Agenda: Redis Caching

- Caching Overview
- What is Redis Cache
- Creating a Redis Cache
- Developing Redis Cache Client
- Controlling Expiration

Caching Overview

Caching is a mechanism for:-

- Storing data close to where it is used than the original source.
- A mechanism used to speed up the system.
- A mechanism for decreasing load on a system component.

Caching is most effective when a client instance repeatedly reads the same data, especially if all the following conditions apply to the original data store:

- It remains relatively static.
- It's slow compared to the cache's speed.
- It's subject to a significant level of contention.
- It's far away when network latency can cause access to be slow.

When you consider Caching!

- Application should be able to withstand cached data unavailability.
- Do not store important information only in the cache.
- Balance the number of the objects versus the size of the objects.
- Use the minimal but complete set of key for caching.

Strategies when caching data:

- **Using a private cache**, where data is held locally on the computer that's running an instance of an application or service.
- **Using a shared cache**, serving as a common source which multiple processes and/or machines can access.

In both cases, caching can occur on the. The process that provides the user interface for a system, such as a web browser or desktop application, performs client-side caching,

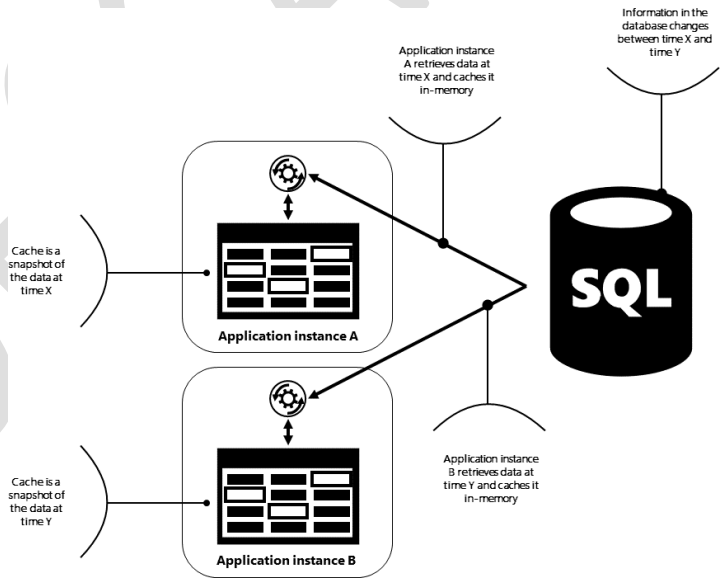
while the process **client-side and the server-side** that provides the business services that are running remotely performs the server-side caching.

Private caching

The most basic type of cache is an in-memory store. It's held in the address space of a single process and accessed directly by the code that runs in that process. This type of cache is very quick to access. It can also provide an extremely effective means for storing modest amounts of static data, since the size of a cache is typically constrained by the volume of memory that's available on the machine hosting the process.

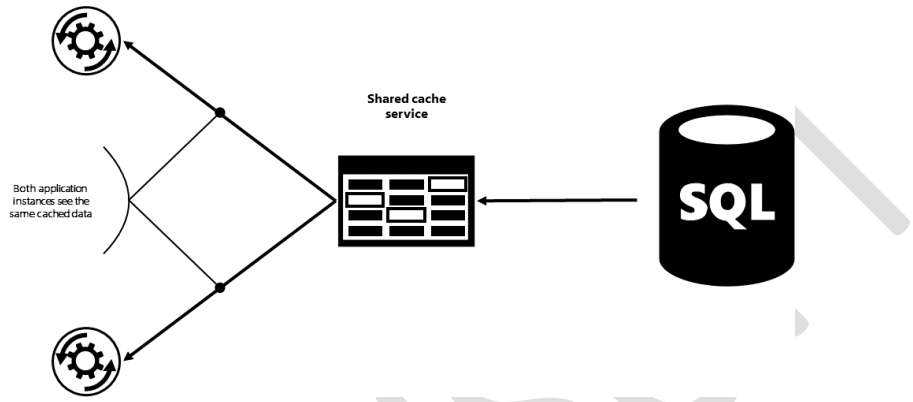
If you need to cache more information than is physically possible in memory, you can write cached data to the local file system. This will be slower to access than data that's held in-memory but should still be faster and more dependable than retrieving data across a network. If you have multiple instances of an application that uses this model running concurrently, each application instance has its own independent cache holding its own copy of the data.

Think of a cache as a snapshot of the original data at a point in the past. If this data is not static, it is likely that different application instances hold different versions of the data in their caches. Therefore, the same query performed by these instances can return different results.



