Which Asp.net Core Session Storage Strategies Developers Using?

A stateless protocol is used by web based applications. This is a HTTP protocol and so, there is no such place that can be used for storing data. This means for every new web request, browser opens the new HTTP connection. For

circumstances, where you have to store your data, asp.net core offers sessions for that. Asp.net core app maintains these sessions itself on the same server. Although asp.net core offers an in-memory session provider where all sessions are stored, there are times when the load needs to be balanced. In such cases, session storage strategies like distributed cache or sticky sessions are applied.

Why developers never prefer Sticky Sessions?

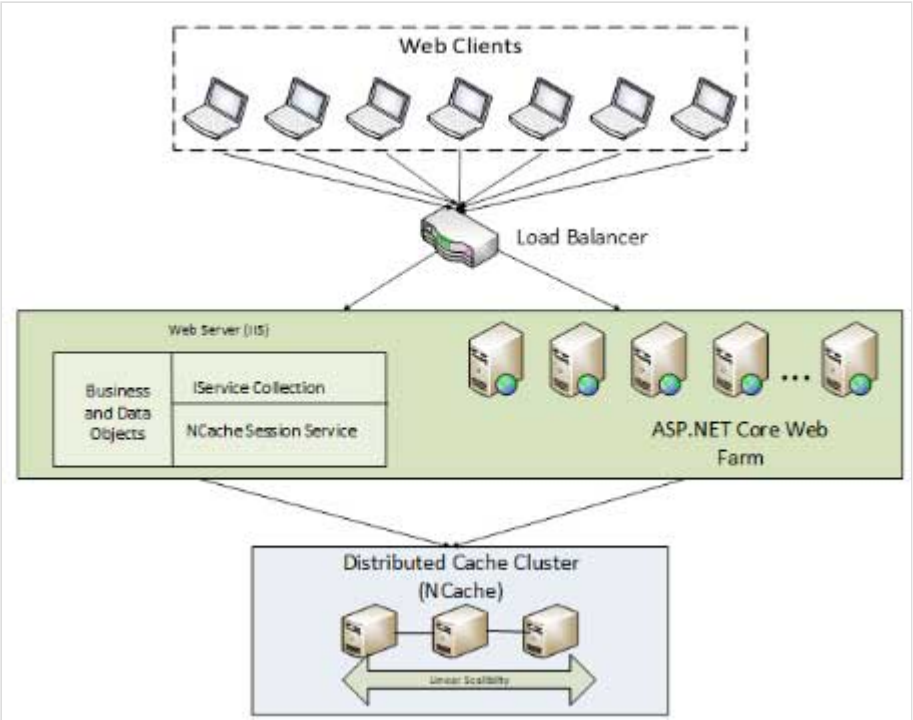
There is a reason to not to call Sticky sessions - because during the use of these sessions followed by the use of load balancer; all the requests made by users are transferred to a single unit server. The uneven load distribution is the only major drawback of using sticky sessions for asp.net core app.

There are times the requests increase to an extent that they overload the same server that maintains data completely. There is a risk of data loss if server goes down and this ultimately results in a single point of failure.

Storing Asp.net core sessions

All the issued encountered when applying sticky sessions are resolved with distributed cache. It is a type of cache store that is being applied by several app servers. This cache store is maintained as an external service that can be used for accessing and storing data. The key benefit of using this cache store type is that it works on scalability and enhances it along with the asp.net core app performance.

Here we are sharing an image that shows the working of distributed cache for storing sessions. Asp.net MVC Developers India can add multiple servers in the distributed cache cluster to make it scalable. Users may interact with asp.net core web farm via a load balancer. Distributed cache is further applied by the web farm to store session services provider.



IDistributedCache Interface

IDistributedCache interface is a Microsoft CRM product that is used for integrating a distributed cache in asp.net core apps. The interface allows interaction between the app and cache even if cache implementation is being involved. You can configure your session store provider via IDistributedCache with the help of following code-

```
namespace Microsoft.Extensions.Caching.Distributed
{
public interface IDistributedCache
{
// Each of these methods also has an “Async” overload
byte[] Get(string key);
void Refresh(string key);
void Remove(string key);
// Specify absolute & sliding expiration thru options
```

```
void Set(string key, byte[] value,
DistributedCacheEntryOptions options);
}
}
```

Important note – All of these methods have async overload too.

Application of IDistributedCache

Here we got a reference code that you may consider and understand the best possible way to use the IDistributed cache.

```
IDistributedCache _cache;
...
private byte[] LoadCustomer(string custId) {
    string key = "Customers:CustomerID:"+ custId;
    // is the customer in the cache?
    byte[] customer = _cache.Get(key);
    if(customer == null)
    {
       // the cache doesn't have it. so load from DB
       customer = LoadFromDB(key);
       // And, cache it for next time
       _cache.Set(key, customer);
    }
    return customer;
}
```

What are the limitations of IDistributedCache Interface?

There are some limitations of using IDistributedCache interface. There were two features omitted by asp.net core session which were earlier supported in Asp.net core session state.

These include-

- slideInUpByte [] for custom objects
- slideInUpSession locking

Other than this, there are limited cache features for your use.

NCache is one of the distributed caches that offer a user-friendly implementation for session storage. This thing saves time in coding your personal provider. You are also able to access different distributed cache features using NCache. It also handles limitations of IDistributedCache. This gives an internal session locking mechanism and supports custom objects.

Configuration

To configure session provider for distributed cache, you don't need much programming effort. NCache lets you to do that by either organizing all settings in Startup.cs class or simply by reading them via AppSettings.json file within the asp.net core applications.

Here is the reference code to explain configuration of distributed cache provider in the startup.cs class of the asp.net core app.

```
public void ConfigureServices(IServiceCollection services)
{
. . .
services.AddNCacheDistributedCache(configuration =>
{
configuration.CacheName = "myDistributedCache";
configuration.EnableLogs = true;
configuration.ExceptionsEnabled = true;
});
}
```

Once the configuration of NCache for storing sessions is done successfully, you can simply use it with your asp.net core app for Read and Write purposes or operations.

There is minimum coding required for configuring NCache session provider for asp.net core apps. There is no need to write lengthy code snippets- it can be easily achieved with simple codes. You can also avail several benefits from such configuration, such as-

- Linearly scalable
- High availability
- Intelligent session replication
- Quick Dynamic Compact serialization

When you work with NCache, you will find that it is not just fast but scalable too. It is purely native .net, which makes it perfect fit for your asp.net core application.

If you have any doubt about NCache or asp.net core application, you can anytime get in touch with the field professionals. They will guide you through and make you understand the things about the same.

Project Inquiry:
info@aegissofttech.com

Business Inquiry:
hs@aegissofttech.com

Call Us:
+1.646.971.0799

Aegis Associate Company: