IV.1 Integrate caching and content delivery within solutions

jeudi 16 septembre 2021 14:31

Configure cache and expiration policies for Azure Redis Cache

Definition :
Azure Cache for Redis provides an **in-memory data store** based on the Redis software. Redis improves the performance and scalability of an application that uses backend data stores heavily. It's able to **process large volumes of application requests** by keeping frequently accessed data in the server memory, which can be written to and read from quickly. Redis brings a **critical low-latency and high-throughput data storage solution** to modern applications.

Azure Cache for Redis offers both the Redis open-source (OSS Redis) and a commercial product from Redis Labs (Redis Enterprise) as a managed service. It provides secure and dedicated Redis server instances and full Redis API compatibility. The service is operated by Microsoft, hosted on Azure, and usable by any application within or outside of Azure.

When to use it :

- Data cache
- Content cache
- Session store
- Job and message queuing
- Distributed transactions

Configuration

- **Name** : Unique within Azure
- **Location** : In the same region than the application
- **Pricing tier** :
  - Basic : 1 node, multiple cache sizes, for dev/test
  - Standard : replicated in 2 nodes, high-availability, SLA
  - Premium : better perf
- **Virtual network support** : Only premium tier, make the cache available to only other virtual machines and applications in the same virtual network.
- **Clustering support** : Only premium tier. Automatically split your dataset among multiple nodes.
- **DNS name**

Access the instance

Via a command-line tool to install :
COMMAND parameter1 parameter2 parameter3

Example : Set variable with cache expiration

```
SETEX myKey 1800 "some-value"
```

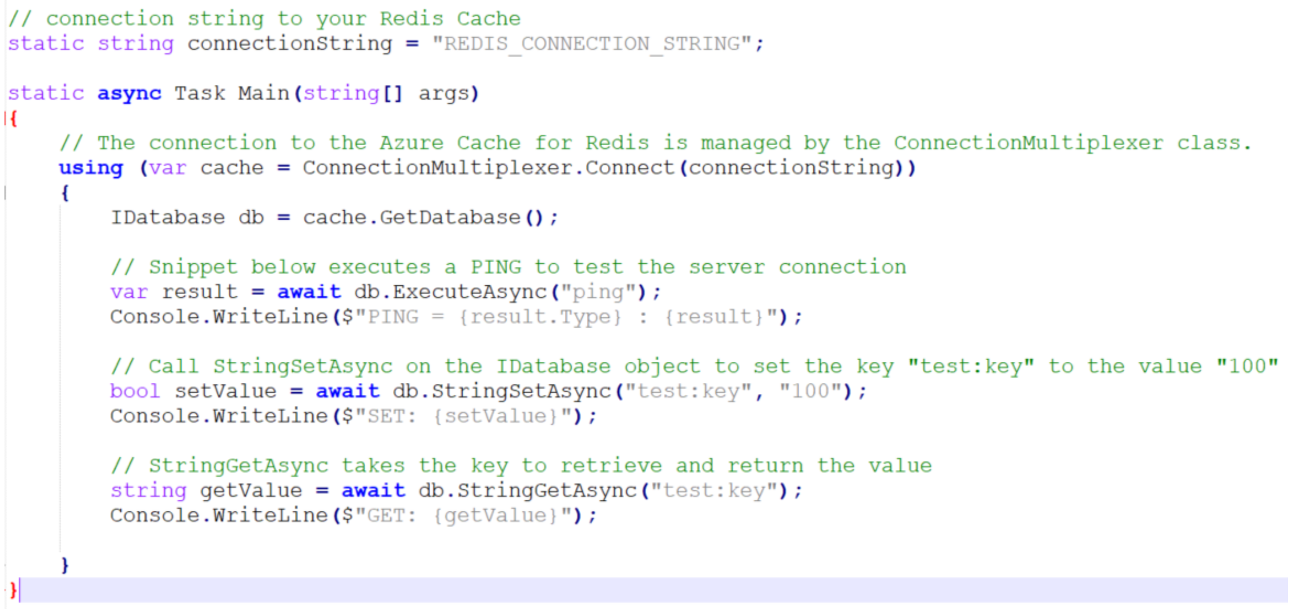
Or

```
SET myKey "some-value"
EXPIRE myKey 1800
```

