100 XP

# Introduction

3 minutes

A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users. CDNs' store cached content on edge servers in point-of-presence (POP) locations that are close to end users, to minimize latency.

After completing this module, you'll be able to:

- Explain how the Azure Content Delivery Network works and how it can improve the user experience.
- Control caching behavior and purge content.
- Perform actions on Azure CDN by using the Azure CDN Library for .NET.

## Next unit: Explore Azure Content Delivery Networks

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Explore Azure Content Delivery Networks

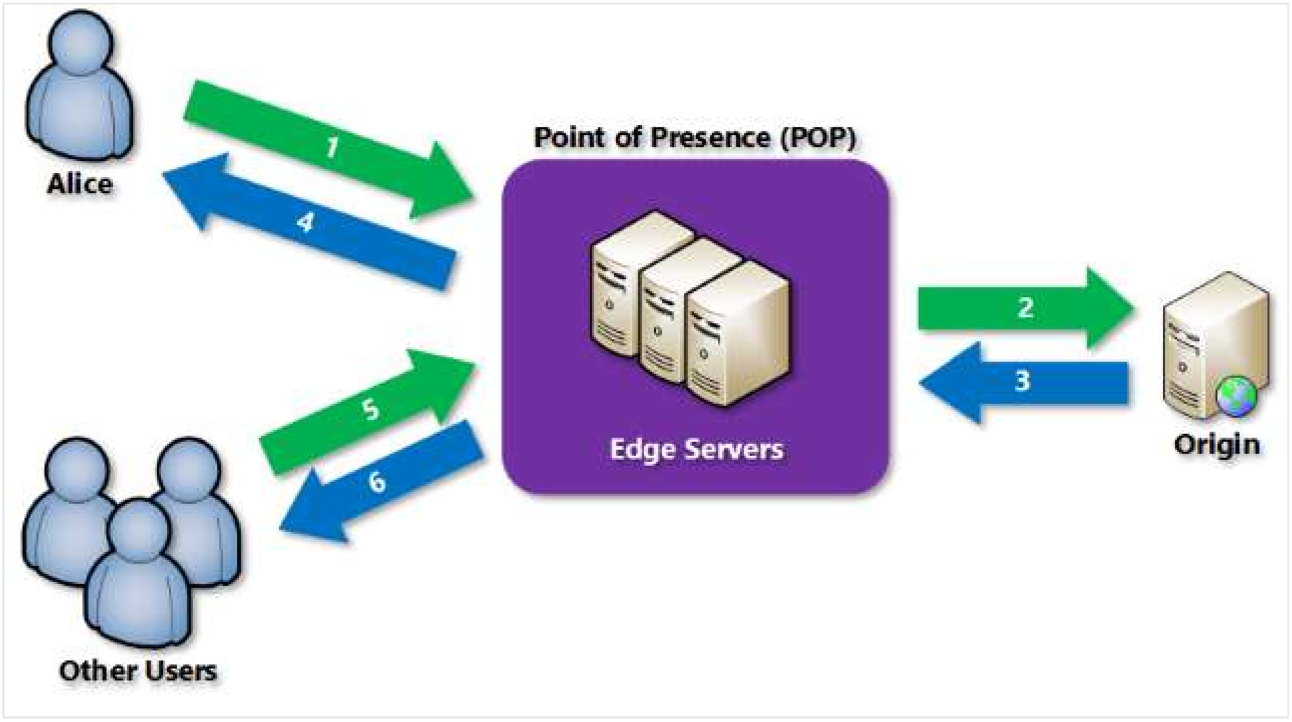
3 minutes

Azure Content Delivery Network (CDN) offers developers a global solution for rapidly delivering high-bandwidth content to users by caching their content at strategically placed physical nodes across the world. Azure CDN can also accelerate dynamic content, which cannot be cached, by leveraging various network optimizations using CDN POPs. For example, route optimization to bypass Border Gateway Protocol (BGP).

The benefits of using Azure CDN to deliver web site assets include:

- Better performance and improved user experience for end users, especially when using applications in which multiple round-trips are required to load content.
- Large scaling to better handle instantaneous high loads, such as the start of a product launch event.
- Distribution of user requests and serving of content directly from edge servers so that less traffic is sent to the origin server.

