Implementation of CDN (Content Delivery Network) to serve web contents to end-users with high availability and high speed.

By Prodip K Saha
Date: September 26, 2013

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centers across the Internet. The idea of CDN is to bring the contents closer to the end user and serve them from there.

I have had the privilege and opportunity to evaluate Content Delivery Network (CDN) , help management to select the provider who can provide best services with least cost, design the solution and lead the implementation effort (make it happen).

Implementation of CDN can be as easy as 1-2-3 but you would end up in trouble (technically and financially) unless you do the due diligence. Frist thing first, you do need to write down the must-have requirements before evaluating CDN providers. Next, you do need conceptual design and physical design backed by a working POC.

Requirements:

- User experience increase by 25-30 percent. Example: a page takes 2 seconds to load in user's browser without CDN. The same page takes 1.5 seconds with CDN. That's 25% increase [(2.0-1.5)/2*100]. Keep in mind your goal is to improve the user experience without changing the application codes.
- Static contents must be delivered over SSL for two reasons. One- to protect the data in transit. Two- to avoid being prompted for switching between http and https protocol.
- Ability to auto purge contents upon deployment of codes at the origin servers.
- Optional: Websites must be able to counter distributed denial of service attack (DDOS).
- Think now: Ability to switch CDN vendor seamlessly (if and when warranted).

Design Matters

There are two broader design choices I have evaluated and tested- URL Rewrite and CNAME (DNS Change). Both works fine and they both have pros and cons.

The easiest way to design CDN integration is to create a CNAME and point it to CDN provided DNS. CDN provider in turn will come to origin IP (VIP) to cache the contents. Sounds so easy but there are complexities when you have to have SSL. Design gets further complicated if you have the provision for DDOS protection (double proxy).

Url Rewrite	
Pros	**Cons**
Performance increase: 20-30% and up.	Must rewrite URL at the origin.
Control over DNS and SSL certificate.	Url rewrite operation on a page can take few milliseconds.
Provision for DDOS in parallel. That mean both CDN and DDOS can run at the same time.	Mixed URL (static from CDN and dynamic from Origin).

Save on bandwidth (static traffics go through CDN)	
Automatic Content Refresh upon deployment.	
Authentication and Authorization of resources can be better managed.	
Seamless transition to new vendor. You can switch to another vendor and it can be done in parallel.	

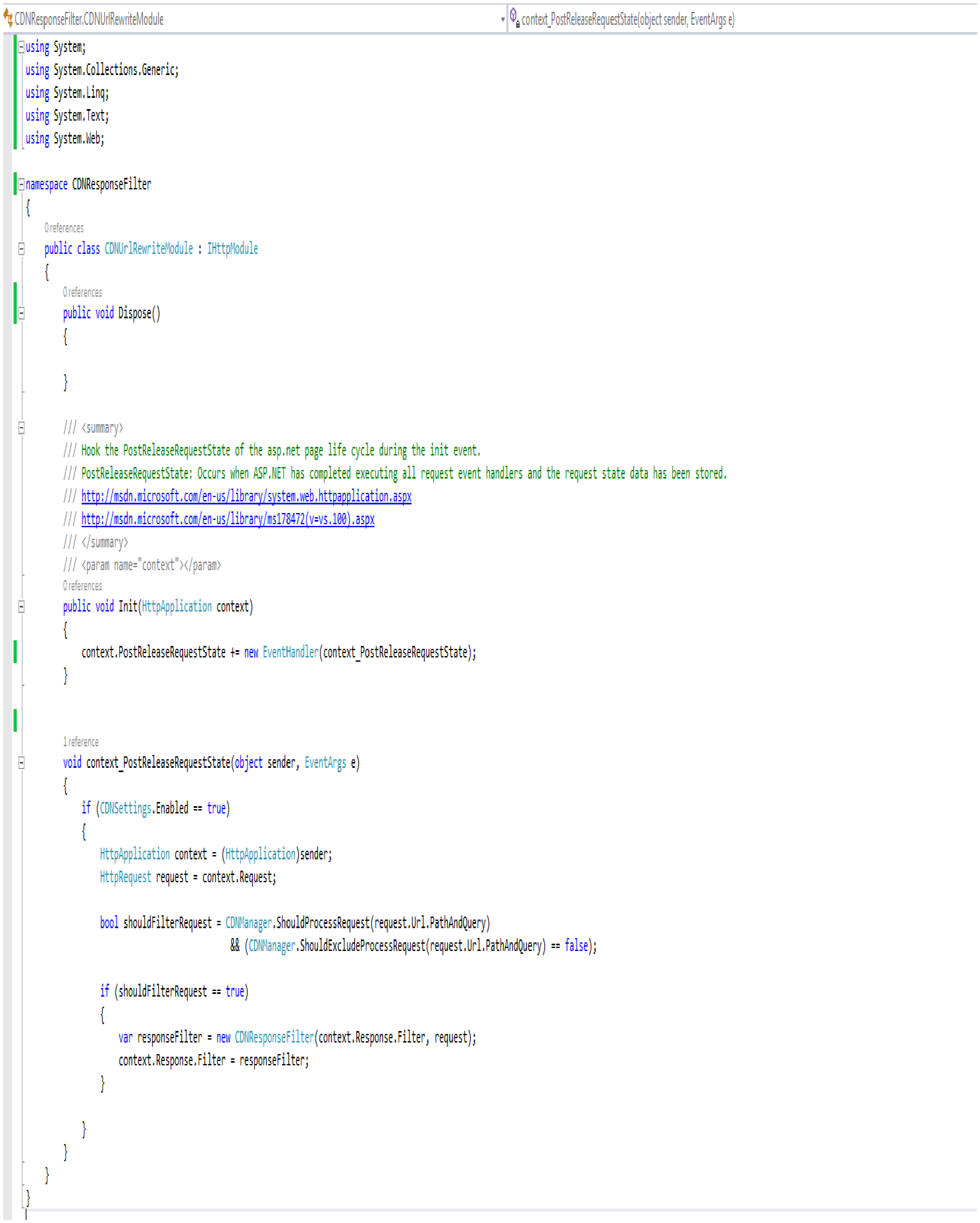
CNAME(DNS Change)	
Pros	**Cons**
Response time is faster because of shortest route between CDN edge and origin on all requests.	Transfer of SSL certificate increase the risk of Man-in-the-middle attack since certs are distributed to many data centers controlled by partners.
No need to write URL at the origin.	There is cost to deploy SSL certificates at CDN edge and the cost would multiply if you have multiple domains. SAN or wildcard SSL is an option to reduce the cost.
Performance increase: 25-30% and up.	CDN and DDOS protection at the same time can be implemented but the design would be complex. Plus, you end up with double proxy- one at CDN and another at DDOS.
	Must use Pure API to purge the contents or use control panel to purge the contents at the CDN when new codes are deployed at the origin.

