# Azure Developer Immersions

Content Delivery Network

A Content Delivery Network (CDN) helps web sites scale by caching copies of content in multiple datacenters across the planet and arranging for end users to download whichever copy of that content is closest to them. Not only does this reduce the load on your web server, but it can also significantly improve performance for any users who happen not to be located near to whichever datacenter your web server is in; pages load faster when most of the assets can be downloaded from a CDN server in a nearby datacenter instead of having to fetch them from one that is many thousands of miles away.

CDNs are most effective for static content, or files that do not change frequently. The usual best candidates are image files (e.g., jpeg, png, or svg), CSS, and JavaScript. HTML does not always benefit; it depends on the application. In our case, the most frequently used HTML pages change frequently—perhaps even several times a minute—because they reflect user activity. In an application with frequently accessed content that might not change for several days (e.g., a marketing website during a high profile event launch), using a CDN for HTML would make sense. For our application, however, we will just be applying it to graphics, CSS, and script.

There are two exercises in this walkthrough:

1. Create a CDN profile
2. Serve graphics, script, and CSS via the CDN

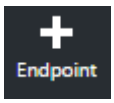
## Exercise 1: Create a CDN profile

You will create a new CDN profile in Azure in this exercise. You will also create an endpoint in that profile to serve content from your web app.

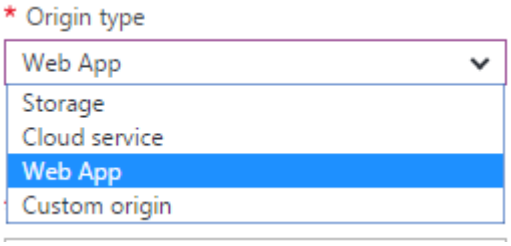
1. In the Azure portal, click **+ New**, select **Media + CDN** and, in the list that appears, find and select **CDN**.


2. Set the **Name** to **rGroup**.
3. Set the **Resource group** to **rGroup**.

4. Set the **Pricing tier** to **S2 Standard Akamai**. (The main reason we're using Akamai here is that newly-created endpoints take only a few minutes to become available with Akamai, whereas with a Verizon profile, it typically takes 90 minutes or more for new configuration to propagate through the network.)
5. Click **Create**. Azure will create the CDN profile. This will take a few minutes. We can't modify the web app to use the CDN until we've established the domain name our cache will use, and we can't do that until we've set up the profile, so unfortunately you'll have to wait.
6. Once Azure has finished creating the profile, open the blade for it. As always, you can just click the relevant icon in your resource group.
7. In the button bar near the top of the CDN profile blade, click the **Endpoint** button to add a new endpoint. (A single CDN profile can work with content from multiple web servers, and you set up one endpoint per server. We have only one web server, so we need just one endpoint.)




8. Type in a name for the endpoint. This will need to be one that is not already taken. If you choose one that is in use, Azure will tell you by showing a red exclamation mark next to the name. You'll need to pick a variation that makes it show a green tick.
9. In the **Origin type** dropdown, select **Web App**.




10. The **Origin hostname** dropdown should show a list of all the Azure Web Apps in your subscription. Select the one you've been using in these labs.
11. Click **Add**.
12. It will take a few minutes to create the endpoint, but now that you have the endpoint name, you can move on to the next exercise.

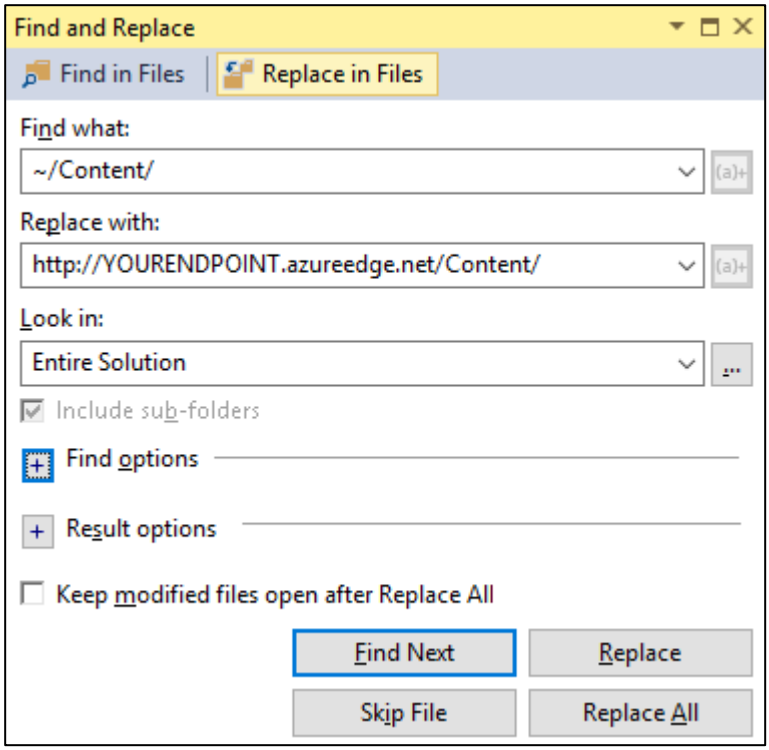
## Exercise 2: Serve graphics, script, and CSS through the CDN

In this exercise, you will modify your web app to serve static content through the CDN.