Interact in .NET

```
// Create the cache in Azure
redisName=az204redis$RANDOM
az redis create --location <myLocation> \
    --resource-group az204-redis-rg \
    --name $redisName \
    --sku Basic --vm-size c0

using StackExchange.Redis;
using System.Threading.Tasks;
```



```
// connection string to your Redis Cache
static string connectionString = "REDIS_CONNECTION_STRING";

static async Task Main(string[] args)
{
    // The connection to the Azure Cache for Redis is managed by the ConnectionMultiplexer class.
    using (var cache = ConnectionMultiplexer.Connect(connectionString))
    {
        IDatabase db = cache.GetDatabase();

        // Snippet below executes a PING to test the server connection
        var result = await db.ExecuteAsync("ping");
        Console.WriteLine($"PING = {result.Type} : {result}");

        // Call StringSetAsync on the IDatabase object to set the key "test:key" to the value "100"
        bool setValue = await db.StringSetAsync("test:key", "100");
        Console.WriteLine($"SET: {setValue}");

        // StringGetAsync takes the key to retrieve and return the value
        string getValue = await db.StringGetAsync("test:key");
        Console.WriteLine($"GET: {getValue}");

    }
}
```

Eviction policy

The default policy for Azure Cache for Redis is **volatile-lru**, (lru = less recently used) which means that only keys that have a TTL value set are eligible for eviction. If no keys have a TTL value, then the system won't evict any keys. If you want the system to allow any key to be evicted if under memory pressure, then you may want to consider the **allkeys-lru** policy.

Implement secure and optimized application cache patterns including data sizing, connections, encryption, and expiration

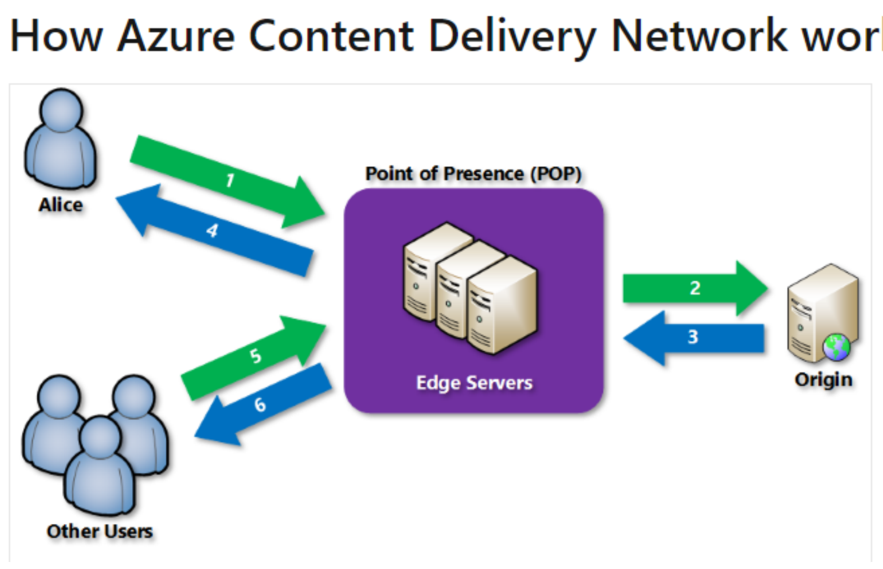
CDN Def :
Azure Content Delivery Network (CDN) offers developers a global solution for rapidly delivering high-bandwidth content to users by caching their content at strategically placed physical nodes across the world. Azure CDN can also accelerate dynamic content, which cannot be cached, by leveraging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

Why to use it :

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 it 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.

Caching behavior

- **Bypass cache**: Do not cache and ignore origin-provided cache-directive headers.
- **Override**: Ignore origin-provided cache duration; use the provided cache duration instead. This will not override cache-control: no-cache.
- **Set if missing**: Honor origin-provided cache-directive headers, if they exist; otherwise, use the provided cache duration.

3 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.

