsoleEndpoint on profile CdnConsoleApp.
Purge CDN endpoint CdnConsoleEndpoint? (Y/N): _

```

When the program reaches the above prompt, you should be able to return to your resource group in the Azure portal and see that the profile has been created.

The screenshot shows the Azure portal interface for a resource group named 'CdnConsoleTutorial'. The top navigation bar includes icons for Settings, Add, Delete, and Refresh. The 'Essentials' section provides a summary of the subscription 'msdn' and its location 'Central US'. Below this, a table lists the resources in the group. The first row, 'CdnConsoleApp', is highlighted with a red border, indicating it is the selected resource. This resource is identified as a 'CDN profile'.

NAME	TYPE	RESOURCE GRO...	LOCATION	SUBSCRIP1
CdnConsoleApp	CDN profile	CdnConsoleTu...	Central US	msdn

We can then confirm the prompts to run the rest of the program.

```
file:///c:/users/camso/documents/visual studio 2015/Projects/CdnConsoleTutorial/CdnConsoleTutor
Creating profile CdnConsoleApp.
Creating endpoint CdnConsoleEndpoint on profile CdnConsoleApp.

Purge CDN endpoint CdnConsoleEndpoint? (Y/N): y
Purging endpoint. Please wait...
Done.

Delete CDN endpoint CdnConsoleEndpoint on profile CdnConsoleApp? (Y/N): y
Deleting endpoint. Please wait...
Done.

Delete CDN profile CdnConsoleApp? (Y/N): y
Deleting profile. Please wait...
Done.

Press Enter to end program.
```

## Next Steps

To see the completed project from this walkthrough, [download the sample](#).

To find additional documentation on the Azure CDN Management Library for .NET, view the [reference on MSDN](#).

Manage your CDN resources with [PowerShell](#).

# Get started with Azure CDN development

1/24/2017 • 9 min to read • [Edit Online](#)

You can use the [Azure CDN SDK for Node.js](#) to automate creation and management of CDN profiles and endpoints. This tutorial walks through the creation of a simple Node.js console application that demonstrates several of the available operations. This tutorial is not intended to describe all aspects of the Azure CDN SDK for Node.js in detail.

To complete this tutorial, you should already have [Node.js 4.x.x](#) or higher installed and configured. You can use any text editor you want to create your Node.js application. To write this tutorial, I used [Visual Studio Code](#).

## TIP

The [completed project from this tutorial](#) is available for download on MSDN.

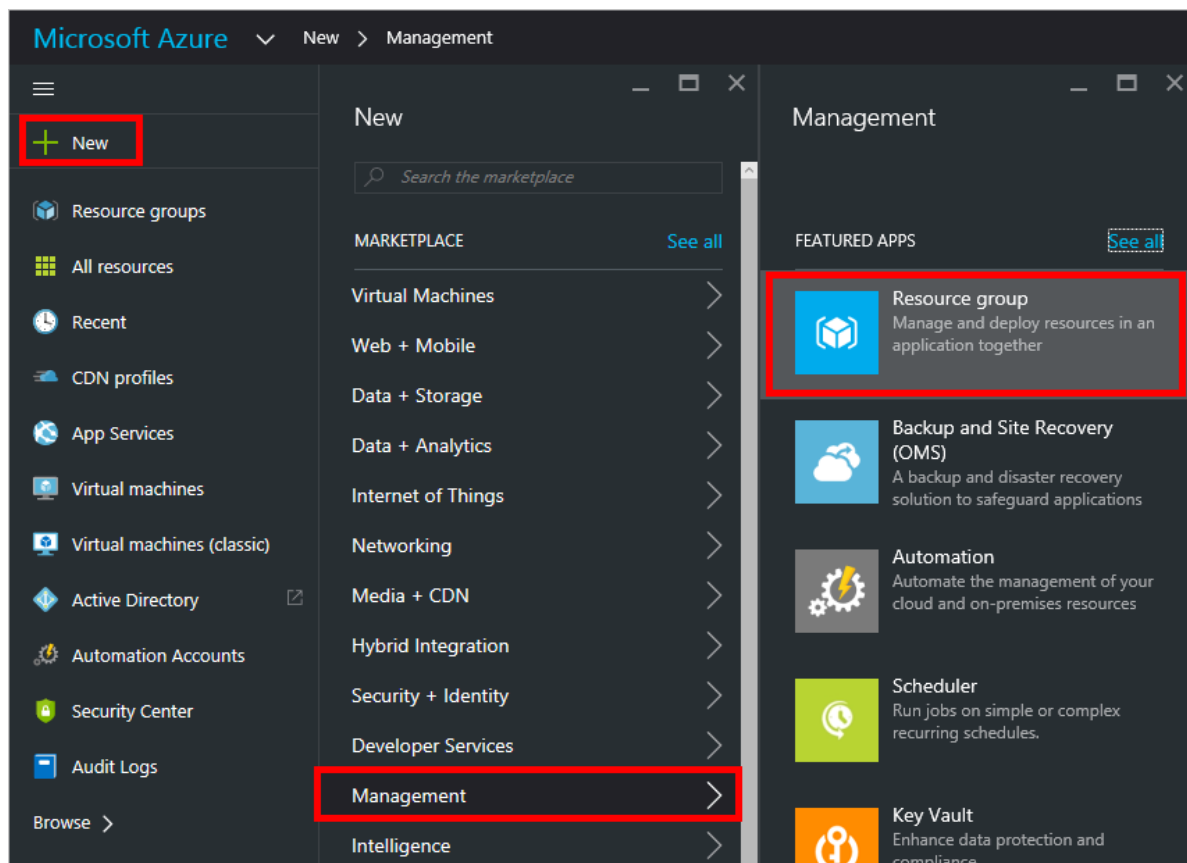
## Prerequisites

Before we can write CDN management code, we need to do some preparation to enable our code to interact with the Azure Resource Manager. To do this, you'll need to:

- Create a resource group to contain the CDN profile we create in this tutorial
- Configure Azure Active Directory to provide authentication for our application
- Apply permissions to the resource group so that only authorized users from our Azure AD tenant can interact with our CDN profile

### Creating the resource group

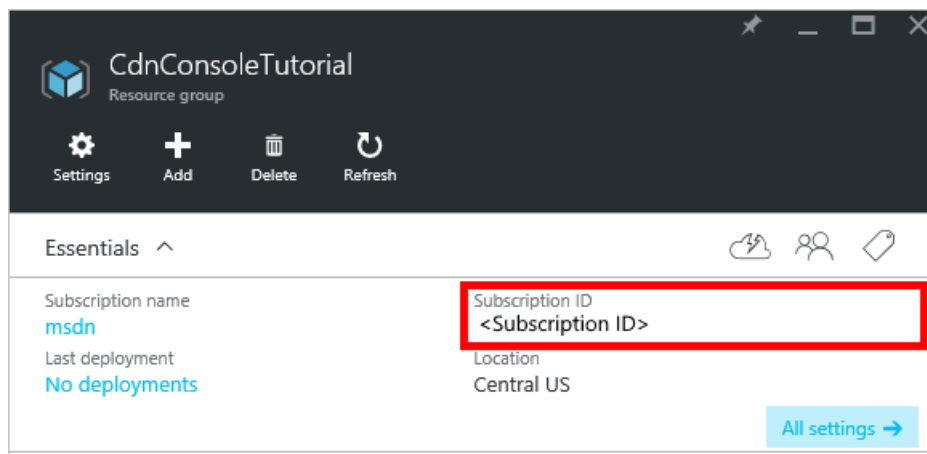
1. Log into the [Azure Portal](#).
2. Click the **New** button in the upper left, and then **Management**, and **Resource Group**.



3. Call your resource group *CdnConsoleTutorial*. Select your subscription and choose a location near you. If you wish, you may click the **Pin to dashboard** checkbox to pin the resource group to the dashboard in the portal. This will make it easier to find later. After you've made your selections, click **Create**.

The screenshot shows the 'Resource group' creation form. The title is 'Resource group' with the subtitle 'Create an empty resource group'. The form contains three required fields: 'Resource group name' (with the value 'CdnConsoleTutorial'), 'Subscription' (with the value 'msdn'), and 'Resource group location' (with the value 'Central US'). Below these fields is a checkbox labeled 'Pin to dashboard' which is currently unchecked. At the bottom of the form is a blue 'Create' button.

4. After the resource group is created, if you didn't pin it to your dashboard, you can find it by clicking **Browse**, then **Resource Groups**. Click the resource group to open it. Make a note of your **Subscription ID**. We'll need it later.



## Creating the Azure AD application and applying permissions

There are two approaches to app authentication with Azure Active Directory: Individual users or a service principal. A service principal is similar to a service account in Windows. Instead of granting a particular user permissions to interact with the CDN profiles, we instead grant the permissions to the service principal. Service principals are generally used for automated, non-interactive processes. Even though this tutorial is writing an interactive console app, we'll focus on the service principal approach.

Creating a service principal consists of several steps, including creating an Azure Active Directory application. To do this, we're going to [follow this tutorial](#).

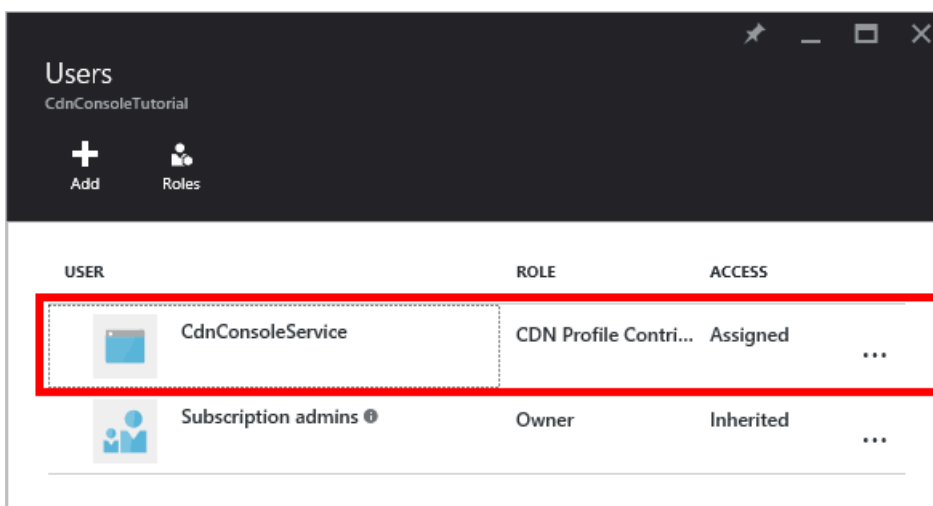
### IMPORTANT

Be sure to follow all the steps in the [linked tutorial](#). It is *extremely important* that you complete it exactly as described. Make sure to note your **tenant ID**, **tenant domain name** (commonly a `.onmicrosoft.com` domain unless you've specified a custom domain), **client ID**, and **client authentication key**, as we will need these later. Be very careful to guard your **client ID** and **client authentication key**, as these credentials can be used by anyone to execute operations as the service principal.

When you get to the step named Configure multi-tenant application, select **No**.

When you get to the step [Assign application to role](#), use the resource group we created earlier, `CdnConsoleTutorial`, but instead of the **Reader** role, assign the **CDN Profile Contributor** role. After you assign the application the **CDN Profile Contributor** role on your resource group, return to this tutorial.

Once you've created your service principal and assigned the **CDN Profile Contributor** role, the **Users** blade for your resource group should look similar to this.



## Interactive user authentication

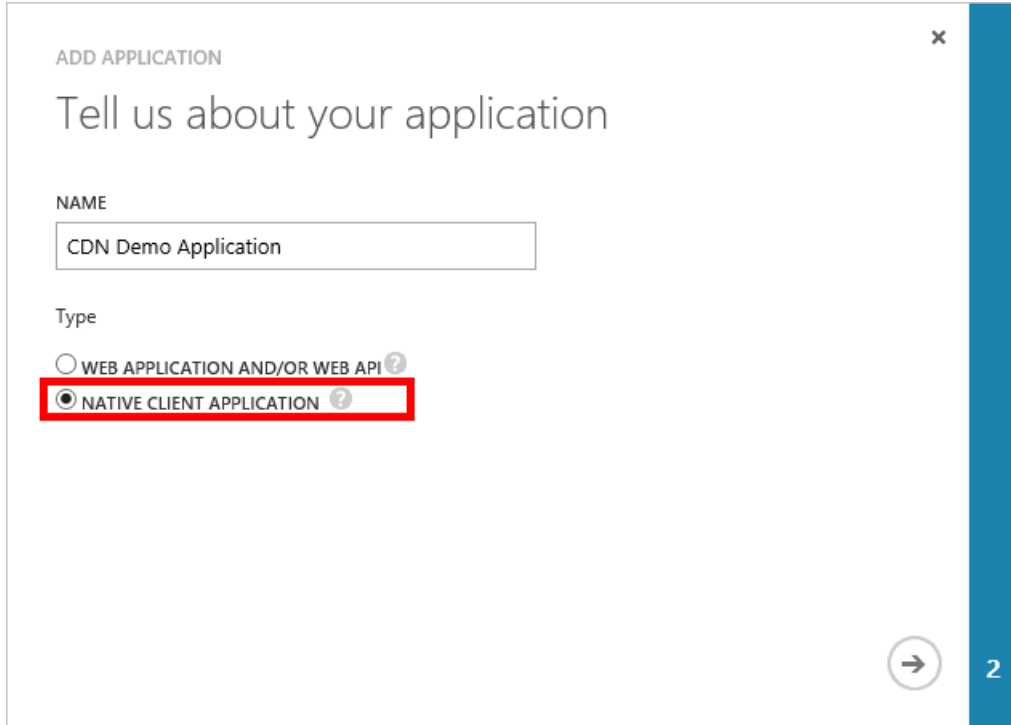
If, instead of a service principal, you'd rather have interactive individual user authentication, the process is very

similar to that for a service principal. In fact, you will need to follow the same procedure, but make a few minor changes.

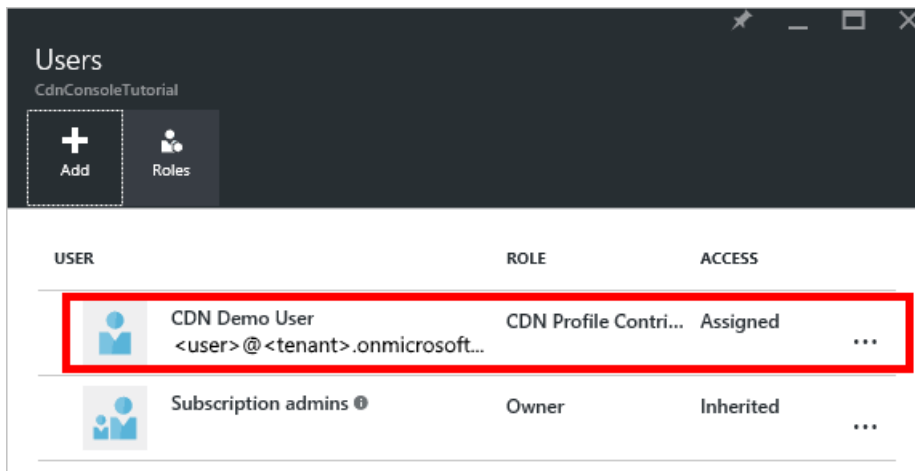
#### IMPORTANT



Only follow these next steps if you are choosing to use individual user authentication instead of a service principal.

1. When creating your application, instead of **Web Application**, choose **Native application**.



2. On the next page, you will be prompted for a **redirect URI**. The URI won't be validated, but remember what you entered. You'll need it later.
3. There is no need to create a **client authentication key**.
4. Instead of assigning a service principal to the **CDN Profile Contributor** role, we're going to assign individual users or groups. In this example, you can see that I've assigned *CDN Demo User* to the **CDN Profile Contributor** role.



USER	ROLE	ACCESS
 CDN Demo User <user>@<tenant>.onmicrosoft...	CDN Profile Contri...	Assigned
 Subscription admins	Owner	Inherited

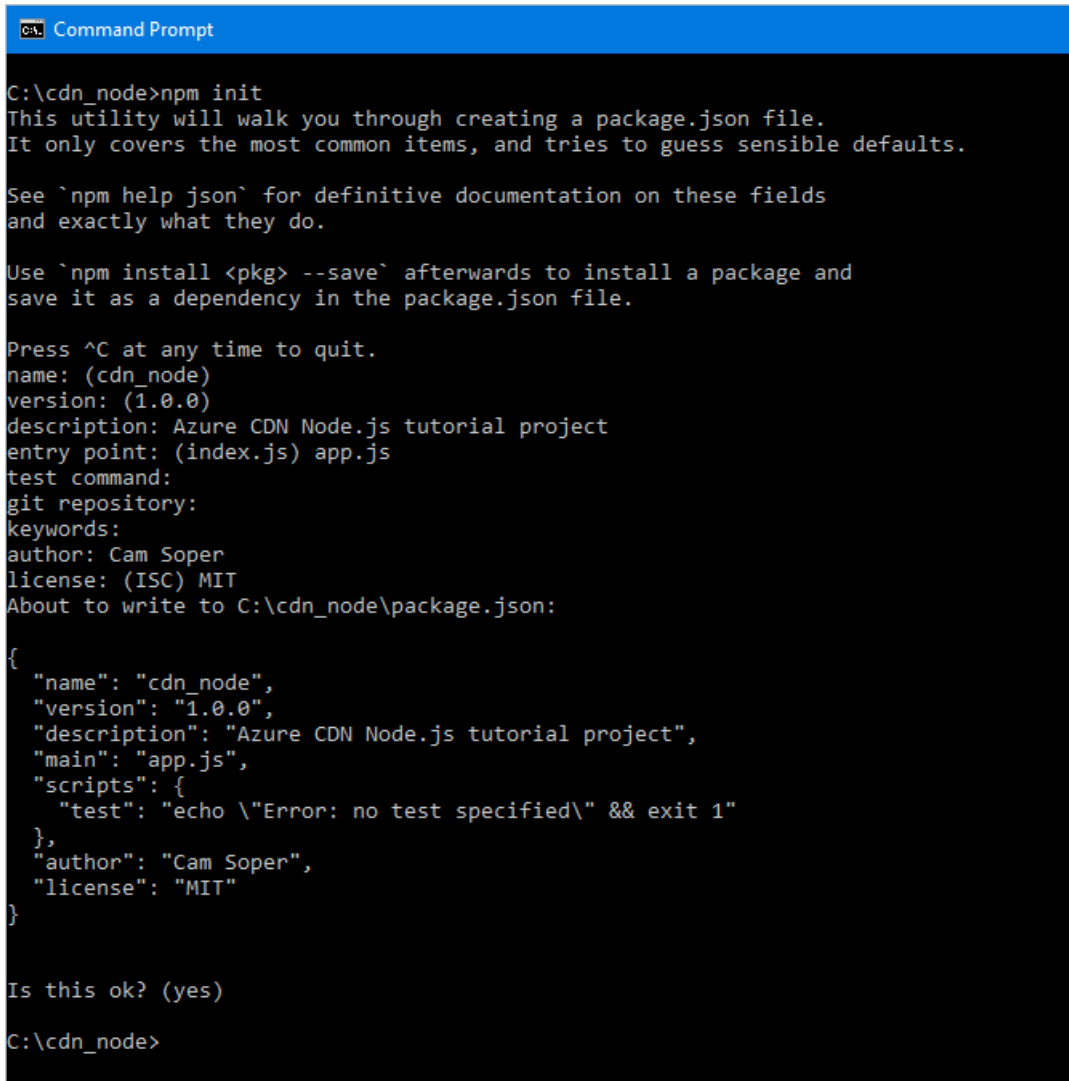
## Create your project and add NPM dependencies

Now that we've created a resource group for our CDN profiles and given our Azure AD application permission to manage CDN profiles and endpoints within that group, we can start creating our application.

Create a folder to store your application. From a console with the Node.js tools in your current path, set your current location to this new folder and initialize your project by executing:

```
npm init
```

You will then be presented a series of questions to initialize your project. For **entry point**, this tutorial uses *app.js*. You can see my other choices in the following example.

A screenshot of a Windows Command Prompt window with a blue title bar that says "Command Prompt". The window shows the output of the 'npm init' command. It starts with the prompt "C:\cdn\_node>npm init". The output text includes: "This utility will walk you through creating a package.json file. It only covers the most common items, and tries to guess sensible defaults.", "See `npm help json` for definitive documentation on these fields and exactly what they do.", "Use `npm install <pkg> --save` afterwards to install a package and save it as a dependency in the package.json file.", "Press ^C at any time to quit.", followed by a series of prompts and answers: "name: (cdn\_node)", "version: (1.0.0)", "description: Azure CDN Node.js tutorial project", "entry point: (index.js) app.js", "test command:", "git repository:", "keywords:", "author: Cam Soper", "license: (ISC) MIT". Then it says "About to write to C:\cdn\_node\package.json:" and displays a JSON object: {"name": "cdn\_node", "version": "1.0.0", "description": "Azure CDN Node.js tutorial project", "main": "app.js", "scripts": {"test": "echo \"Error: no test specified\" && exit 1"}, "author": "Cam Soper", "license": "MIT"}. It then asks "Is this ok? (yes)" and finally shows the prompt "C:\cdn\_node>".

```
C:\cdn_node>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (cdn_node)
version: (1.0.0)
description: Azure CDN Node.js tutorial project
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Cam Soper
license: (ISC) MIT
About to write to C:\cdn_node\package.json:
{
  "name": "cdn_node",
  "version": "1.0.0",
  "description": "Azure CDN Node.js tutorial project",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Cam Soper",
  "license": "MIT"
}

Is this ok? (yes)
C:\cdn_node>
```

Our project is now initialized with a *package.json* file. Our project is going to use some Azure libraries contained in NPM packages. We'll use the Azure Client Runtime for Node.js (ms-rest-azure) and the Azure CDN Client Library for Node.js (azure-arm-cdn). Let's add those to the project as dependencies.

```
npm install --save ms-rest-azure
npm install --save azure-arm-cdn
```

After the packages are done installing, the *package.json* file should look similar to this example (version numbers may differ):

```
{
  "name": "cdn_node",
  "version": "1.0.0",
  "description": "Azure CDN Node.js tutorial project",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Cam Soper",
  "license": "MIT",
  "dependencies": {
    "azure-arm-cdn": "^0.2.1",
    "ms-rest-azure": "^1.14.4"
  }
}
```

Finally, using your text editor, create a blank text file and save it in the root of our project folder as *app.js*. We're now ready to begin writing code.

## Requires, constants, authentication, and structure

With *app.js* open in our editor, let's get the basic structure of our program written.

1. Add the "requires" for our NPM packages at the top with the following:

```
var msRestAzure = require('ms-rest-azure');
var cdnManagementClient = require('azure-arm-cdn');
```

2. We need to define some constants our methods will use. Add the following. Be sure to replace the placeholders, including the **<angle brackets>**, with your own values as needed.

```
//Tenant app constants
const clientId = "<YOUR CLIENT ID>";
const clientSecret = "<YOUR CLIENT AUTHENTICATION KEY>"; //Only for service principals
const tenantId = "<YOUR TENANT ID>";

//Application constants
const subscriptionId = "<YOUR SUBSCRIPTION ID>";
const resourceGroupName = "CdnConsoleTutorial";
const resourceLocation = "<YOUR PREFERRED AZURE LOCATION, SUCH AS Central US>";
```

3. Next, we'll instantiate the CDN management client and give it our credentials.

```
var credentials = new msRestAzure.ApplicationTokenCredentials(clientId, tenantId, clientSecret);
var cdnClient = new cdnManagementClient(credentials, subscriptionId);
```

If you are using individual user authentication, these two lines will look slightly different.

### IMPORTANT

Only use this code sample if you are choosing to have individual user authentication instead of a service principal. Be careful to guard your individual user credentials and keep them secret.



```
var credentials = new msRestAzure.UserTokenCredentials(clientId,
    tenantId, '<username>', '<password>', '<redirect URI>');
var cdnClient = new cdnManagementClient(credentials, subscriptionId);
```

Be sure to replace the items in **<angle brackets>** with the correct information. For `<redirect URI>`, use the redirect URI you entered when you registered the application in Azure AD.

4. Our Node.js console application is going to take some command-line parameters. Let's validate that at least one parameter was passed.

```
//Collect command-line parameters
var parms = process.argv.slice(2);

//Do we have parameters?
if(parms == null || parms.length == 0)
{
    console.log("Not enough parameters!");
    console.log("Valid commands are list, delete, create, and purge.");
    process.exit(1);
}
```

5. That brings us to the main part of our program, where we branch off to other functions based on what parameters were passed.

```
switch(parms[0].toLowerCase())
{
    case "list":
        cdnList();
        break;

    case "create":
        cdnCreate();
        break;

    case "delete":
        cdnDelete();
        break;

    case "purge":
        cdnPurge();
        break;

    default:
        console.log("Valid commands are list, delete, create, and purge.");
        process.exit(1);
}
```

6. At several places in our program, we'll need to make sure the right number of parameters were passed in and display some help if they don't look correct. Let's create functions to do that.

```

function requireParms(parmCount) {
    if(parms.length < parmCount) {
        usageHelp(parms[0].toLowerCase());
        process.exit(1);
    }
}

function usageHelp(cmd) {
    console.log("Usage for " + cmd + ":");
    switch(cmd)
    {
        case "list":
            console.log("list profiles");
            console.log("list endpoints <profile name>");
            break;

        case "create":
            console.log("create profile <profile name>");
            console.log("create endpoint <profile name> <endpoint name> <origin hostname>");
            break;

        case "delete":
            console.log("delete profile <profile name>");
            console.log("delete endpoint <profile name> <endpoint name>");
            break;

        case "purge":
            console.log("purge <profile name> <endpoint name> <path>");
            break;

        default:
            console.log("Invalid command.");
    }
}

```

7. Finally, the functions we'll be using on the CDN management client are asynchronous, so they need a method to call back when they're done. Let's make one that can display the output from the CDN management client (if any) and exit the program gracefully.

```

function callback(err, result, request, response) {
    if (err) {
        console.log(err);
        process.exit(1);
    } else {
        console.log((result == null) ? "Done!" : result);
        process.exit(0);
    }
}

```

Now that the basic structure of our program is written, we should create the functions called based on our parameters.

## List CDN profiles and endpoints

Let's start with code to list our existing profiles and endpoints. My code comments provide the expected syntax so we know where each parameter goes.

```
// list profiles
// list endpoints <profile name>
function cdnList(){
  requireParams(2);
  switch(params[1].toLowerCase())
  {
    case "profiles":
      console.log("Listing profiles...");
      cdnClient.profiles.listByResourceGroup(resourceGroupName, callback);
      break;

    case "endpoints":
      requireParams(3);
      console.log("Listing endpoints...");
      cdnClient.endpoints.listByProfile(params[2], resourceGroupName, callback);
      break;

    default:
      console.log("Invalid parameter.");
      process.exit(1);
  }
}
```

## Create CDN profiles and endpoints

Next, we'll write the functions to create profiles and endpoints.