I had the opportunity to evaluate both the architecture during the POC and the performance difference is negligible between the two architectures. URL Rewrite was the obvious winner since it gives you greater flexibilities and it is less costly.

POC to Real World Implementation:

Now that you have settled the URL Rewrite design, you would have to implement the solution for URL Rewrite. In my case, websites were running under IIS and they were powered by ASP.NET. So, I wrote a custom IHttpModule which intercepts the response filter stream, converts stream into html string, rewrite the static urls with CDN domain with version date query string and converts strings back to stream before sending to browser.

Acknowledgments:
https://github.com/NTTDATA/SitecoreCDN
http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx
http://msdn.microsoft.com/en-us/library/ms178472(v=vs.100).aspx
HtmlAgilityPack @ github



CDNResponseFilter.CDNUrlRewriteModule | context_PostReleaseRequestState(object sender, EventArgs e)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;

namespace CDNResponseFilter
{
    0 references
    public class CDNUrlRewriteModule : IHttpModule
    {
        0 references
        public void Dispose()
        {

        }

        /// <summary>
        /// Hook the PostReleaseRequestState of the asp.net page life cycle during the init event.
        /// PostReleaseRequestState: Occurs when ASP.NET has completed executing all request event handlers and the request state data has been stored.
        /// http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx
        /// http://msdn.microsoft.com/en-us/library/ms178472(v=vs.100).aspx
        /// </summary>
        /// <param name="context"></param>
        0 references
        public void Init(HttpApplication context)
        {
            context.PostReleaseRequestState += new EventHandler(context_PostReleaseRequestState);
        }


        1 reference
        void context_PostReleaseRequestState(object sender, EventArgs e)
        {
            if (CDNSettings.Enabled == true)
            {
                HttpApplication context = (HttpApplication)sender;
                HttpRequest request = context.Request;

                bool shouldFilterRequest = CDNManager.ShouldProcessRequest(request.Url.PathAndQuery)
                                            && (CDNManager.ShouldExcludeProcessRequest(request.Url.PathAndQuery) == false);

                if (shouldFilterRequest == true)
                {
                    var responseFilter = new CDNResponseFilter(context.Response.Filter, request);
                    context.Response.Filter = responseFilter;
                }

            }
        }
    }
}
```

Figure shows how to hook your url rewrite codes to asp.net page life cycle event.

Cache-Control in ASP.NET

Configure the ASP.NET application to specify max-age header (3 days in this reference).

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

Time to live (TTL) at CDN Edge

It’s very important for us to know how CDN calculates TTL in cache. CDN will refresh the resources from origin only when TTL expires. CDN will use “max-age” header if it is provided by the origin. If max-age is not provided by the origin, the default TTL is used. Different CDN provide may calculate default TTL differently.

Purging of contents from CDN

We know CDN improve the user experience by serving contents from the proximity (edge). However, it comes with the risk of content freshness and how efficiently we can keep the contents fresh at the edge as we deploy new resources at the origin (web servers). Most of the CDN providers would allow you to purge the contents from Control Panel. Alternatively, you can purge one or more resources via API.

If you rewrite the URLs with a query string (like test.css?vd=20130927102030) and CDN cache the url with query string, you don’t have to purge the contents upon deployment. Keep in mind, if you change the image referenced in the CSS file you must purge the image.

Disclaimer

This document is published solely for the purpose of illustration of CDN Implementation. The author does not represent or endorse any company or organization.