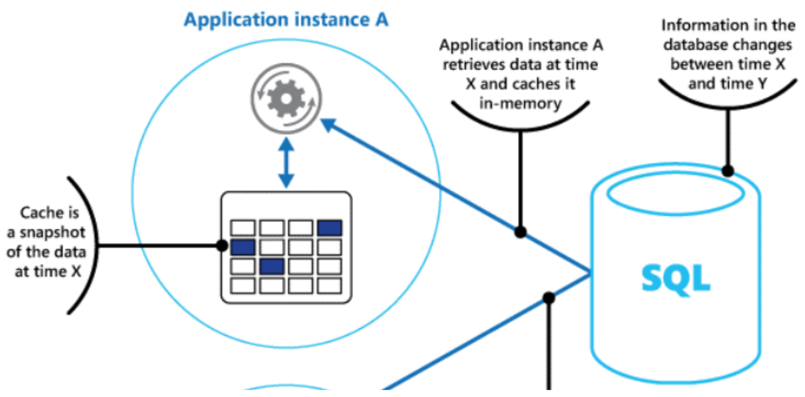
Time to live

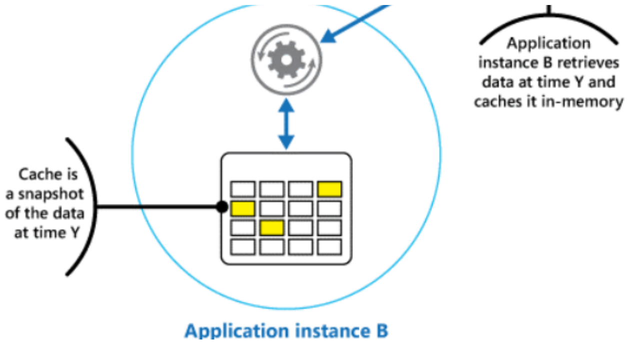
If you publish a website through Azure CDN, the files on that site are cached until their TTL expires. The **Cache-Control** header contained in the HTTP response from origin server determines the TTL duration.

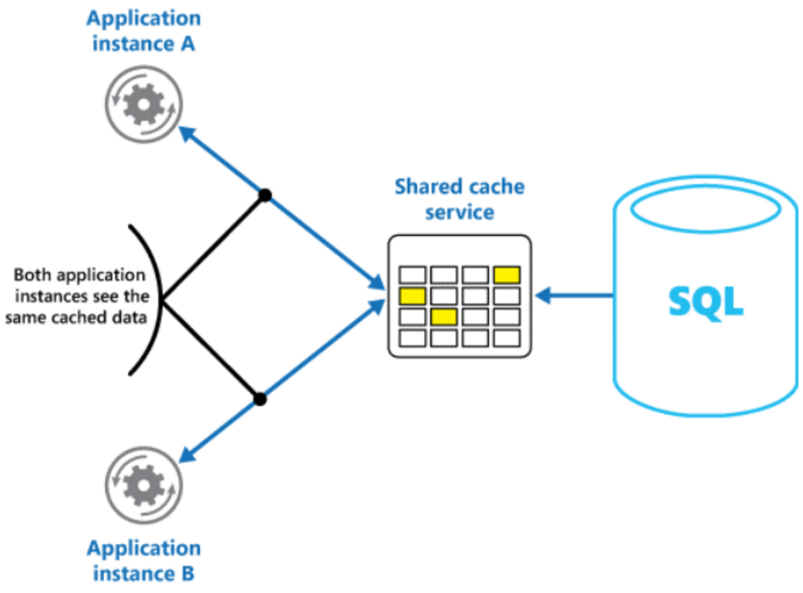
If you don't set a TTL on a file, Azure CDN sets a default value. However, this default may be overridden if you have set up caching rules in Azure. Default TTL values are as follows:

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

Private VS Shared caching


Application instance A
Application instance A retrieves data at time X and caches it in-memory
Information in the database changes between time X and time Y
Cache is a snapshot of the data at time X
SQL


Application instance B retrieves data at time Y and caches it in-memory
Cache is a snapshot of the data at time Y
Application instance B



Dev in .NET :

Create a CDN client

```
C#

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

List CDN profiles and endpoints

```
C#

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:");
        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

```
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);
    }
}
```

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, resourceGroupName);
    }
}
```

Purge an 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();
    }
}
```