```

function cdnCreate() {
  requireParams(2);
  switch(params[1].toLowerCase())
  {
    case "profile":
      cdnCreateProfile();
      break;

    case "endpoint":
      cdnCreateEndpoint();
      break;

    default:
      console.log("Invalid parameter.");
      process.exit(1);
  }
}

// create profile <profile name>
function cdnCreateProfile() {
  requireParams(3);
  console.log("Creating profile...");
  var standardCreateParameters = {
    location: resourceLocation,
    sku: {
      name: 'Standard_Verizon'
    }
  };

  cdnClient.profiles.create(params[2], standardCreateParameters, resourceGroupName, callback);
}

// create endpoint <profile name> <endpoint name> <origin hostname>
function cdnCreateEndpoint() {
  requireParams(5);
  console.log("Creating endpoint...");
  var endpointProperties = {
    location: resourceLocation,
    origins: [{
      name: params[4],
      hostName: params[4]
    }]
  };

  cdnClient.endpoints.create(params[3], endpointProperties, params[2], resourceGroupName, callback);
}

```

## Purge an endpoint

Assuming the endpoint has been created, one common task that we might want to perform in our program is purging content in our endpoint.

```

// purge <profile name> <endpoint name> <path>
function cdnPurge() {
  requireParams(4);
  console.log("Purging endpoint...");
  var purgeContentPaths = [ params[3] ];
  cdnClient.endpoints.purgeContent(params[2], params[1], resourceGroupName, purgeContentPaths, callback);
}

```

## Delete CDN profiles and endpoints

The last function we will include deletes endpoints and profiles.

```
function cdnDelete() {
  requireParams(2);
  switch(params[1].toLowerCase())
  {
    // delete profile <profile name>
    case "profile":
      requireParams(3);
      console.log("Deleting profile...");
      cdnClient.profiles.deleteIfExists(params[2], resourceGroupName, callback);
      break;

    // delete endpoint <profile name> <endpoint name>
    case "endpoint":
      requireParams(4);
      console.log("Deleting endpoint...");
      cdnClient.endpoints.deleteIfExists(params[3], params[2], resourceGroupName, callback);
      break;

    default:
      console.log("Invalid parameter.");
      process.exit(1);
  }
}
```

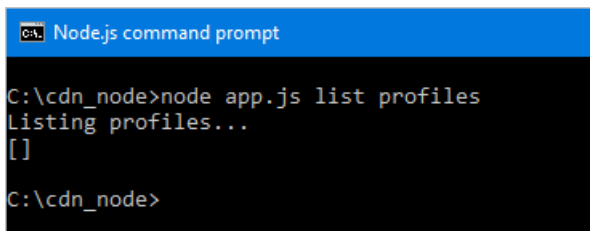
## Running the program

We can now execute our Node.js program using our favorite debugger or at the console.

### TIP

If you're using Visual Studio Code as your debugger, you'll need to set up your environment to pass in the command-line parameters. Visual Studio Code does this in the **lanuch.json** file. Look for a property named **args** and add an array of string values for your parameters, so that it looks similar to this: `"args": ["list", "profiles"]`.

Let's start by listing our profiles.



```
C:\> Node.js command prompt

C:\cdn_node>node app.js list profiles
Listing profiles...
[]

C:\cdn_node>
```

We got back an empty array. Since we don't have any profiles in our resource group, that's expected. Let's create a profile now.

```

CA. Node.js command prompt

C:\cdn_node>node app.js create profile MyNodeCdnProfile
Creating profile...
{ id: '/subscriptions/<subscription ID>/resourcegroups/CdnCons
  name: 'MyNodeCdnProfile',
  type: 'Microsoft.Cdn/profiles',
  location: 'CentralUs',
  tags: {},
  sku: { name: 'Standard_Verizon' },
  resourceState: 'Active',
  provisioningState: 'Succeeded' }

C:\cdn_node>

```

Now, let's add an endpoint.

```

CA. Node.js command prompt

C:\cdn_node>node app.js create endpoint MyNodeCdnProfile cdnnodedemo www.contoso.com
Creating endpoint...
{ id: '/subscriptions/<subscription ID>/resourcegroups/CdnConsoleTutorial/providers/Microso
  name: 'cdnnodedemo',
  type: 'Microsoft.Cdn/profiles/endpoints',
  location: 'CentralUs',
  tags: {},
  hostName: 'cdnnodedemo.azureedge.net',
  contentTypesToCompress: [],
  isCompressionEnabled: false,
  isHttpAllowed: true,
  isHttpsAllowed: true,
  queryStringCachingBehavior: 'IgnoreQueryString',
  origins: [ { name: 'www.contoso.com', hostName: 'www.contoso.com' } ],
  resourceState: 'Running',
  provisioningState: 'Succeeded' }

C:\cdn_node>

```

Finally, let's delete our profile.

```

CA. Node.js command prompt

C:\cdn_node>node app.js delete profile MyNodeCdnProfile
Deleting profile...
Done!