## How Azure Content Delivery Network works



1. A user (Alice) requests a file (also called an asset) by using a URL with a special domain name, such as `<endpoint name>.azureedge.net`. This name can be an endpoint hostname or a custom domain. The DNS routes the request to the best performing POP location, which is usually the POP that is geographically closest to the user.
2. If no edge servers in the POP have the file in their cache, the POP requests the file from the origin server. The origin server can be an Azure Web App, Azure Cloud Service, Azure Storage account, or any publicly accessible web server.
3. The origin server returns the file to an edge server in the POP.
4. An edge server in the POP caches the file and returns the file to the original requestor (Alice). The file remains cached on the edge server in the POP until the time-to-live (TTL) specified by its HTTP headers expires. If the origin server didn't specify a TTL, the default TTL is seven days.
5. Additional users can then request the same file by using the same URL that Alice used, and can also be directed to the same POP.
6. If the TTL for the file hasn't expired, the POP edge server returns the file directly from the cache. This process results in a faster, more responsive user experience.

## Requirements

To use Azure CDN you need to create at least one CDN profile, which is a collection of CDN endpoints. Every CDN endpoint represents a specific configuration of content deliver behavior and access. To organize your CDN endpoints by internet domain, web application, or some other criteria, you can use multiple profiles. Because Azure CDN pricing is applied at the CDN profile level, you must create multiple CDN profiles if you want to use a mix of pricing tiers.

### Limitations

Each Azure subscription has default limits for the following resources:

- The number of CDN profiles that can be created.
- The number of endpoints that can be created in a CDN profile.
- The number of custom domains that can be mapped to an endpoint.

For more information about CDN subscription limits, visit CDN limits.

## Azure Content Delivery Network products

Azure Content Delivery Network (CDN) includes four products:

- Azure CDN Standard from Microsoft
- Azure CDN Standard from Akamai
- Azure CDN Standard from Verizon
- Azure CDN Premium from Verizon

Visit the product comparison for a detailed feature comparison between the four products.

### Next unit: Control cache behavior on Azure Content Delivery Networks

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

✓ 100 XP

# Control cache behavior on Azure Content Delivery Networks

3 minutes

Because a cached resource can potentially be out-of-date or stale (compared to the corresponding resource on the origin server), it is important for any caching mechanism to control when content is refreshed. To save time and bandwidth consumption, a cached resource is not compared to the version on the origin server every time it is accessed. Instead, as long as a cached resource is considered to be fresh, it is assumed to be the most current version and is sent directly to the client. A cached resource is considered to be fresh when its age is less than the age or period defined by a cache setting. For example, when a browser reloads a webpage, it verifies that each cached resource on your hard drive is fresh and loads it. If the resource is not fresh (stale), an up-to-date copy is loaded from the server.

## Controlling caching behavior

Azure CDNs provide two mechanisms for caching files. However, these configuration settings depend on the tier you've selected. Caching rules in Azure CDN Standard for Microsoft are set at the endpoint level and provide three configuration options. Other tiers provide additional configuration options, which include:

- **Caching rules**. Caching rules can be either global (apply to all content from a specified endpoint) or custom. Custom rules apply to specific paths and file extensions.
- **Query string caching**. Query string caching enables you to configure how Azure CDN responds to a query string. Query string caching has no effect on files that can't be cached.

With the Azure CDN Standard for Microsoft Tier, caching rules are as simple as the following three options:

- Ignore query strings. This option is the default mode. A CDN POP simply passes the request and any query strings directly to the origin server on the first request and caches the asset. New requests for the same asset will ignore any query strings until the TTL expires.

- Bypass caching for query strings. Each query request from the client is passed directly to the origin server with no caching.
- Cache every unique URL. Every time a requesting client generates a unique URL, that URL is passed back to the origin server and the response cached with its own TTL. This final method is inefficient where each request is a unique URL, as the cache-hit ratio becomes low.

To change these settings, in the Endpoint pane, select **Caching rules** and then select the caching option that you want to apply to the endpoint and select **Save**.

## Caching and time to live

If you publish a website through Azure CDN, the files on that site are cached until their TTL expires. The Cache-Control header contained in the HTTP response from origin server determines the TTL duration.

If you don't set a TTL on a file, Azure CDN sets a default value. However, this default may be overridden if you have set up caching rules in Azure. Default TTL values are as follows:

- Generalized web delivery optimizations: seven days
- Large file optimizations: one day
- Media streaming optimizations: one year

## Content updating

In normal operation, an Azure CDN edge node will serve an asset until its TTL expires. The edge node reconnects to the origin server when the TTL expires and a client makes a request to the same asset. The node will fetch another copy of the asset, resetting the TTL in the process.

To ensure that users always receive the latest version of an asset, consider including a version string in the asset URL. This approach causes the CDN to retrieve the new asset immediately.