1. In Visual Studio, in the **Rg.Web** project's **Views\Shared** folder, open **_Layout.cshtml**.
2. Around 10 lines from the top, you will find some **<link>** elements bringing in CSS files:

```
<link rel="stylesheet" href="~/Content/bootstrap.min.css" />
<link rel="stylesheet" href="~/Content/Site.css" />
```

3. These links and others like them need to be changed to refer to your new CDN endpoint. You can update all of them with a global search and replace. Select the **Edit | Find and Replace | Replace in Files** menu item (or just type **Ctrl+Shift+H**).
4. In the **Find and Replace** dialog, type **~/Content/** in the **Find what** field. In **Replace with** type a URL of the form **https://YOURENDPOINT.azureedge.net/Content/**, but replace YOURENDPOINT with the name of the endpoint you created in the first exercise.




5. Click **Replace All**.
6. Visual Studio will search every file in your solution and replace links to anything in the Content folder with equivalent links going via the CDN. (It should find and modify five links.)
7. Right-click on **Rg.Web** in **Solution Explorer** and select **Set as StartUp Project**.
8. Press **F5** to run the web app locally.
9. Log in to the app. It should all look exactly as it did before.
10. Open your browser's debugging tools (in most browsers, pressing **F12** will do this). Find the **Network** tab. (The details are slightly different, but all the popular web browsers have a Network tab.). Click the green arrow to start capturing network traffic.
11. Refresh the page.
12. You should be able to see that while the main page itself came from your local web server, the browser went to your CDN endpoint for CSS resources. Here's how that looks in the debug tools for Microsoft's web browsers (IE and Edge look very similar):

| | | | | |
|---|---|---|---|---|
| http://localhost:31893/ | HTTP | GET | 200 OK | text/html |
| bootstrap.min.css http://rgroup101.azureedge.net/Content/ | HTTP | GET | 304 Not Modified | text/css |
| Site.css http://rgroup101.azureedge.net/Content/ | HTTP | GET | 304 Not Modified | text/css |
| modernizr.js http://localhost:31893/Scripts/ | HTTP | GET | 304 Not Modified | application/java... |

**Note:** You might see a **302 Moved Temporarily** response from the CDN and if you inspect the response you'll see that it redirected the browser to load the resource from your web app in Azure. This happens the very first time you use a particular resource; if the CDN doesn't yet have a cached copy it will start fetching it at that moment, but to avoid unnecessary delays, it redirects the browser to fetch the file from the original web site. It can sometimes take a few minutes for cached content to propagate to all of the CDN's servers, but you will eventually find all the resources that report a 302 will at some point stop doing that.

13. You can see that script resources are still being fetched locally, because they live in the **Scripts** folder, and we only updated links for the **Content** folder.
14. Perform another **Replace in Files** search and replace, this time replacing **~/Scripts/** with **https://YOURENDPOINT.azureedge.net/Scripts/**, again replacing YOURENDPOINT with the name of your CDN endpoint.
15. The web site uses various images (mostly SVG and JPEG files) store in the Images folder. You will need to be a bit more careful with these because they are not so easy to search for. Because many of the links are in in CSS files, they do not start with the "~" character. (That's an ASP.NET-ism and doesn't work in CSS.)Doing a global replace of just **/Images/** won't work because you'll end up modifying a completely unrelated jquery-related URL. So you need to be more selective.
    a. Again, choose the **Edit | Find and Replace | Replace in Files** menu item.
    b. Set **Find what** to **../Images/** (note the two '.' characters at the start)
    c. Set **Replace with** to **https://YOURENDPOINT.azureedge.net/Scripts/**, again replacing YOURENDPOINT with the name of your CDN endpoint
    d. Expand the **Find options** section
    e. In **Look at these file types,** type ***.css**
    f. Click **Replace All**. This should replace seven items.
    g. Change **Find what** to just **/Images/** (removing the ..)
    h. Leave **Replace with** unchanged.
    i. Change **Look at these file type** to ***.cs;*.cshtml**
    j. Click **Replace All**. This should replace 8 items.
    k. This will break one link. Open **_LayoutNoUser** in the **Views\Shared** folder, go to line 41 and you will find this:

```
<img src="~https://rgroup101.azureedge.net/Images/temp-avatar.jpg"
alt="temp-avatar" class="avatar header-avatar">
```

l. Delete the ~ from the front of that URL.

16. Debug the app again, check the **Network** tab in the browser developer tools again, and you should now see that static image and JavaScript resources are also coming from the CDN. (User-uploaded images such as avatars are still coming directly from the web server, but we need that to happen because those should only be accessible to authenticated users. CDN servers are not able to implement security policy for us.)
17. Before we push these changes to Azure, there's one more thing to do: we should explicitly configure the default cache settings for our static content. In the **Rg.Web** project, open **Web.config**. Find the **<staticContent>** section. Add the highlighted line:

```
<staticContent>
  <remove fileExtension=".svg" />
  <mimeMap fileExtension=".svg" mimeType="image/svg+xml" />
  <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="10.00:00:00"/>
</staticContent>
```

This indicates that we want all static content to be cached for ten days. Depending on how often you think you might want to change any of your static resources, you could adjust this up or down.

18. Right click on the **Rg.Web** project, choose **Publish**, and then click **Publish** to deploy your updated site to Azure.
19. Log into the application and verify that it all looks as it should. Check with the browser developer tools that static graphics, script, and CSS are now being fetched from the CDN, and not your web server.

Let's review what you have done. You created a new Azure CDN profile, and in that profile, you configured a new endpoint to cache your Web App's static content in servers distributed around the world. You then modified the web app to use URLs that refer to the CDN endpoint for all static images, script, and CSS. This means that, for the numerous resources of this type, web browser will no longer hit your web server. Instead, they will fetch these resources from the CDN servers nearest to them, reducing the load on your web server and improving the page load time for your users.