C:\cdn_node>

```

## Next Steps

To see the completed project from this walkthrough, [download the sample](#).

To see the reference for the Azure CDN SDK for Node.js, view the [reference](#).

To find additional documentation on the Azure SDK for Node.js, view the [full reference](#).

Manage your CDN resources with [PowerShell](#).

# Troubleshooting CDN endpoints returning 404 statuses

1/24/2017 • 6 min to read • [Edit Online](#)

This article helps you troubleshoot issues with [CDN endpoints](#) returning 404 errors.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and the Stack Overflow forums](#). Alternatively, you can also file an Azure support incident. Go to the [Azure Support site](#) and click on **Get Support**.

## Symptom

You've created a CDN profile and an endpoint, but your content doesn't seem to be available on the CDN. Users who attempt to access your content via the CDN URL receive HTTP 404 status codes.

## Cause

There are several possible causes, including:

- The file's origin isn't visible to the CDN
- The endpoint is misconfigured, causing the CDN to look in the wrong place
- The host is rejecting the host header from the CDN
- The endpoint hasn't had time to propagate throughout the CDN

## Troubleshooting steps

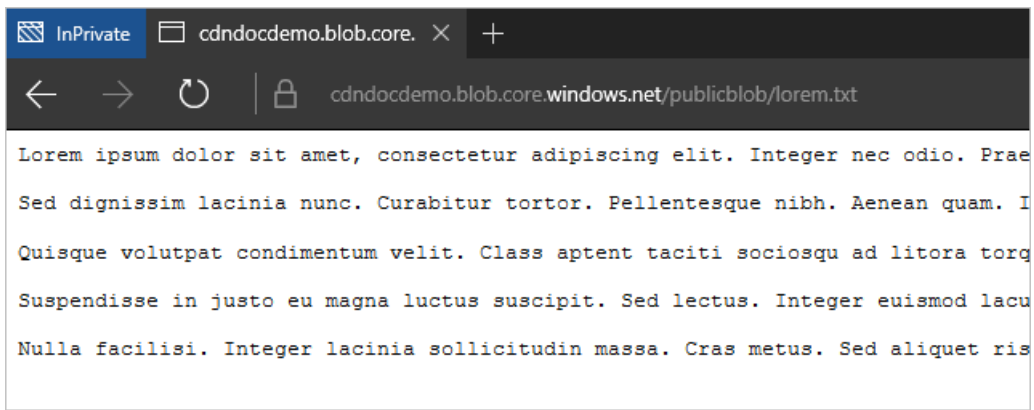
### IMPORTANT

After creating a CDN endpoint, it will not immediately be available for use, as it takes time for the registration to propagate through the CDN. For **Azure CDN from Akamai** profiles, propagation usually completes within one minute. For **Azure CDN from Verizon** profiles, propagation will usually complete within 90 minutes, but in some cases can take longer. If you complete the steps in this document and you're still getting 404 responses, consider waiting a few hours to check again before opening a support ticket.

### Check the origin file

First, we should verify that the file we want cached is available on our origin and is publicly accessible. The quickest way to do that is to open a browser in an In-Private or Incognito session and browse directly to the file. Just type or paste the URL into the address box and see if that results in the file you expect. For this example, I'm going to use a file I have in an Azure Storage account, accessible at

`https://cdndocdemo.blob.core.windows.net/publicblob/lorem.txt`. As you can see, it successfully passes the test.

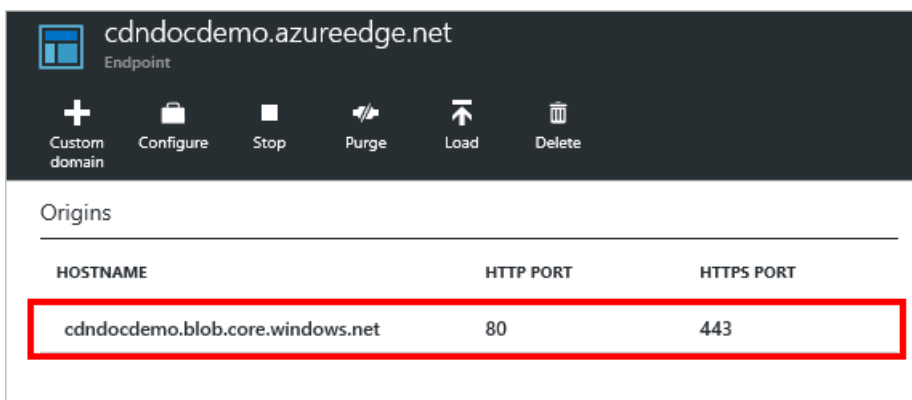


### WARNING

While this is the quickest and easiest way to verify your file is publicly available, some network configurations in your organization could give you the illusion that this file is publicly available when it is, in fact, only visible to users of your network (even if it's hosted in Azure). If you have an external browser from which you can test, such as a mobile device that is not connected to your organization's network, or a virtual machine in Azure, that would be best.

### Check the origin settings

Now that we've verified the file is publicly available on the internet, we should verify our origin settings. In the [Azure Portal](#), browse to your CDN profile and click the endpoint you're troubleshooting. In the resulting **Endpoint** blade, click the origin.



The **Origin** blade appears.



cdndocdemo.blob.core.windows...

Origin

Save Discard

\* Origin type

Storage

\* Origin hostname

cdndocdemo.blob.core.windows.net

\* HTTP port

80

\* HTTPS port

443

### Origin type and hostname

Verify the **Origin type** is correct, and verify the **Origin hostname**. In my example,

`https://cdndocdemo.blob.core.windows.net/publicblob/lorem.txt`, the hostname portion of the URL is

`cdndocdemo.blob.core.windows.net`. As you can see in the screenshot, this is correct. For Azure Storage, Web App,

and Cloud Service origins, the **Origin hostname** field is a dropdown, so we don't need to worry about spelling it correctly. However, if you're using a custom origin, it is *absolutely critical* your hostname is spelled correctly!

### HTTP and HTTPS ports

The other thing to check here is your **HTTP** and **HTTPS** ports. In most cases, 80 and 443 are correct, and you will require no changes. However, if the origin server is listening on a different port, that will need to be represented here. If you're not sure, just look at the URL for your origin file. The HTTP and HTTPS specifications specify ports 80 and 443 as the defaults. In my URL, `https://cdndocdemo.blob.core.windows.net/publicblob/lorem.txt`, a port is not specified, so the default of 443 is assumed and my settings are correct.

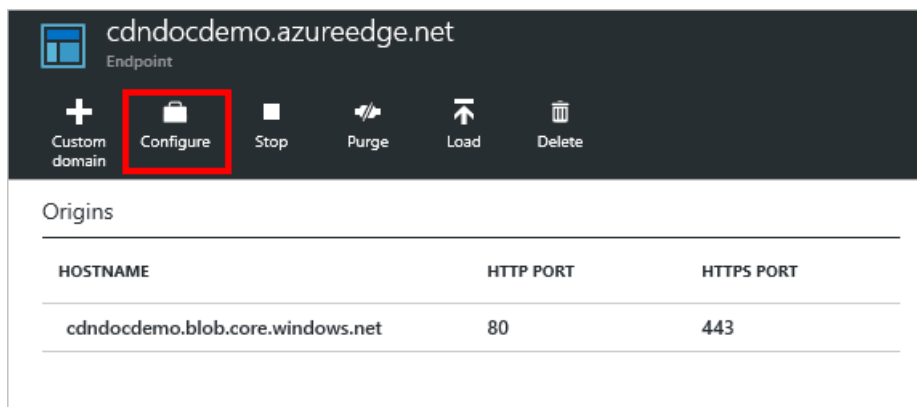
However, say the URL for your origin file that you tested earlier is `http://www.contoso.com:8080/file.txt`. Note the `:8080` at the end of the hostname segment. That tells the browser to use port `8080` to connect to the web server at `www.contoso.com`, so you'll need to enter 8080 in the **HTTP port** field. It's important to note that these port settings only affect what port the endpoint uses to retrieve information from the origin.

#### NOTE

**Azure CDN from Akamai** endpoints do not allow the full TCP port range for origins. For a list of origin ports that are not allowed, see [Azure CDN from Akamai Allowed Origin Ports](#).

### Check the endpoint settings

Back on the **Endpoint** blade, click the **Configure** button.



The endpoint's **Configure** blade appears.

Save Discard

Compression ⓘ

Query string caching behavior ⓘ  
 Ignore query strings ▼

Protocols ⓘ  
☒ HTTP  
☒ HTTPS

Origin host header ⓘ  
 cdndocdemo.blob.core.windows.net

Origin path ⓘ  
 /Path

### Protocols

For **Protocols**, verify that the protocol being used by the clients is selected. The same protocol used by the client will be the one used to access the origin, so it's important to have the origin ports configured correctly in the previous section. The endpoint only listens on the default HTTP and HTTPS ports (80 and 443), regardless of the origin ports.

Let's return to our hypothetical example with `http://www.contoso.com:8080/file.txt`. As you'll remember, Contoso specified `8080` as their HTTP port, but let's also assume they specified `44300` as their HTTPS port. If they created an endpoint named `contoso`, their CDN endpoint hostname would be `contoso.azureedge.net`. A request for `http://contoso.azureedge.net/file.txt` is an HTTP request, so the endpoint would use HTTP on port 8080 to retrieve it from the origin. A secure request over HTTPS, `https://contoso.azureedge.net/file.txt`, would cause the endpoint to use HTTPS on port 44300 when retrieving the file from the origin.

### Origin host header

The **Origin host header** is the host header value sent to the origin with each request. In most cases, this should be the same as the **Origin hostname** we verified earlier. An incorrect value in this field won't generally cause 404 statuses, but is likely to cause other 4xx statuses, depending on what the origin expects.

#### Origin path

Lastly, we should verify our **Origin path**. By default this is blank. You should only use this field if you want to narrow the scope of the origin-hosted resources you want to make available on the CDN.

For example, in my endpoint, I wanted all resources on my storage account to be available, so I left **Origin path** blank. This means that a request to `https://cdndocdemo.azureedge.net/publicblob/lorem.txt` results in a connection from my endpoint to `cdndocdemo.core.windows.net` that requests `/publicblob/lorem.txt`. Likewise, a request for `https://cdndocdemo.azureedge.net/donotcache/status.png` results in the endpoint requesting `/donotcache/status.png` from the origin.

But what if I don't want to use the CDN for every path on my origin? Say I only wanted to expose the `publicblob` path. If I enter `/publicblob` in my **Origin path** field, that will cause the endpoint to insert `/publicblob` before every request being made to the origin. This means that the request for `https://cdndocdemo.azureedge.net/publicblob/lorem.txt` will now actually take the request portion of the URL, `/publicblob/lorem.txt`, and append `/publicblob` to the beginning. This results in a request for `/publicblob/publicblob/lorem.txt` from the origin. If that path doesn't resolve to an actual file, the origin will return a 404 status. The correct URL to retrieve `lorem.txt` in this example would actually be `https://cdndocdemo.azureedge.net/lorem.txt`. Note that we don't include the `/publicblob` path at all, because the request portion of the URL is `/lorem.txt` and the endpoint adds `/publicblob`, resulting in `/publicblob/lorem.txt` being the request passed to the origin.

# Troubleshooting CDN file compression

1/24/2017 • 3 min to read • [Edit Online](#)

This article helps you troubleshoot issues with [CDN file compression](#).

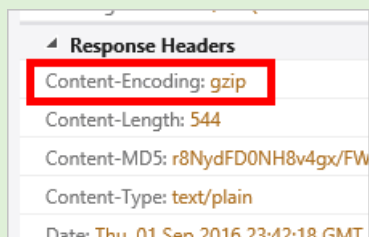
If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and the Stack Overflow forums](#). Alternatively, you can also file an Azure support incident. Go to the [Azure Support site](#) and click **Get Support**.

## Symptom

Compression for your endpoint is enabled, but files are being returned uncompressed.

### TIP

To check whether your files are being returned compressed, you need to use a tool like [Fiddler](#) or your browser's [developer tools](#). Check the HTTP response headers returned with your cached CDN content. If there is a header named `Content-Encoding` with a value of **gzip**, **bzip2**, or **deflate**, your content is compressed.



## Cause

There are several possible causes, including:

- The requested content is not eligible for compression.
- Compression is not enabled for the requested file type.
- The HTTP request did not include a header requesting a valid compression type.

## Troubleshooting steps

### TIP

As with deploying new endpoints, CDN configuration changes take some time to propagate through the network. Usually, changes are applied within 90 minutes. If this is the first time you've set up compression for your CDN endpoint, you should consider waiting 1-2 hours to be sure the compression settings have propagated to the POPs.

### Verify the request

First, we should do a quick sanity check on the request. You can use your browser's [developer tools](#) to view the requests being made.

- Verify the request is being sent to your endpoint URL, `<endpointname>.azureedge.net`, and not your origin.
- Verify the request contains an **Accept-Encoding** header, and the value for that header contains **gzip**, **deflate**, or **bzip2**.

#### NOTE

Azure CDN from Akamai profiles only support **gzip** encoding.

Request Headers
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US, en; q=0.5
Cache-Control: no-cache
Connection: Keep-Alive
Host: cdndocdemoarm.azureedge.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5

#### Verify compression settings (Standard CDN profile)



#### NOTE

This step only applies if your CDN profile is an **Azure CDN Standard from Verizon** or **Azure CDN Standard from Akamai** profile.

Navigate to your endpoint in the [Azure portal](#) and click the **Configure** button.

- Verify compression is enabled.
- Verify the MIME type for the content to be compressed is included in the list of compressed formats.

### Configure

 Save  Discard

#### Compression ⓘ

Off On

#### Formats to compress ⓘ

application/x-javascript	...
text/css	...
text/html	...
text/javascript	...
text/plain	...
<input type="text"/>	...

#### Verify compression settings (Premium CDN profile)

#### NOTE

This step only applies if your CDN profile is an **Azure CDN Premium from Verizon** profile.

Navigate to your endpoint in the [Azure portal](#) and click the **Manage** button. The supplemental portal will open.

Hover over the **HTTP Large** tab, then hover over the **Cache Settings** flyout. Click **Compression**.

- Verify compression is enabled.
- Verify the **File Types** list contains a comma-separated list (no spaces) of MIME types.
- Verify the MIME type for the content to be compressed is included in the list of compressed formats.

☐ Compression Disabled  
☒ Compression Enabled

File Types:  
text/plain,text/html,text/css,application/x-javascript,te  
xt/javascript

Last Updated: 4/20/2016 8:45:59 PM  
Expected Complete Time (1 Hour):4/20/2016 9:45:59 PM

### Verify the content is cached

#### NOTE

This step only applies if your CDN profile is an **Azure CDN from Verizon** profile (Standard or Premium).

Using your browser's developer tools, check the response headers to ensure the file is cached in the region where it is being requested.

- Check the **Server** response header. The header should have the format **Platform (POP/Server ID)**, as seen in the following example.
- Check the **X-Cache** response header. The header should read **HIT**.

Response Headers
Accept-Ranges: bytes
Content-Length: 2206
Content-MD5: QZF4+XrFhV9d5rJ4ZZdelQ==
Content-Type: text/plain
Date: Thu, 21 Apr 2016 01:24:45 GMT
Etag: 0x8D36982CD751E5D
Last-Modified: Thu, 21 Apr 2016 01:18:10 GMT
Server: ECAcc (dfw/565B)
X-Cache: HIT
x-ms-blob-type: BlockBlob

### Verify the file meets the size requirements

#### NOTE

This step only applies if your CDN profile is an **Azure CDN from Verizon** profile (Standard or Premium).

To be eligible for compression, a file must meet the following size requirements:

- Larger than 128 bytes.
- Smaller than 1 MB.

#### **Check the request at the origin server for a Via header**

The **Via** HTTP header indicates to the web server that the request is being passed by a proxy server. Microsoft IIS web servers by default do not compress responses when the request contains a **Via** header. To override this behavior, perform the following:

- **IIS 6:** [Set HcNoCompressionForProxies="FALSE" in the IIS Metabase properties](#)
- **IIS 7 and up:** [Set both \*\*noCompressionForHttp10\*\* and \*\*noCompressionForProxies\*\* to False in the server configuration](#)

# Azure CDN rules engine

1/25/2017 • 3 min to read • [Edit Online](#)

This topic lists detailed descriptions of the available match conditions and features for Azure Content Delivery Network (CDN) [Rules Engine](#).