Shared caching

Using a shared cache can help alleviate concerns that data might differ in each cache, which can occur with in-memory caching. Shared caching ensures that different application instances see the same view of cached data. It does this by locating the cache in a separate location, typically hosted as part of a separate service.



An important benefit of the shared caching approach is the scalability it provides. Many shared cache services are implemented by using a cluster of servers and utilize software that distributes the data across the cluster in a transparent manner. An application instance simply sends a request to the cache service. The underlying infrastructure is responsible for determining the location of the cached data in the cluster. You can easily scale the cache by adding more servers.

There are two main disadvantages of the shared caching approach:

- The cache is slower to access because it isn't held locally to each application instance.
- The requirement to implement a separate cache service might add complexity to the solution.

Azure Redis Cache

- Redis is an advanced **key-value store**, where keys can contain data structures such as strings, hashes, lists, sets and sorted sets. Redis supports a set of **atomic operations** on these data types.
- **Azure Redis Cache** is based on the popular open-source Redis cache. It gives you access to a secure, dedicated Redis cache, managed by Microsoft and accessible from any application within Azure.
- Azure Redis Cache uses Redis authentication and also supports SSL connections to Redis.
- Azure Redis Cache is easy to use. Just provision a cache using the Microsoft Azure portal and call into its end point using any client that supports Redis.

- Azure Redis Cache is easy to manage. You can also easily monitor the health and performance of your cache through the Azure portal.
- Also, you can have Microsoft manage replication of the cache for you, helping increase the availability of your cache data across cache failures.
- Redis offers many features beyond a traditional key-value pair cache, including manipulating existing entries with **atomic operations** such as the following capabilities:
  - Appending to a string.
  - Incrementing or decrementing a value.
  - Adding to a list.
  - Computing results or intersections and unions.
  - Working with sorted sets.

Azure Redis Cache is available in the following tiers:

- **Basic** – Single node, Multiple sizes, ideal for development/test and non-critical workloads. The basic tier has **no SLA**.
- **Standard** – A replicated cache in a two node Primary/Secondary configuration managed by Microsoft, with a high availability SLA.
- **Premium** – The new Premium tier includes a high availability SLA and all the Standard-tier features and more, such as better performance over Basic or Standard-tier Caches, bigger workloads, disaster recovery and enhanced security.

  Additional features of Premium includes:
  - **Redis persistence** allows you to persist data stored in Redis cache. You can also take snapshots and back up the data which you can load in case of a failure.
  - **Redis cluster** allows you to create caches larger than 53 GB and to share data across multiple Redis nodes.
  - **Azure Virtual Network (VNET)** provides enhanced security and isolation by restricting access to your cache to only those clients within the specified Azure Virtual Network.

Creating Redis Cache

To Create Redis Cache from portal follow below steps:

Click on New → Click on Database → Click on RedisCache → Provide DNS Name → Click Create

Using PowerShell:

New-AzureRedisCache –ResourceGroupName myGroup –Name myCache –Location “West US”

Developing Redis Cache Client

.NET applications can use the **StackExchange.Redis** cache client, which can be configured in Visual Studio using a NuGet package that simplifies the configuration of cache client applications.

Sample Client:

1. Get the Redis Connection String: Azure Portal → Redis Caches → Select the Cache Created → Settings → Access Keys → Copy to Notepad **Primary Connection String** to be used in code later.
2. Open Visual Studio 2017, and create a new console application called RedisCacheClient.
3. Add the **StackExchange.Redis** NuGet package to the application.
4. Add the Newtonsoft.**Json.net** NuGet package to the application.
5. Add the following to the Console Application.

```
using Newtonsoft.Json;
using StackExchange.Redis;
using System;

namespace RedisDemoApp
{
   public class Person
   {
      public string Name { get; set; }
      public int Age { get; set; }
   }
   class Program
   {
      static void Main(string[] args)
      {
         ConnectionMultiplexer connection =
ConnectionMultiplexer.Connect("dssdemocache.redis.cache.windows.net:6380,password=knes7fctaScr+VArB7SVlF =,ssl=True,abortConnect=False");
         IDatabase cache = connection.GetDatabase();
         cache.StringSet("stringValue", "my string");
         cache.StringAppend("stringValue", " - more added to my string");
         string entry = cache.StringGet("stringValue");
         Console.WriteLine(entry);
```

```
        RedisKey key = "intValue";
        RedisValue value = 10;
        cache.StringSet(key, value);
        cache.StringIncrement("intValue");
        value = (int)cache.StringGet("intValue");
        Console.WriteLine(value);

        Person person = new Person() { Age = 19, Name = "Foo1" };
        var serializedFoo = JsonConvert.SerializeObject(person);
        cache.StringSet("serializedFoo", serializedFoo);
        person = JsonConvert.DeserializeObject<Person>(cache.StringGet("serializedFoo"));
        Console.WriteLine(person.Name + " " + person.Age);

        cache.HashSet("user1", "S1", "v1");
        cache.HashSet("user1", "S2", "v2");
        cache.HashSet("user2", "S21", "v21");
        cache.HashSet("user2", "S22", "v22");
    }
  }
}
```