Alternatively, you can purge cached content from the edge nodes, which refreshes the content on the next client request. You might purge cached content when publishing a new version of a web app or to replace any out-of-date assets.

You can purge content in several ways.

- On an endpoint by endpoint basis, or all endpoints simultaneously should you want to update everything on your CDN at once.

- Specify a file, by including the path to that file or all assets on the selected endpoint by checking the **Purge All** checkbox in the Azure portal.
- Based on wildcards (*) or using the root (/).

The Azure CLI provides a special purge verb that will unpublish cached assets from an endpoint. This is very useful if you have an application scenario where a large amount of data is invalidated and should be updated in the cache. To unpublish assets, you must specify either a file path, a wildcard directory, or both:

Bash Copy

```bash
az cdn endpoint purge \
    --content-paths '/css/*' '/js/app.js' \
    --name ContosoEndpoint \
    --profile-name DemoProfile \
    --resource-group ExampleGroup
```

You can also preload assets into an endpoint. This is useful for scenarios where your application creates a large number of assets, and you want to improve the user experience by prepopulating the cache before any actual requests occur:

Bash Copy

```bash
az cdn endpoint load \
    --content-paths '/img/*' '/js/module.js' \
    --name ContosoEndpoint \
    --profile-name DemoProfile \
    --resource-group ExampleGroup
```

## Geo-filtering

Geo-filtering enables you to allow or block content in specific countries, based on the country code. In the Azure CDN Standard for Microsoft Tier, you can only allow or block the entire site. With the Verizon and Akamai tiers, you can also set up restrictions on directory paths. For more information, see the further reading section in the Summary unit.

Next unit: Interact with Azure Content Delivery Networks by using .NET

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Interact with Azure Content Delivery Networks by using .NET

3 minutes

You can use the Azure CDN Library for .NET to automate creation and management of CDN profiles and endpoints. Install the Microsoft.Azure.Management.Cdn.Fluent directly from the Visual Studio Package Manager console or with the .NET Core CLI.

In this unit you will see code examples illustrating common actions.

## Create a CDN client

The example below shows creating a client by using the `CdnManagementClient` class.

C# Copy

```
static void Main(string[] args)
{
    // Create CDN client
    CdnManagementClient cdn = new CdnManagementClient(new
TokenCredentials(authResult.AccessToken))
        { SubscriptionId = subscriptionId };
}
```

## List CDN profiles and endpoints

The first thing the method below 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.

C# Copy

```
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

The example below shows creating an Azure CDN profile.

C# Copy

```
private static void CreateCdnProfile(CdnManagementClient cdn)
{
    if (profileAlreadyExists)
    {
        //Check to see if the profile already exists
    }
    else
    {
        //Create the new profile
        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.

C# Copy

```
private static void CreateCdnEndpoint(CdnManagementClient cdn)
{
    if (endpointAlreadyExists)
    {
        //Check to see if the endpoint already exists
    }
    else
    {
        //Create the new endpoint
        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, resourceGroup-
Name);
    }
}
```

## Purge an endpoint

A common task that we might want to perform is purging the content in our endpoint.

C# Copy

```
private static void PromptPurgeCdnEndpoint(CdnManagementClient cdn)
{
    if (PromptUser(String.Format("Purge CDN endpoint {0}?", endpointName)))
    {
        Console.WriteLine("Purging endpoint. Please wait...");
        cdn.Endpoints.PurgeContent(resourceGroupName, profileName, endpointName, new
List<string>() { "/*" });
        Console.WriteLine("Done.");
        Console.WriteLine();
    }
}
```

## Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

✓ 200 XP

# Knowledge check

3 minutes

## Check your knowledge

**1.** Each Azure subscription has default limits on resources needed for an Azure Content Delivery Network. Which of the following resources has subscription limitations that may impact your solution?

- ○ Resource group
- [✓ CDN profiles]

  ✓ **That's correct. The number of CDN profiles that can be created is limited by the type of Azure subscription.**
- ○ Storage account

**2.** When publishing a website through Azure CDN, the files on that site are cached until their time-to-live (TTL) expires. What is the default TTL for large file optimizations?

- ○ One day

  ✓ **That's correct. The default TTL for large file optimizations is one day.**
- [✓ One week]

  ✗ **That's incorrect. Generalized web delivery optimizations have a default TTL of one week.**
- ○ One year

## Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

✓ 100 XP

# Summary

3 minutes

In this module, you learned how to:

- Explain how the Azure Content Delivery Network works and how it can improve the user experience.
- Control caching behavior and purge content.
- Perform actions on Azure CDN by using the Azure CDN Library for .NET.

## Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