The HTTP Rules Engine is designed to be the final authority on how specific types of requests are processed by the CDN.

## Common uses:

- Override or define a custom cache policy.
- Secure or deny requests for sensitive content.
- Redirect requests.
- Store custom log data.

## Terminology

A rule is defined through the use of **conditional expressions**, **matches**, and **features**. These elements are highlighted in the following illustration.

Name / Description:  Add Cancel

	URL Path Extension	Matches	Value	
<b>IF</b>			JPG JPEG GIF BMP PNG	Ignore Case <input checked="" type="checkbox"/>
Features			Force Internal Max-Age	Response Code 200 12 Seconds
			External Max-Age	12 Seconds
<b>ELSE IF</b>			FLV F4V MP4	Ignore Case <input checked="" type="checkbox"/>
Features			Force Internal Max-Age	Response Code 200 3 Days
			External Max-Age	3 Days
Matches				

**Legend**

- IF Conditional Expression
- JPG JPEG GIF BMP PNG Match
- Force Internal Max-Age Feature

## Syntax

The manner in which special characters will be treated varies according to how a match condition or feature handles text values. A match condition or feature may interpret text in one of the following ways:

1. **Literal Values**
2. **Wildcard Values**
3. **Regular Expressions**

### Literal Values

Text that is interpreted as a literal value will treat all special characters, with the exception of the % symbol, as a



part of the value that must be matched. In other words, a literal match condition set to `\*'\'` will only be satisfied when that exact value (i.e., `\*'\'`) is found.

A percentage symbol is used to indicate URL encoding (e.g., `%20`).

## Wildcard Values

Text that is interpreted as a wildcard value will assign additional meaning to special characters. The following table describes how the following set of characters will be interpreted.

CHARACTER	DESCRIPTION
\	A backslash is used to escape any of the characters specified in this table. A backslash must be specified directly before the special character that should be escaped. For example, the following syntax escapes an asterisk: <code>\*</code>
%	A percentage symbol is used to indicate URL encoding (e.g., <code>%20</code> ).
*	An asterisk is a wildcard that represents one or more characters.
Space	A space character indicates that a match condition may be satisfied by either of the specified values or patterns.
'value'	<p>A single quote does not have special meaning. However, a set of single quotes is used to indicate that a value should be treated as a literal value. It can be used in the following ways:</p> <ul style="list-style-type: none"><li>- It allows a match condition to be satisfied whenever the specified value matches any portion of the comparison value. For example, <code>'ma'</code> would match any of the following strings:  <code>/business/marathon/asset.htm</code> <code>map.gif</code> <code>/business/template.map</code></li><li>- It allows a special character to be specified as a literal character. For example, you may specify a literal space character by enclosing a space character within a set of single quotes (i.e., <code>' '</code> or <code>'sample value'</code>).</li><li>- It allows a blank value to be specified. Specify a blank value by specifying a set of single quotes (i.e., <code>''</code>).</li></ul> <p><b>Important:</b></p> <ul style="list-style-type: none"><li>- If the specified value does not contain a wildcard, then it will automatically be considered a literal value. This means that it is not necessary to specify a set of single quotes.</li><li>- If a backslash does not escape another character in this table, then it will be ignored when specified within a set of single quotes.</li><li>- Another way to specify a special character as a literal character is to escape it using a backslash (i.e., <code>\</code>).</li></ul>

## Regular Expressions

Regular expressions define a pattern that will be searched for within a text value. Regular expression notation defines specific meanings to a variety of symbols. The following table indicates how special characters are treated by match conditions and features that support regular expressions.

SPECIAL CHARACTER	DESCRIPTION
\	A backslash escapes the character the follows it. This causes that character to be treated as a literal value instead of taking on its regular expression meaning. For example, the following syntax escapes an asterisk: \*
%	<p>The meaning of a percentage symbol depends on its usage.</p> <p><code>{HTTPVariable}</code> : This syntax identifies an HTTP variable.</p> <p><code>{HTTPVariable%Pattern}</code> : This syntax uses a percentage symbol to identify an HTTP variable and as a delimiter.</p> <p><code>\%</code> : Escaping a percentage symbol allows it to be used as a literal value or to indicate URL encoding (e.g., <code>\%20</code> ).</p>
*	An asterisk allows the preceding character to be matched zero or more times.
Space	A space character is typically treated as a literal character.
'value'	Single quotes are treated as literal characters. A set of single quotes does not have special meaning.

## Next steps

- [Rules Engine Match Conditions](#)
- [Rules Engine Conditional Expressions](#)
- [Rules Engine Features](#)
- [Overriding default HTTP behavior using the rules engine](#)
- [Azure CDN Overview](#)

# Azure CDN rules engine conditional expressions

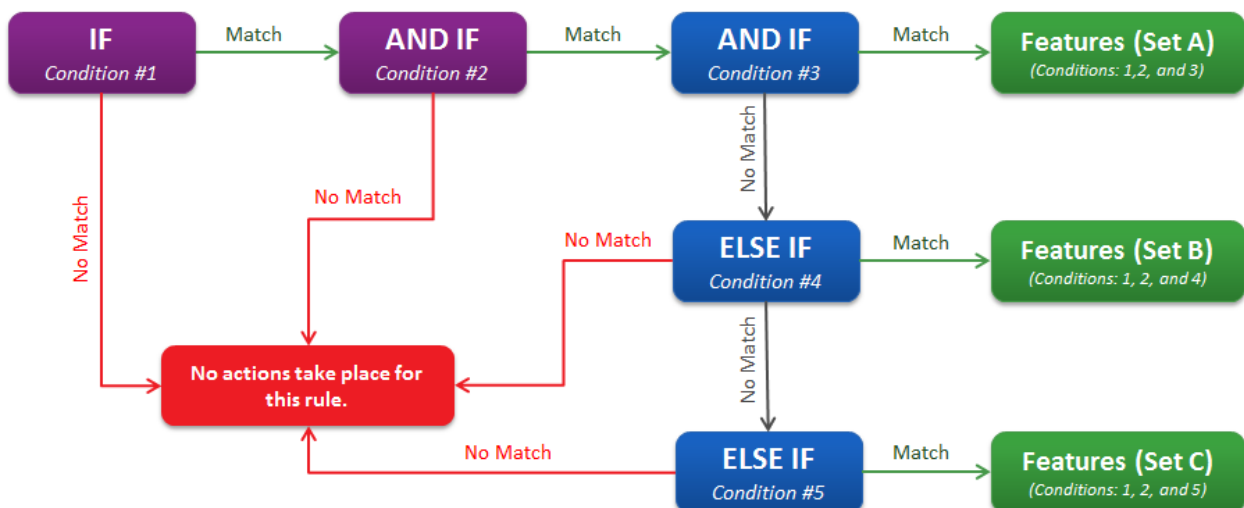
1/25/2017 • 1 min to read • [Edit Online](#)

This topic lists detailed descriptions of the Conditional Expressions for Azure Content Delivery Network (CDN) Rules Engine.

The first part of a rule is the Conditional Expression.

CONDITIONAL EXPRESSION	DESCRIPTION
IF	An IF expression is always a part of the first statement in a rule. Like all other conditional expressions, this IF statement must be associated with a match. If no additional conditional expressions are defined, then this match determines the criterion that must be met before a set of features may be applied to a request.
AND IF	An AND IF expression may only be added after the following types of conditional expressions: IF, AND IF. It indicates that there is another condition that must be met for the initial IF statement.
ELSE IF	An ELSE IF expression specifies an alternative condition that must be met before a set of features specific to this ELSE IF statement takes place. The presence of an ELSE IF statement indicates the end of the previous statement. The only conditional expression that may be placed after an ELSE IF statement is another ELSE IF statement. This means that an ELSE IF statement may only be used to specify a single additional condition that has to be met.

**Example:**



## TIP

A subsequent rule may override the actions specified by a previous rule. Example: A catch-all rule secures all requests via Token-Based Authentication. Another rule may be created directly below it to make an exception for certain types of requests.

**Next steps**

- [Azure CDN Overview](#)
- [Rules Engine Reference](#)
- [Rules Engine Match Conditions](#)
- [Rules Engine Features](#)
- [Overriding default HTTP behavior using the rules engine](#)

# Azure CDN rules engine match conditions

1/25/2017 • 4 min to read • [Edit Online](#)

This topic lists detailed descriptions of the available Match Conditions for Azure Content Delivery Network (CDN) [Rules Engine](#).

The second part of a rule is the match condition. A match condition identifies specific types of requests for which a set of features will be performed.

For example, it may be used to filter requests for content at a particular location, requests generated from a particular IP address or country, or by header information.

## Always

The Always match condition is designed to apply a default set of features to all requests.

## Device

The Device match condition identifies requests made from a mobile device based on its properties. Mobile device detection is achieved through [WURFL](#). WURFL capabilities and their CDN Rules Engine variables are listed below.

### NOTE

The variables below are supported in the **Modify Client Request Header** and **Modify Client Response Header** features.

CAPABILITY	VARIABLE	DESCRIPTION	SAMPLE VALUE(S)
Brand Name	%{wurfl_cap_brand_name}	A string that indicates the brand name of the device.	Samsung
Device OS	%{wurfl_cap_device_os}	A string that indicates the operating system installed on the device.	IOS
Device OS Version	%{wurfl_cap_device_os_version}	A string that indicates the version number of the OS installed on the device.	1.0.1
Dual Orientation	%{wurfl_cap_dual_orientation}	A Boolean that indicates whether the device supports dual orientation.	true
HTML Preferred DTD	%{wurfl_cap_html_preferred_dtd}	A string that indicates the mobile device's preferred document type definition (DTD) for HTML content.	none xhtml_basic html5
Image Inlining	%{wurfl_cap_image_inlining}	A Boolean that indicates whether the device supports Base64 encoded images.	false

CAPABILITY	VARIABLE	DESCRIPTION	SAMPLE VALUE(S)
Is Android	%{wurfl_vcap_is_android}	A Boolean that indicates whether the device uses the Android OS.	true
Is IOS	%{wurfl_vcap_is_ios}	A Boolean that indicates whether the device uses iOS.	false
Is Smart TV	%{wurfl_cap_is_smarttv}	A Boolean that indicates whether the device is a smart TV.	false
Is Smartphone	%{wurfl_vcap_is_smartphone}	A Boolean that indicates whether the device is a smartphone.	true
Is Tablet	%{wurfl_cap_is_tablet}	A Boolean that indicates whether the device is a tablet. This is an OS-independent description.	true
Is Wireless Device	%{wurfl_cap_is_wireless_device}	A Boolean that indicates whether the device is considered a wireless device.	true
Marketing Name	%{wurfl_cap_marketing_name}	A string that indicates the device's marketing name.	BlackBerry 8100 Pearl
Mobile Browser	%{wurfl_cap_mobile_browser}	A string that indicates the browser used to request content from the device.	Chrome
Mobile Browser Version	%{wurfl_cap_mobile_browser_version}	A string that indicates the version of the browser used to request content from the device.	31
Model Name	%{wurfl_cap_model_name}	A string that indicates the device's model name.	s3
Progressive Download	%{wurfl_cap_progressive_download}	A Boolean that indicates whether the device supports the playback of audio/video while it is still being downloaded.	true
Release Date	%{wurfl_cap_release_date}	A string that indicates the year and month on which the device was added to the WURFL database.  Format: <code>yyyy_mm</code>	2013_december
Resolution Height	%{wurfl_cap_resolution_height}	An integer that indicates the device's height in pixels.	768

CAPABILITY	VARIABLE	DESCRIPTION	SAMPLE VALUE(S)
Resolution Width	% {wurfl_cap_resolution_width}	An integer that indicates the device's width in pixels.	1024

## Location

These match conditions are designed to identify requests based on the requester's location.

NAME	PURPOSE
AS Number	Identifies requests that originate from a particular network.
Country	Identifies requests that originate from the specified countries.

## Origin

These match conditions are designed to identify requests that point to CDN storage or a customer origin server.

NAME	PURPOSE
CDN Origin	Identifies requests for content stored on CDN storage.
Customer Origin	Identifies requests for content stored on a specific customer origin server.

## Request

These match conditions are designed to identify requests based on their properties.

NAME	PURPOSE
Client IP Address	Identifies requests that originate from a particular IP address.
Cookie Parameter	Checks the cookies associated with each request for the specified value.
Cookie Parameter Regex	Checks the cookies associated with each request for the specified regular expression.
Edge Cname	Identifies requests that point to a specific edge CNAME.
Referring Domain	Identifies requests that were referred from the specified hostname(s).
Request Header Literal	Identifies requests that contain the specified header set to a specified value(s).
Request Header Regex	Identifies requests that contain the specified header set to a value that matches the specified regular expression.
Request Header Wildcard	Identifies requests that contain the specified header set to a value that matches the specified pattern.

NAME	PURPOSE
Request Method	Identifies requests by their HTTP method.
Request Scheme	Identifies requests by their HTTP protocol.

## URL

These match conditions are designed to identify requests based on their URLs.

NAME	PURPOSE
URL Path Directory	Identifies requests by their relative path.
URL Path Extension	Identifies requests by their filename extension.
URL Path Filename	Identifies requests by their filename.
URL Path Literal	Compares a request's relative path to the specified value.
URL Path Regex	Compares a request's relative path to the specified regular expression.
URL Path Wildcard	Compares a request's relative path to the specified pattern.
URL Query Literal	Compares a request's query string to the specified value.
URL Query Parameter	Identifies requests that contain the specified query string parameter set to a value that matches a specified pattern.
URL Query Regex	Identifies requests that contain the specified query string parameter set to a value that matches a specified regular expression.
URL Query Wildcard	Compares the specified value(s) against the request's query string.

## Next steps

- [Azure CDN Overview](#)
- [Rules Engine Reference](#)
- [Rules Engine Conditional Expressions](#)
- [Rules Engine Features](#)
- [Overriding default HTTP behavior using the rules engine](#)



# Azure CDN rules engine features

1/25/2017 • 44 min to read • [Edit Online](#)

This topic lists detailed descriptions of the available features for Azure Content Delivery Network (CDN) [Rules Engine](#).

The third part of a rule is the feature. A feature defines the type of action that will be applied to the type of request identified by a set of match conditions.

## Access

These features are designed to control access to content.

NAME	PURPOSE
Deny Access	Determines whether all requests are rejected with a 403 Forbidden response.
Token Auth	Determines whether Token-Based Authentication will be applied to a request.