Please look at how cache.HashSet(…) and cache.HashGet(…) works

Look at the example for handling List: **https://stackoverflow.com/questions/31955977/how-to-store-list-element-in-redis-cache**

Note:

- Redis stores most data as Redis strings, but these strings can contain many types of data, including serialized binary data, which can be used when storing .NET objects in the cache. **You can add any non-primitive type to the cache when it is serialized. Use any of your favorite serializer, such as JSON format, and then store the result as a string with StringSet(). When you retrieve it, use the same serializer to de-serialize the value after using StringGet().**

- If **abortConnect** is set to false, it means that the call will succeed even if a connection to the Azure Redis Cache is not established. One key feature of **ConnectionMultiplexer** is that it will automatically restore connectivity to the cache once the network issue or other causes are resolved.

Controlling Expiration

You can control the expiration of any item you store in the cache by passing the parameter to StringSet(). When the string expires, it will be automatically removed from the cache.

The following example sets the cache expiry to 30 minutes:

```
IDatabase cache = connection.GetDatabase();
cache.StringAppend("expiringString", "this string will expire in 30 mins", TimeSpan.FromMinutes(30));
```

ASP.NET Output Cache Provider for Azure Redis Cache

1. Create an ASP.NET Application
2. Manage NuGet Package → Search **RedisOutputCacheProvider** → Add reference
3. The NuGet package downloads and adds the required assembly references and adds the <caching> section into your web.config file. Edit the settings as below:

   Note: host, port and accessKey can be taken from Redis Cache Properties (Azure Portal)

```
<caching>
  <outputCache defaultProvider="MyRedisOutputCache">
   <providers>
    <clear/>
    <add name="MyRedisOutputCache" type="Microsoft.Web.Redis.RedisOutputCacheProvider" port="6380"
host="dssdemocache.redis.cache.windows.net"  accessKey="knes7fctaScr+VArB7SVlF+lXJF0Y9kFLhyLkJOIWf4="
ssl="true" />
   </providers>
  </outputCache>
</caching>
```

4. Edit Index method of controller

```
[OutputCache(Duration=10,VaryByParam="id1,id2")]
public ActionResult Index(int id1, int id2)
{
  return View(DateTime.Now);
}
```

5. Add the following to Index.cshtml as below

@Model.ToString()

6. Run the application and refresh the page to note that time rendered doesn't change for 10 secs.

Note: If it's a WebForm Application

Add an OutputCache directive to each ASPX page for which you wish to cache the output.

<%@ OutputCache Duration="60" VaryByParam="*" %>

```
public class Repository<T>  : IDisposable where T:class
{
    DemoDbEntities db = ContextHelper.GetDataContext();// new DemoDbEntities();
    DbSet<T> ds;
    ObjectCache cache = MemoryCache.Default;
    public Repository()
    {
      ds = db.Set<T>();
    }
    public IEnumerable<T> GetAll()
    {
      Type tp = typeof(T);
      string type = tp.ToString();
      if (cache[type] == null)
      {
        cache[type] = ds.ToList();
      }
      return (IEnumerable<T>) cache[type];
    }
    public T GetById(int id)
    {
      return ds.Find(id);
    }

    public T Add(T entity)
    {
      ds.Add(entity);
      db.SaveChanges();
```

```
        Type tp = typeof(T);
        string type = tp.ToString();
        cache.Remove(type);
        //Welcome email
        return entity;
    }
    public void Update(T entity)
    {
        db.Entry<T>(entity).State = EntityState.Modified;
        db.SaveChanges();
        Type tp = typeof(T);
        string type = tp.ToString();
        cache.Remove(type);
    }
    public void Delete(int id)
    {
        T entity = ds.Find(id);
        ds.Remove(entity);
        db.SaveChanges();
        Type tp = typeof(T);
        string type = tp.ToString();
        cache.Remove(type);
    }
    public void Dispose()
    {
        db.Dispose();
    }
}
```

Using Redis Docker Image (Not Azure)

```
$ docker run --name some-redis -d redis
```