Token Auth Denial Code	Determines the type of response that will be returned to a user when a request is denied due to Token-Based Authentication.
Token Auth Ignore URL Case	Determines whether URL comparisons made by Token-Based Authentication will be case-sensitive.
Token Auth Parameter	Determines whether the Token-Based Authentication query string parameter should be renamed.

### Deny Access

**Purpose:** Determines whether all requests are rejected with a 403 Forbidden response.

VALUE	RESULT
Enabled	Causes all requests that satisfy the matching criteria to be rejected with a 403 Forbidden response.
Disabled	Restores the default behavior. The default behavior is to allow the origin server to determine the type of response that will be returned.

**Default Behavior:** Disabled

#### TIP

One possible use for this feature is to associate it with a Request Header match condition to block access to HTTP referrers that are using inline links to your content.

### Token Auth

**Purpose:** Determines whether Token-Based Authentication will be applied to a request.

If Token-Based Authentication is enabled, then only requests that provide an encrypted token and comply to the requirements specified by that token will be honored.

The encryption key that will be used to encrypt and decrypt token values is determined by the Primary Key and the Backup Key options on the Token Auth page. Keep in mind that encryption keys are platform-specific.

VALUE	RESULT
Enabled	Protects the requested content with Token-Based Authentication. Only requests from clients that provide a valid token and meet its requirements will be honored. FTP transactions are excluded from Token-Based Authentication.
Disabled	Restores the default behavior. The default behavior is to allow your Token-Based Authentication configuration to determine whether a request will be secured.

**Default Behavior:** Disabled.

### Token Auth Denial Code

**Purpose:** Determines the type of response that will be returned to a user when a request is denied due to Token-Based Authentication.

The available response codes are listed below.

RESPONSE CODE	RESPONSE NAME	DESCRIPTION
301	Moved Permanently	This status code redirects unauthorized users to the URL specified in the Location header.
302	Found	This status code redirects unauthorized users to the URL specified in the Location header. This status code is the industry standard method of performing a redirect.
307	Temporary Redirect	This status code redirects unauthorized users to the URL specified in the Location header.
401	Unauthorized	Combining this status code with the WWW-Authenticate response header allows you to prompt a user for authentication.
403	Forbidden	This is the standard 403 Forbidden status message that an unauthorized user will see when trying to access protected content.
404	File Not Found	This status code indicates that the HTTP client was able to communicate with the server, but the requested content was not found.

### URL Redirection

This feature supports URL redirection to a user-defined URL when it is configured to return a 3xx status code. This user-defined URL can be specified by performing the following steps:

1. Select a 3xx response code for the Token Auth Denial Code feature.
2. Select "Location" from the Optional Header Name option.
3. Set the Optional Header Value option to the desired URL.

If a URL is not defined for a 3xx status code, then the standard response page for a 3xx status code will be returned to the user.

URL redirection is only applicable for 3xx response codes.

The Optional Header Value option supports alphanumeric characters, quotation marks, and spaces.

#### Authentication

This feature supports the capability to include the WWW-Authenticate header when responding to an unauthorized request for content protected by Token-Based Authentication. If the WWW-Authenticate header has been set to "basic" in your configuration, then the unauthorized user will be prompted for account credentials.

The above configuration can be achieved by performing the following steps:

1. Select "401" as the response code for the Token Auth Denial Code feature.
2. Select "WWW-Authenticate" from the Optional Header Name option.
3. Set the Optional Header Value option to "basic."

The WWW-Authenticate header is only applicable for 401 response codes.

#### Token Auth Ignore URL Case

**Purpose:** Determines whether URL comparisons made by Token-Based Authentication will be case-sensitive.

The parameters affected by this feature are:

- ec\_url\_allow
- ec\_ref\_allow
- ec\_ref\_deny

Valid values are:

VALUE	RESULT
Enabled	Causes our edge server to ignore case when comparing URLs for Token-Based Authentication parameters.
Disabled	Restores the default behavior. The default behavior is for URL comparisons for Token Authentication to be case-sensitive.

**Default Behavior:** Disabled.

#### Token Auth Parameter

**Purpose:** Determines whether the Token-Based Authentication query string parameter should be renamed.

Key information:

- The Value option defines the query string parameter name through which a token may be specified.
- The Value option cannot be set to "ec\_token."
- Make sure that the name defined in the Value option only
- contains valid URL characters.

VALUE	RESULT
Enabled	The Value option defines the query string parameter name through which tokens should be defined.
Disabled	A token may be specified as an undefined query string parameter in the request URL.

**Default Behavior:** Disabled. A token may be specified as an undefined query string parameter in the request URL.

## Caching

These features are designed to customize when and how content is cached.

NAME	PURPOSE
Bandwidth Parameters	Determines whether bandwidth throttling parameters (i.e., ec_rate and ec_prebuf) will be active.
Bandwidth Throttling	Throttles the bandwidth for the response provided by our edge servers.
Bypass Cache	Determines whether the request can leverage our caching technology.
Cache-Control Header Treatment	Controls the generation of Cache-Control headers by the edge server when External Max-Age feature is active.
Cache-Key Query String	Determines whether the cache-key will include or exclude query string parameters associated with a request.
Cache-Key Rewrite	Rewrites the cache-key associated with a request.
Complete Cache Fill	Determines what happens when a request results in a partial cache miss on an edge server.
Compress File Types	Defines the file formats that will be compressed on the server.
Default Internal Max-Age	Determines the default max-age interval for edge server to origin server cache revalidation.
Expires Header Treatment	Controls the generation of Expires headers by an edge server when the External Max-Age feature is active.
External Max-Age	Determines the max-age interval for browser to edge server cache revalidation.
Force Internal Max-Age	Determines the max-age interval for edge server to origin server cache revalidation.
H.264 Support (HTTP Progressive Download)	Determines the types of H.264 file formats that may be used to stream content.

NAME	PURPOSE
Honor No-Cache Request	Determines whether an HTTP client's no-cache requests will be forwarded to the origin server.
Ignore Origin No-Cache	Determines whether our CDN will ignore certain directives served from an origin server.
Ignore Unsatisfiable Ranges	Determines the response that will be returned to clients when a request generates a 416 Requested Range Not Satisfiable status code.
Internal Max-Stale	Controls how long past the normal expiration time a cached asset may be served from an edge server when the edge server is unable to revalidate the cached asset with the origin server.
Partial Cache Sharing	Determines whether a request can generate partially cached content.
Prevalidate Cached Content	Determines whether cached content will be eligible for early revalidation before its TTL expires.
Refresh Zero-Byte Cache Files	Determines how an HTTP client's request for a 0-byte cache asset is handled by our edge servers.
Set Cacheable Status Codes	Defines the set of status codes that can result in cached content.
Stale Content Delivery on Error	Determines whether expired cached content will be delivered when an error occurs during cache revalidation or when retrieving the requested content from the customer origin server.
Stale While Revalidate	Improves performance by allowing our edge servers to serve stale client to the requester while revalidation takes place.
Comment	The Comment feature allows a note to be added within a rule.

## Bandwidth Parameters

**Purpose:** Determines whether bandwidth throttling parameters (i.e., `ec_rate` and `ec_prebuf`) will be active.

Bandwidth throttling parameters determine whether the data transfer rate for a client's request will be limited to a custom rate.

VALUE	RESULT
Enabled	Allows our edge servers to honor bandwidth throttling requests.
Disabled	Causes our edge servers to ignore bandwidth throttling parameters. The requested content will be served normally (i.e., without bandwidth throttling).

**Default Behavior:** Enabled.

## Bandwidth Throttling

**Purpose:** Throttles the bandwidth for the response provided by our edge servers.

Both of the following options must be defined to properly set up bandwidth throttling.

OPTION	DESCRIPTION
Kbytes per second	Set this option to the maximum bandwidth (Kb per second) that may be used to deliver the response.
Prebuf seconds	Set this option to the number of seconds that our edge servers will wait until throttling bandwidth. The purpose of this time period of unrestricted bandwidth is to prevent a media player from experiencing stuttering or buffering issues due to bandwidth throttling.

**Default Behavior:** Disabled.

### Bypass Cache

**Purpose:** Determines whether the request can leverage our caching technology.

VALUE	RESULT
Enabled	Causes all requests to fall through to the origin server, even if the content was previously cached on edge servers.
Disabled	Causes edge servers to cache assets according to the cache policy defined in its response headers.

**Default Behavior:**

- **HTTP Large:** Disabled

### Cache Control Header Treatment

**Purpose:** Controls the generation of Cache-Control headers by the edge server when External Max-Age Feature is active.

The easiest way to achieve this type of configuration is to place the External Max-Age and the Cache-Control Header Treatment features in the same statement.

VALUE	RESULT
Overwrite	Ensures that the following actions will take place: <ul style="list-style-type: none"><li>- Overwrites the Cache-Control header generated by the origin server.</li><li>- Adds the Cache-Control header produced by the External Max-Age feature to the response.</li></ul>
Pass Through	Ensures that the Cache-Control header produced by the External Max-Age feature is never added to the response. If the origin server produces a Cache-Control header, it will pass through to the end-user. If the origin server does not produce a Cache-Control header, then this option may cause the response header to not contain a Cache-Control header.

VALUE	RESULT
Add if Missing	If a Cache-Control header was not received from the origin server, then this option adds the Cache-Control header produced by the External Max-Age feature. This option is useful for ensuring that all assets will be assigned a Cache-Control header.
Remove	This option ensures that a Cache-Control header is not included with the header response. If a Cache-Control header has already been assigned, then it will be stripped from the header response.

**Default Behavior:** Overwrite.

### Cache-Key Query String

**Purpose:** Determines whether the cache-key will include or exclude query string parameters associated with a request.

Key information:

- Specify one or more query string parameter name(s). Each parameter name should be delimited with a single space.
- This feature determines whether query string parameters will be included or excluded from the cache-key. Additional information is provided for each option below.

TYPE	DESCRIPTION
Include	Indicates that each specified parameter should be included in the cache-key. A unique cache-key will be generated for each request that contains a unique value for a query string parameter defined in this feature.
Include All	Indicates that a unique cache-key will be created for each request to an asset that includes a unique query string. This type of configuration is not typically recommended since it may lead to a small percentage of cache hits. This will increase the load on the origin server, since it will have to serve more requests. This configuration duplicates the caching behavior known as "unique-cache" on the Query-String Caching page.
Exclude	Indicates that only the specified parameter(s) will be excluded from the cache-key. All other query string parameters will be included in the cache-key.
Exclude All	Indicates that all query string parameters will be excluded from the cache-key. This configuration duplicates the default caching behavior, which is known as "standard-cache" on the Query-String Caching page.

The power of HTTP Rules Engine allows you to customize the manner in which query string caching is implemented. For example, you can specify that query string caching only be performed on certain locations or file types.

If you would like to duplicate the query string caching behavior known as "no-cache" on the Query-String Caching page, then you will need to create a rule that contains a URL Query Wildcard match condition and a Bypass Cache feature. The URL Query Wildcard match condition should be set to an asterisk (\*).

## Sample Scenarios

Sample usage for this feature is provided below. A sample request and the default cache-key are provided below.

- **Sample request:** http://wpc.0001.<Domain>/800001/Origin/folder/asset.htm?sessionid=1234&language=EN&userid=01
- **Default cache-key:** /800001/Origin/folder/asset.htm

### Include

Sample configuration:

- **Type:** Include
- **Parameter(s):** language

This type of configuration would generate the following query string parameter cache-key:

```
/800001/Origin/folder/asset.htm?language=EN
```

### Include All

Sample configuration:

- **Type:** Include All

This type of configuration would generate the following query string parameter cache-key:

```
/800001/Origin/folder/asset.htm?sessionid=1234&language=EN&userid=01
```

### Exclude

Sample configuration:

- **Type:** Exclude
- **Parameter(s):** sessionid userid

This type of configuration would generate the following query string parameter cache-key:

```
/800001/Origin/folder/asset.htm?language=EN
```

### Exclude All

Sample configuration:

- **Type:** Exclude All

This type of configuration would generate the following query string parameter cache-key:

```
/800001/Origin/folder/asset.htm
```

## Cache-Key Rewrite

**Purpose:** Rewrites the cache-key associated with a request.

A cache-key is the relative path that identifies an asset for the purposes of caching. In other words, our servers will check for a cached version of an asset according to its path as defined by its cache-key.

Configure this feature by defining both of the following options:

OPTION	DESCRIPTION
--------	-------------



OPTION	DESCRIPTION
Original Path	Define the relative path to the types of requests whose cache-key will be rewritten. A relative path can be defined by selecting a base origin path and then defining a regular expression pattern.
New Path	Define the relative path for the new cache-key. A relative path can be defined by selecting a base origin path and then defining a regular expression pattern. This relative path can be dynamically constructed through the use of HTTP variables

**Default Behavior:** A request's cache-key is determined by the request URI.

### Complete Cache Fill

**Purpose:** Determines what happens when a request results in a partial cache miss on an edge server.

A partial cache miss describes the cache status for an asset that was not completely downloaded to an edge server. If an asset is only partially cached on an edge server, then the next request for that asset will be forwarded again to the origin server.

A partial cache miss typically occurs after a user aborts a download or for assets that are solely requested using HTTP range requests. This feature is most useful for large assets where users will not typically download them from start to finish (e.g., videos). As a result, this feature is enabled by default on the HTTP Large platform. It is disabled on all other platforms.

It is recommended to leave the default configuration for the HTTP Large platform, since it will reduce the load on your customer origin server and increase the speed at which your customers download your content.

Due to the manner in which cache settings are tracked, this feature cannot be associated with the following Match conditions: Edge Cname, Request Header Literal, Request Header Wildcard, URL Query Literal, and URL Query Wildcard.

VALUE	RESULT
Enabled	Restores the default behavior. The default behavior is to force the edge server to initiate a background fetch of the asset from the origin server. After which, the asset will be in the edge server's local cache.
Disabled	Prevents an edge server from performing a background fetch for the asset. This means that the next request for that asset from that region will cause an edge server to request it from the customer origin server.

**Default Behavior:** Enabled.

### Compress File Types

**Purpose:** Defines the file formats that will be compressed on the server.

A file format can be specified using its Internet media type (i.e., Content-Type). Internet media type is platform-independent metadata that allows our servers to identify the file format of a particular asset. A list of common Internet media types is provided below.

INTERNET MEDIA TYPE	DESCRIPTION
text/plain	Plain text files

INTERNET MEDIA TYPE	DESCRIPTION
text/html	HTML files
text/css	Cascading Style Sheets (CSS)
application/x-javascript	Javascript
application/javascript	Javascript

Key information:

- Specify multiple Internet media types by delimiting each one with a single space.
- This feature will only compress assets whose size is less than 1 MB. Larger assets will not be compressed by our servers.
- Certain types of content, such as images, video, and audio media assets (e.g., JPG, MP3, MP4, etc.), are already compressed. Additional compression on these types of assets will not significantly diminish file size. Therefore, it is recommended that you do not enable compression on these types of assets.
- Wildcard characters, such as asterisks, are not supported.
- Before adding this feature to a rule, make sure to set the Compression Disabled option on the Compression page for the platform to which this rule will be applied.

### Default Internal Max-Age

**Purpose:** Determines the default max-age interval for edge server to origin server cache revalidation. In other words, the amount of time that will pass before an edge server will check whether a cached asset matches the asset stored on the origin server.

Key information:

- This action will only take place for responses from an origin server that did not assign a max-age indication in the Cache-Control or Expires header.
- This action will not take place for assets that are not deemed cacheable.
- This action does not affect browser to edge server cache revalidations. These types of revalidations are determined by the Cache-Control or Expires headers sent to the browser, which can be customized with the External Max-Age feature.
- The results of this action do not have an observable effect on the response headers and the content returned from edge servers for your content, but it may have an effect on the amount of revalidation traffic sent from edge servers to your origin server.
- Configure this feature by:
  - Selecting the status code for which a default internal max-age can be applied.
  - Specifying an integer value and then selecting the desired time unit (i.e., seconds, minutes, hours, etc.). This value defines the default internal max-age interval.
- Setting the time unit to "Off" will assign a default internal max-age interval of 7 days for requests that have not been assigned a max-age indication in their Cache-Control or Expires header.
- Due to the manner in which cache settings are tracked, this feature cannot be associated with the following match conditions:
  - Edge
  - Cname
  - Request Header Literal
  - Request Header Wildcard

- Request Method
- URL Query Literal
- URL Query Wildcard

**Default Value:** 7 days

### Expires Header Treatment

**Purpose:** Controls the generation of Expires headers by an edge server when the External Max-Age feature is active.

The easiest way to achieve this type of configuration is to place the External Max-Age and the Expires Header Treatment features in the same statement.

VALUE	RESULT
Overwrite	Ensures that the following actions will take place: - Overwrites the Expires header generated by the origin server. - Adds the Expires header produced by the External Max-Age feature to the response.
Pass Through	Ensures that the Expires header produced by the External Max-Age feature is never added to the response. If the origin server produces an Expires header, it will pass through to the end-user. If the origin server does not produce an Expires header, then this option may cause the response header to not contain an Expires header.
Add if Missing	If an Expires header was not received from the origin server, then this option adds the Expires header produced by the External Max-Age feature. This option is useful for ensuring that all assets will be assigned an Expires header.
Remove	Ensures that an Expires header is not included with the header response. If an Expires header has already been assigned, then it will be stripped from the header response.

**Default Behavior:** Overwrite

### External Max-Age

**Purpose:** Determines the max-age interval for browser to edge server cache revalidation. In other words, the amount of time that will pass before a browser can check for a new version of an asset from an edge server.

Enabling this feature will generate Cache-Control:max-age and Expires headers from our edge servers and send them to the HTTP client. By default, these headers will overwrite those created by the origin server. However, the Cache-Control Header Treatment and the Expires Header Treatment features may be used to alter this behavior.

Key information:

- This action does not affect edge server to origin server cache revalidations. These types of revalidations are determined by the Cache-Control/Expires headers received from the origin server, and can be customized with the Default Internal Max-Age and the Force Internal Max-Age features.
- Configure this feature by specifying an integer value and selecting the desired time unit (i.e., seconds, minutes, hours, etc.).
- Setting this feature to a negative value causes our edge servers to send a Cache-Control:no-cache and an Expires time that is set in the past with each response to the browser. Although an HTTP client will not cache

the response, this setting will not affect our edge servers' ability to cache the response from the origin server.

- Setting the time unit to "Off" will disable this feature. The Cache-Control/Expires headers cached with the response of the origin server will pass through to the browser.

**Default Behavior:** Off

### Force Internal Max-Age

**Purpose:** Determines the max-age interval for edge server to origin server cache revalidation. In other words, the amount of time that will pass before an edge server can check whether a cached asset matches the asset stored on the origin server.

Key information:

- This feature will override the max-age interval defined in Cache-Control or Expires headers generated from an origin server.
- This feature does not affect browser to edge server cache revalidations. These types of revalidations are determined by the Cache-Control or Expires headers sent to the browser.
- This feature does not have an observable effect on the response delivered by an edge server to the requester. However, it may have an effect on the amount of revalidation traffic sent from our edge servers to the origin server.
- Configure this feature by:
  - Selecting the status code for which an internal max-age will be applied.
  - Specifying an integer value and selecting the desired time unit (i.e., seconds, minutes, hours, etc.). This value defines the request's max-age interval.
- Setting the time unit to "Off" disables this feature. An internal max-age interval will not be assigned to requested assets. If the original header does not contain caching instructions, then the asset will be cached according to the active setting in the Default Internal Max-Age feature.
- Due to the manner in which cache settings are tracked, this feature cannot be associated with the following match conditions:
  - Edge
  - Cname
  - Request Header Literal
  - Request Header Wildcard
  - Request Method
  - URL Query Literal
  - URL Query Wildcard

**Default Behavior:** Off

### H.264 Support (HTTP Progressive Download)

**Purpose:** Determines the types of H.264 file formats that may be used to stream content.

Key information:

- Define a space-delimited set of allowed H.264 filename extensions in the File Extensions option. The File Extensions option will override the default behavior. Maintain MP4 and F4V support by including those filename extensions when setting this option.
- Make sure to include a period when specifying each filename extension (e.g., .mp4 .f4v).

**Default Behavior:** HTTP Progressive Download supports MP4 and F4V media by default.

### Honor no-cache request

**Purpose:** Determines whether an HTTP client's no-cache requests will be forwarded to the origin server.

A no-cache request occurs when the HTTP client sends a Cache-Control:no-cache and/or Pragma:no-cache header in the HTTP request.

VALUE	RESULT
Enabled	Allows an HTTP client's no-cache requests to be forwarded to the origin server, and the origin server will return the response headers and the body through the edge server back to the HTTP client.
Disabled	Restores the default behavior. The default behavior is to prevent no-cache requests from being forwarded to the origin server.

For all production traffic, it is highly recommended to leave this feature in its default disabled state. Otherwise, origin servers will not be shielded from end-users who may inadvertently trigger many no-cache requests when refreshing web pages, or from the many popular media players that are coded to send a no-cache header with every video request. Nevertheless, this feature can be useful to apply to certain non-production staging or testing directories, in order to allow fresh content to be pulled on-demand from the origin server.

The cache status that will be reported for a request that is allowed to be forwarded to an origin server due to this feature is TCP\_Client\_Refresh\_Miss. The Cache Statuses report, which is available in the Core reporting module, provides statistical information by cache status. This allows you to track the number and percentage of requests that are being forwarded to an origin server due to this feature.

**Default Behavior:** Disabled.

### Ignore Origin no-cache

**Purpose:** Determines whether our CDN will ignore the following directives served from an origin server:

- Cache-Control: private
- Cache-Control: no-store
- Cache-Control: no-cache
- Pragma: no-cache

Key information:

- Configure this feature by defining a space-delimited list of status codes for which the above directives will be ignored.
- The set of valid status codes for this feature are: 200, 203, 300, 301, 302, 305, 307, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 500, 501, 502, 503, 504, and 505.
- Disable this feature by setting it to a blank value.
- Due to the manner in which cache settings are tracked, this feature cannot be associated with the following match conditions:
  - Edge
  - Cname
  - Request Header Literal
  - Request Header Wildcard
  - Request Method
  - URL Query Literal
  - URL Query Wildcard

**Default Behavior:** The default behavior is to honor the above directives.

### Ignore Unsatisfiable Ranges

**Purpose:** Determines the response that will be returned to clients when a request generates a 416 Requested Range Not Satisfiable status code.

By default, this status code is returned when the specified byte-range request cannot be satisfied by an edge server and an If-Range request header field was not specified.

VALUE	RESULT
Enabled	Prevents our edge servers from responding to an invalid byte-range request with a 416 Requested Range Not Satisfiable status code. Instead our servers will deliver the requested asset and return a 200 OK to the client.
Disabled	Restores the default behavior. The default behavior is to honor the 416 Requested Range Not Satisfiable status code.

**Default Behavior:** Disabled.

### Internal Max-Stale

**Purpose:** Controls how long past the normal expiration time a cached asset may be served from an edge server when the edge server is unable to revalidate the cached asset with the origin server.

Normally, when an asset's max-age time expires, the edge server will send a revalidation request to the origin server. The origin server will then respond with either a 304 Not Modified to give the edge server a fresh lease on the cached asset, or else with 200 OK to provide the edge server with an updated version of the cached asset.

If the edge server is unable to establish a connection with the origin server while attempting such a revalidation, then this Internal Max-Stale feature controls whether, and for how long, the edge server may continue to serve the now-stale asset.

Note that this time interval starts when the asset's max-age expires, not when the failed revalidation occurs. Therefore, the maximum period during which an asset can be served without successful revalidation is the amount of time specified by the combination of max-age plus max-stale. For example, if an asset was cached at 9:00 with a max-age of 30 minutes and a max-stale of 15 minutes, then a failed revalidation attempt at 9:44 would result in an end-user receiving the stale cached asset, while a failed revalidation attempt at 9:46 would result in the end user receiving a 504 Gateway Timeout.

Any value configured for this feature is superseded by Cache-Control:must-revalidate or Cache-Control:proxy-revalidate headers received from the origin server. If either of those headers is received from the origin server when an asset is initially cached, then the edge server will not serve a stale cached asset. In such a case, if the edge server is unable to revalidate with the origin when the asset's max-age interval has expired, then the edge server will return a 504 Gateway Timeout.

Key information:

- Configure this feature by:
  - Selecting the status code for which a max-stale will be applied.
  - Specifying an integer value and then selecting the desired time unit (i.e., seconds, minutes, hours, etc.). This value defines the internal max-stale that will be applied.
- Setting the time unit to "Off" will disable this feature. A cached asset will not be served beyond its normal expiration time.
- Due to the manner in which cache settings are tracked, this feature cannot be associated with the following match conditions:
  - Edge
  - Cname

- Request Header Literal
- Request Header Wildcard
- Request Method
- URL Query Literal
- URL Query Wildcard

**Default Behavior:** 2 minutes

### Partial Cache Sharing

**Purpose:** Determines whether a request can generate partially cached content.

This partial cache may then be used to fulfill new requests for that content until the requested content is fully cached.

VALUE	RESULT
Enabled	Requests can generate partially cached content.
Disabled	Requests can only generate a fully cached version of the requested content.

**Default Behavior:** Disabled.

### Prevalidate Cached Content

**Purpose:** Determines whether cached content will be eligible for early revalidation before its TTL expires.

Define the amount of time prior to the expiration of the requested content's TTL during which it will be eligible for early revalidation.

Key information:

- Selecting "Off" as the time unit requires revalidation to take place after the cached content's TTL has expired. Time should not be specified and it will be ignored.

**Default Behavior:** Off. Revalidation may only take place after the cached content's TTL has expired.

### Refresh Zero Byte Cache Files

**Purpose:** Determines how an HTTP client's request for a 0-byte cache asset is handled by our edge servers.

Valid values are:

VALUE	RESULT
Enabled	Causes our edge server to re-fetch the asset from the origin server.
Disabled	Restores the default behavior. The default behavior is to serve up valid cache assets upon request.

This feature is not required for correct caching and content delivery, but may be useful as a workaround. For example, dynamic content generators on origin servers can inadvertently result in 0-byte responses being sent to the edge servers. These types of responses are typically cached by our edge servers. If you know that a 0-byte response is never a valid response

for such content, then this feature can prevent these types of assets from being served to your clients.

**Default Behavior:** Disabled.

## Set Cacheable Status Codes

**Purpose:** Defines the set of status codes that can result in cached content.

By default, caching is only enabled for 200 OK responses.

Define a space-delimited set of the desired status codes.

Key information:

- Please also enable the Ignore Origin No-Cache feature. If that feature is not enabled, then non-200 OK responses may not be cached.
- The set of valid status codes for this feature are: 203, 300, 301, 302, 305, 307, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 500, 501, 502, 503, 504, and 505.
- This feature cannot be used to disable caching for responses that generate a 200 OK status code.

**Default Behavior:** Caching is only enabled for responses that generate a 200 OK status code.

## Stale Content Delivery on Error

**Purpose:**

Determines whether expired cached content will be delivered when an error occurs during cache revalidation or when retrieving the requested content from the customer origin server.

VALUE	RESULT
Enabled	Stale content will be served to the requester when an error occurs during a connection to an origin server.
Disabled	The origin server's error will be forwarded to the requester.

**Default Behavior:** Disabled

## Stale While Revalidate

**Purpose:** Improves performance by allowing our edge servers to serve stale content to the requester while revalidation takes place.

Key information:

- The behavior of this feature varies according to the selected time unit.
  - **Time Unit:** Specify a length of time and select a time unit (e.g., Seconds, Minutes, Hours, etc.) to allow stale content delivery. This type of setup allows the CDN to extend the length of time that it may deliver content before requiring validation according to the following formula: **TTL + Stale While Revalidate Time**
  - **Off:** Select "Off" to require revalidation before a request for stale content may be served.
    - Do not specify a length of time since it is inapplicable and will be ignored.

**Default Behavior:** Off. Revalidation must take place before the requested content can be served.

## Comment

**Purpose:** Allows a note to be added within a rule.

One use for this feature is to provide additional information on the general purpose of a rule or why a particular match condition or feature was added to the rule.

Key information:

- A maximum of 150 characters may be specified.



- Make sure to only use alphanumeric characters.
- This feature does not affect the behavior of the rule. It is merely meant to provide an area where you can provide information for future reference or that may help when troubleshooting the rule.

## Headers

These features are designed to add, modify, or delete headers from the request or response.

NAME	PURPOSE
Age Response Header	Determines whether an Age response header will be included in the response sent to the requester.
Debug Cache Response Headers	Determines whether a response may include the X-EC-Debug response header which provides information on the cache policy for the requested asset.
Modify Client Request Header	Overwrites, appends, or deletes a header from a request.
Modify Client Response Header	Overwrites, appends, or deletes a header from a response.
Set Client IP Custom Header	Allows the IP address of the requesting client to be added to the request as a custom request header.

### Age Response Header

**Purpose:** Determines whether an Age response header will be included in the response sent to the requester.

VALUE	RESULT
Enabled	The Age response header will be included in the response sent to the requester.
Disabled	The Age response header will be excluded from the response sent to the requester.

**Default Behavior:** Disabled.

### Debug Cache Response Headers

**Purpose:** Determines whether a response may include the X-EC-Debug response header which provides information on the cache policy for the requested asset.

Debug cache response headers will be included in the response when both of the following are true:

- The Debug Cache Response Headers Feature has been enabled on the desired request.
- The above request defines the set of debug cache response headers that will be included in the response.

Debug cache response headers may be requested by including the following header and the desired directives in the request:

X-EC-Debug: *Directive1,Directive2,DirectiveN*

### Example:

X-EC-Debug: x-ec-cache,x-ec-check-cacheable,x-ec-cache-key,x-ec-cache-state

VALUE	RESULT
Enabled	Requests for debug cache response headers will return a response that includes the X-EC-Debug header.
Disabled	The X-EC-Debug response header will be excluded from the response.

**Default Behavior:** Disabled.

## Modify Client Response Header

**Purpose:** Each request contains a set of [request headers]() that describe it. This feature can either:

- Append or overwrite the value assigned to a request header. If the specified request header does not exist, then this feature will add it to the request.
- Delete a request header from the request.

Requests that are forwarded to an origin server will reflect the changes made by this feature.

One of the following actions can be performed on a request header:

OPTION	DESCRIPTION	EXAMPLE
Append	The specified value will be added to end of the existing request header value.	<b>Request header value (Client):</b> Value1 <b>Request header value (HTTP Rules Engine):</b> Value2 <b>New request header value:</b> Value1Value2
Overwrite	The request header value will be set to the specified value.	<b>Request header value (Client):</b> Value1 <b>Request header value (HTTP Rules Engine):</b> Value2 <b>New request header value:</b> Value2
Delete	Deletes the specified request header.	<b>Request header value (Client):</b> Value1 <b>Modify Client Request Header configuration:</b> Delete the request header in question. <b>Result:</b> The specified request header will not be forwarded to the origin server.

Key information:

- Make sure that the value specified in the Name option is an exact match for the desired request header.
- Case is not taken into account for the purpose of identifying a header. For example, any of the following variations of the Cache-Control header name can be used to identify it:
  - cache-control
  - CACHE-CONTROL
  - cachE-Control
- Make sure to only use alphanumeric characters, dashes, or underscores when specifying a header name.
- Deleting a header will prevent it from being forwarded to an origin server by our edge servers.
- The following headers are reserved and cannot be modified by this feature:
  - forwarded
  - host
  - via

- warning
- x-forwarded-for
- All header names that start with "x-ec" are reserved.

## Modify Client Response Header

Each response contains a set of [response headers]() that describe it. This feature can either:

- Append or overwrite the value assigned to a response header. If the specified request header does not exist, then this feature will add it to the response.
- Delete a response header from the response.

By default, response header values are defined by an origin server and by our edge servers.

One of the following actions can be performed on a response header:

OPTION	DESCRIPTION	EXAMPLE
Append	The specified value will be added to end of the existing request header value.	<b>Response header value (Client):</b> Value1 <b>Response header value (HTTP Rules Engine):</b> Value2 <b>New Response header value:</b> Value1Value2
Overwrite	The request header value will be set to the specified value.	<b>Response header value (Client):</b> Value1 <b>Response header value (HTTP Rules Engine):</b> Value2 <b>New response header value:</b> Value2
Delete	Deletes the specified request header.	<b>Request header value (Client):</b> Value1 <b>Modify Client Request Header configuration:</b> Delete the response header in question. <b>Result:</b> The specified response header will not be forwarded to the requester.

Key information:

- Make sure that the value specified in the Name option is an exact match for the desired response header.
- Case is not taken into account for the purpose of identifying a header. For example, any of the following variations of the Cache-Control header name can be used to identify it:
  - cache-control
  - CACHE-CONTROL
  - cachE-Control
- Deleting a header will prevent it from being forwarded to the requester.
- The following headers are reserved and cannot be modified by this feature:
  - accept-encoding
  - age
  - connection
  - content-encoding
  - content-length
  - content-range
  - date

- server
- trailer
- transfer-encoding
- upgrade
- vary
- via
- warning
- All header names that start with "x-ec" are reserved.

### Set Client IP Custom Header

**Purpose:** Adds a custom header that identifies the requesting client by IP address to the request.

The Header name option defines the name of the custom request header where the client's IP address will be stored.

This feature allows a customer origin server to find out client IP addresses through a custom request header. If the request is served from cache, then the origin server will not be informed of the client's IP address. Therefore, it is recommended that this feature be used with ADN or assets that will not be cached.

Please make sure that the specified header name does not match any of the following:

- Standard request header names. A list of standard header names can be found in [RFC 2616](#).
- Reserved header names:
  - forwarded-for
  - host
  - vary
  - via
  - warning
  - x-forwarded-for
  - All header names that start with "x-ec" are reserved.

## Logs

These features are designed to customize the data stored in raw log files.

NAME	PURPOSE
Custom Log Field 1	Determines the format and the content that will be assigned to the custom log field in a raw log file.
Log Query String	Determines whether a query string will be stored along with the URL in access logs.

### Custom Log Field 1

**Purpose:** Determines the format and the content that will be assigned to the custom log field in a raw log file.

The main purpose behind this custom field is to allow you to determine which request and response header values will be stored in your log files.

By default, the custom log field is called "x-ec\_custom-1." However, the name of this field can be customized from the [Raw Log Settings page]().

The formatting that you should use to specify request and response headers is defined below.

HEADER TYPE	FORMAT	EXAMPLES
Request Header	%{[RequestHeader()][i]}()	%{Accept-Encoding}i {Referer}i {Authorization}i
Response Header	%{[ResponseHeader()][o]}()	%{Age}o {Content-Type}o {Cookie}o

Key information:

- A custom log field can contain any combination of header fields and plain text.
- Valid characters for this field include the following: alphanumeric (i.e., 0-9, a-z, and A-Z), dashes, colons, semi-colons, apostrophes, commas, periods, underscores, equal signs, parentheses, brackets, and spaces. The percentage symbol and curly braces are only allowed when used to specify a header field.
- The spelling for each specified header field must match the desired request/response header name.
- If you would like to specify multiple headers, then it is recommended that you use a separator to indicate each header. For example, you could use an abbreviation for each header. Sample syntax is provided below.
  - AE: %{Accept-Encoding}i A: %{Authorization}i CT: %{Content-Type}o

**Default Value:** -

### Log Query String

**Purpose:** Determines whether a query string will be stored along with the URL in access logs.

VALUE	RESULT
Enabled	Allows the storage of query strings when recording URLs in an access log. If a URL does not contain a query string, then this option will not have an effect.
Disabled	Restores the default behavior. The default behavior is to ignore query strings when recording URLs in an access log.

**Default Behavior:** Disabled.

## Origin

These features are designed to control how the CDN communicates with an origin server.

NAME	PURPOSE
Maximum Keep-Alive Requests	Defines the maximum number of requests for a Keep-Alive connection before it is closed.
Proxy Special Headers	Defines the set of CDN-specific request headers that will be forwarded from an edge server to an origin server.

### Maximum Keep-Alive Requests

**Purpose:** Defines the maximum number of requests for a Keep-Alive connection before it is closed.

Setting the maximum number of requests to a low value is strongly discouraged and may result in performance degradation.

Key information:

- Specify this value as a whole integer.
- Do not include commas or periods in the specified value.

**Default Value:** 10,000 requests

### Proxy Special Headers

**Purpose:** Defines the set of [CDN-specific request headers]() that will be forwarded from an edge server to an origin server.

Key information:

- Each CDN-specific request header defined in this feature will be forwarded to an origin server.
- Prevent a CDN-specific request header from being forwarded to an origin server by removing it from this list.

**Default Behavior:** All [CDN-specific request headers]() will be forwarded to the origin server.

## Specialty

These features provide advanced functionality that should only be used by advanced users.

NAME	PURPOSE
Cacheable HTTP Methods	Determines the set of additional HTTP methods that can be cached on our network.
Cacheable Request Body Size	Defines the threshold for determining whether a POST response can be cached.

### Cacheable HTTP Methods

**Purpose:** Determines the set of additional HTTP methods that can be cached on our network.

Key information:

- This feature assumes that GET responses should always be cached. As a result, the GET HTTP method should not be included when setting this feature.
- This feature only supports the POST HTTP method. Enable POST response caching by setting this feature to:POST
- By default, only requests whose body is smaller than 14 Kb will be cached. Use the Cacheable Request Body Size Feature to set the maximum request body size.

**Default Behavior:** Only GET responses will be cached.

### Cacheable Request Body Size

**Purpose:** Defines the threshold for determining whether a POST response can be cached.

This threshold is determined by specifying a maximum request body size. Requests that contain a larger request body will not be cached.

Key information:

- This Feature is only applicable when POST responses are eligible for caching. Use the Cacheable HTTP Methods Feature to enable POST request caching.
- The request body is taken into consideration for:
  - x-www-form-urlencoded values
  - Ensuring a unique cache-key

- Defining a large maximum request body size may impact data delivery performance.
  - **Recommended Value:** 14 Kb
  - **Minimum Value:** 1 Kb

**Default Behavior:** 14 Kb

## URL

These features allow a request to be redirected or rewritten to a different URL.

NAME	PURPOSE
Follow Redirects	Determines whether requests can be redirected to the hostname defined in the Location header returned by a customer origin server.
URL Redirect	Redirects requests via the Location header.
URL Rewrite	Rewrites the request URL.

### Follow Redirects

**Purpose:** Determines whether requests can be redirected to the hostname defined in the Location header returned by a customer origin server.

Key information:

- Requests can only be redirected to edge CNAMEs that correspond to the same platform.

VALUE	RESULT
Enabled	Requests can be redirected.
Disabled	Requests will not be redirected.

**Default Behavior:** Disabled.

### URL Redirect

**Purpose:** Redirects requests via the Location header.

The configuration of this feature requires setting the following options:

OPTION	DESCRIPTION
Code	Select the response code that will be returned to the requester.

OPTION	DESCRIPTION
Source & Pattern	<p>These settings define a request URI pattern that identifies the type of requests that may be redirected. Only requests whose URL satisfies both of the following criteria will be redirected:</p> <p><b>Source :</b> (or content access point) Select a relative path that identifies an origin server. This is the "/XXXX/" section and your endpoint name.</p> <p><b>Source (pattern):</b> A pattern that identifies requests by relative path must be defined. This regular expression pattern must define a path that starts directly after the previously selected content access point (see above).</p> <ul style="list-style-type: none"> <li>- Make sure that the request URI criteria (i.e., Source &amp; Pattern) defined above doesn't conflict with any match conditions defined for this feature.</li> <li>- Make sure to specify a pattern. Using a blank value as the pattern will only match requests to the root folder of the selected origin server (e.g., <a href="http://cdn.mydomain.com/">http://cdn.mydomain.com/</a>).</li> </ul>
Destination	<p>Define the URL to which the above requests will be redirected.</p> <p>Dynamically construct this URL using:</p> <ul style="list-style-type: none"> <li>- A regular expression pattern</li> <li>- HTTP variables</li> </ul> <p>Substitute the values captured in the source pattern into the destination pattern using \$<i>n</i> where <i>n</i> identifies a value by the order in which it was captured. For example, \$1 represents the first value captured in the source pattern, while \$2 represents the second value.</p>

It is highly recommended to use an absolute URL. The use of a relative URL may redirect CDN URLs to an invalid path.

### Sample Scenario

In this example, we will demonstrate how to redirect an edge CNAME URL that resolves to this base CDN URL: <http://marketing.azureedge.net/brochures>

Qualifying requests will be redirected to this base edge CNAME URL: <http://cdn.mydomain.com/resources>

This URL redirection may be achieved through the following configuration:

### Key points:

- The URL Redirect feature defines the request URLs that will be redirected. As a result, additional match conditions are not required. Although the match condition was defined as "Always," only requests that point to the "brochures" folder on the "marketing" customer origin will be redirected.
- All matching requests will be redirected to the edge CNAME URL defined in the Destination option.
  - Sample scenario #1:
    - Sample request (CDN URL): <http://marketing.azureedge.net/brochures/widgets.pdf>
    - Request URL (after redirect): <http://cdn.mydomain.com/resources/widgets.pdf>
  - Sample scenario #2:
    - Sample request (Edge CNAME URL): <http://marketing.mydomain.com/brochures/widgets.pdf>
    - Request URL (after redirect): <http://cdn.mydomain.com/resources/widgets.pdf>
  - Sample scenario #3:



- Sample request (Edge CNAME URL): <http://brochures.mydomain.com/campaignA/final/productC.ppt>
- Request URL (after redirect): <http://cdn.mydomain.com/resources/campaignA/final/productC.ppt>
- The Request Scheme (%{scheme}) variable was leveraged in the Destination option. This ensures that the request's scheme remains unchanged after redirection.
- The URL segments that were captured from the request are appended to the new URL via "\$1."

## URL Rewrite

**Purpose:** Rewrites the request URL.

Key information:

- The configuration of this feature requires setting the following options:

OPTION	DESCRIPTION
Source & Pattern	<p>These settings define a request URI pattern that identifies the type of requests that may be rewritten. Only requests whose URL satisfies both of the following criteria will be rewritten:</p> <ul style="list-style-type: none"> <li>- <b>Source (or content access point):</b> Select a relative path that identifies an origin server. This is the "/XXXX/" section and your endpoint name.</li> <li>- <b>Source (pattern):</b> A pattern that identifies requests by relative path must be defined. This regular expression pattern must define a path that starts directly after the previously selected content access point (see above). Make sure that the request URI criteria (i.e., Source &amp; Pattern) defined above doesn't conflict with any of the match conditions defined for this feature. Make sure to specify a pattern. Using a blank value as the pattern will only match requests to the root folder of the selected origin server (e.g., <a href="http://cdn.mydomain.com/">http://cdn.mydomain.com/</a>).</li> </ul>
Destination	<p>Define the relative URL to which the above requests will be rewritten by:</p> <ol style="list-style-type: none"> <li>1. Selecting a content access point that identifies an origin server.</li> <li>2. Defining a relative path using: <ul style="list-style-type: none"> <li>- A regular expression pattern</li> <li>- HTTP variables</li> </ul> </li> </ol> <p>Substitute the values captured in the source pattern into the destination pattern using \$<i>n</i> where <i>n</i> identifies a value by the order in which it was captured. For example, \$1 represents the first value captured in the source pattern, while \$2 represents the second value.</p>

This feature allows our edge servers to rewrite the URL without performing a traditional redirect. This means that the requester will receive the same response code as if the rewritten URL had been requested.

## Sample Scenario 1

In this example, we will demonstrate how to redirect an edge CNAME URL that resolves to this base CDN URL: <http://marketing.azureedge.net/brochures/>

Qualifying requests will be redirected to this base edge CNAME URL: <http://MyOrigin.azureedge.net/resources/>

This URL redirection may be achieved through the following configuration:

The screenshot shows a configuration interface for a URL Rewrite rule. It includes a dropdown for 'IF' set to 'Always', a 'Features' section with a plus icon and 'URL Rewrite' selected, and input fields for 'Source' (containing '/800001/marketing/' and 'brochures/(.\*)') and 'Destination' (containing '/800001/MyOrigin/' and 'resources/\$1').

## Sample Scenario 2

In this example, we will demonstrate how to redirect an edge CNAME URL from UPPERCASE to lowercase using regular expressions.

This URL redirection may be achieved through the following configuration:



### Key points:

- The URL Rewrite feature defines the request URLs that will be rewritten. As a result, additional match conditions are not required. Although the match condition was defined as "Always," only requests that point to the "brochures" folder on the "marketing" customer origin will be rewritten.
- The URL segments that were captured from the request are appended to the new URL via "\$1."

### Compatibility

This feature includes matching criteria that must be met before it can be applied to a request. In order to prevent setting up conflicting match criteria, this feature is incompatible with the following match conditions:

- AS Number
- CDN Origin
- Client IP Address
- Customer Origin
- Request Scheme
- URL Path Directory
- URL Path Extension
- URL Path Filename
- URL Path Literal
- URL Path Regex
- URL Path Wildcard
- URL Query Literal
- URL Query Parameter
- URL Query Regex
- URL Query Wildcard

## Next Steps

- [Rules Engine Reference](#)
- [Rules Engine Conditional Expressions](#)
- [Rules Engine Match Conditions](#)
- [Overriding default HTTP behavior using the rules engine](#)
- [Azure CDN Overview](#)

# Azure CDN POP Locations

1/25/2017 • 1 min to read • [Edit Online](#)

This topic lists current POP locations for **Azure CDN from Verizon** and **Azure CDN from Akamai**.

## IMPORTANT

\***Azure CDN from Akamai** POP locations are not individually disclosed.

Both providers have distinct ways of building their CDN infrastructures. We recommend against using POP locations to decide which Azure CDN product to use, and instead consider features and end-user performance. Test the performance with both providers to choose the right Azure CDN product for your users.

REGION	VERIZON	AKAMAI
North America	Atlanta, GA Philadelphia, PA New York, NY Miami, FL Washington DC Boston, MA Denver, CO Chicago, IL Dallas, TX Los Angeles, CA San Jose, CA Seattle, WA	✓ *
South America	São Paulo, Brazil Rio de Janeiro, Brazil Quito, Ecuador Barranquilla, Colombia Medellin, Colombia Buenos Aires, Argentina	✓ *
North and East Europe	Copenhagen, Denmark Helsinki, Finland London, UK Stockholm, Sweden Warsaw, Poland	✓ *
West Europe	Amsterdam, Netherlands Frankfurt, Germany Paris, France Vienna, Austria	✓ *
South Europe	Madrid, Spain Milan, Italy	✓ *

REGION	VERIZON	AKAMAI
East Asia	Tokyo, Japan Osaka, Japan Batam, Indonesia Jakarta, Indonesia Hong Kong Kaohsiung, Taiwan Seoul, South Korea Singapore	✓*
South and Central Asia	Bangalore, India Chennai, India Delhi, India Mumbai, India	✓*
Middle East/West Asia	Muscat, Oman	✓*
Africa		✓*
Australia and New Zealand	Melbourne, Australia Sydney, Australia	✓*

## See Also

- [Azure CDN Edge Nodes API to get latest IP addresses for whitelisting](#)