NCache™

Case Study

> After an easy migration to NCache, it has begun to act like the glue that holds our application together. And the various features it offers have been a real bonus, especially the queues, that we have used excessively.
>
> Enterprise Application Support Analyst

European Defense Department

NCache allows this organization to maintain high availability and accelerate the performance of their Task Management Application.

> *"After an easy migration to NCache, it has begun to act like the glue that holds our application together. And the various features it offers have been a real bonus, especially the queues, that we have used excessively."*

- Enterprise Application Support Analyst

Country or Region:
Europe

Industry:
Military Defense

Company Profile:
This is a European defense department that works to create a connection between several nations, allowing them to consult and collaborate on security projects while coordinating international crisis management activities.

Introduction

Such organizations demand high availability, data consistency, and easy task management. This organization in particular has designed a tracking application for the use of their employees to manage information and tasks between different departments. The employees log in to the application and enter their daily tasks. Each relevant department can supervise employees and check their task statuses. The firm previously used Redis for their development processes but faced limitations in deploying it to their servers. Additionally, as a leading contributor to peace on the international stage, they wanted to enhance their tracking application to be notified about any changes occurring in real-time.

Application Pain Points

They require a good caching mechanism. They evaluated Redis. At an early stage, they found out that Redis was not fulfilling their data structure requirements meaning they could not deploy Redis in their HQ. Additionally, migration to Redis wasn't as easy as one might have thought, Redis wasn't fulfilling the organization's technical requirements. Redis is developed in C++ on Linux. And, it is not officially supported on Windows. As the organization's application has been built using .NET 6, integrating a product that is not .NET native can cause issues.

How NCache Fits into Their Solution?

NCache provides them with a flexible and scalable in-memory solution for their single-tiered application along with its advanced features. They are experiencing faster data change notifications to process and work on it. "*It was an easy migration from Redis to NCache and it was pretty straightforward*", said the Enterprise Application Support Analyst of the team.

Whatever their application seems to lack, NCache provided the optimal solutions as the fix.

Unplanned changes or planned maintenance will not affect the application's performance as it is using replicated cache topology - if a node is down, the other node makes up for it. The organization's data structure requirements were fulfilled by NCache Queues which are very intuitive and easy to implement. From the SignalR backplane to Cache Loader, backing source provider (Read-Through), and NCache locks; the tracking application benefits from all these powerful features of NCache.

Moreover, NCache is a .NET native distributed caching solution that can be deployed on Linux, as well as Windows. This helped them with their deployment, as Redis only supports Linux. Most .NET applications run on Windows and now .NET Core applications can run on both Windows and Linux. Therefore, it has become essential to ensure that their distributed caches also provide support for both these operating systems. And NCache as one of the top caching solutions, provides this support.

Alachisoft Product:

NCache

Customer Needs Met:

- Easy & Straightforward Migration /Deployment
- Better Data Structures
- Scalability
- Reduced Database Trips

Application Architectural Overview and Deployment with NCache

The application architecture mainly comprises the following components:

- **Front-End:**

The front-end is designed using the SVELTE JS frame.

- **Logic Layer:**

The logic layer interacts with the database using the Rest APIs. It uses .NET Core 6 Web APIs, has SignalR Notification Hubs, and comes with .NET APIs for SearchService and collaboration services.

- **Back-End:**

The back-end is based on SQL Server as the data store. Data sources include SQL Server, ADFS, Sharepoint, and Document Management Server. The Backing source (NCache Read-Through) is implemented in this layer which uses .NET 6 with NCache 5.3 SP1.

For a closer look at the application, the Web API contains the Action service, Notification Hub along with SignalR. NCache acts as a central hub between different parts of the application. The user logs in using ADFS (Active Directory Federation Services), goes to the front-end and

calls the Web API. The Web API is responsible for creating this action and putting it in the queue inside the NCache. The Action Service performs multiple operations such as polling the queue, processing tasker object, etc.

If there is an item present inside the Action Service, it will process it. As part of the processing, the Action Service also sends the notifications via a Notification Hub. This Action Service changes the state of the data object “Tasker”- in case of a change of data state where new data may be added or previously added data is updated. This helps the users to analyze the processing and it sends out notifications to all the concerned users affected by that change. The application also has a collaboration service to manage tasks using SharePoint.

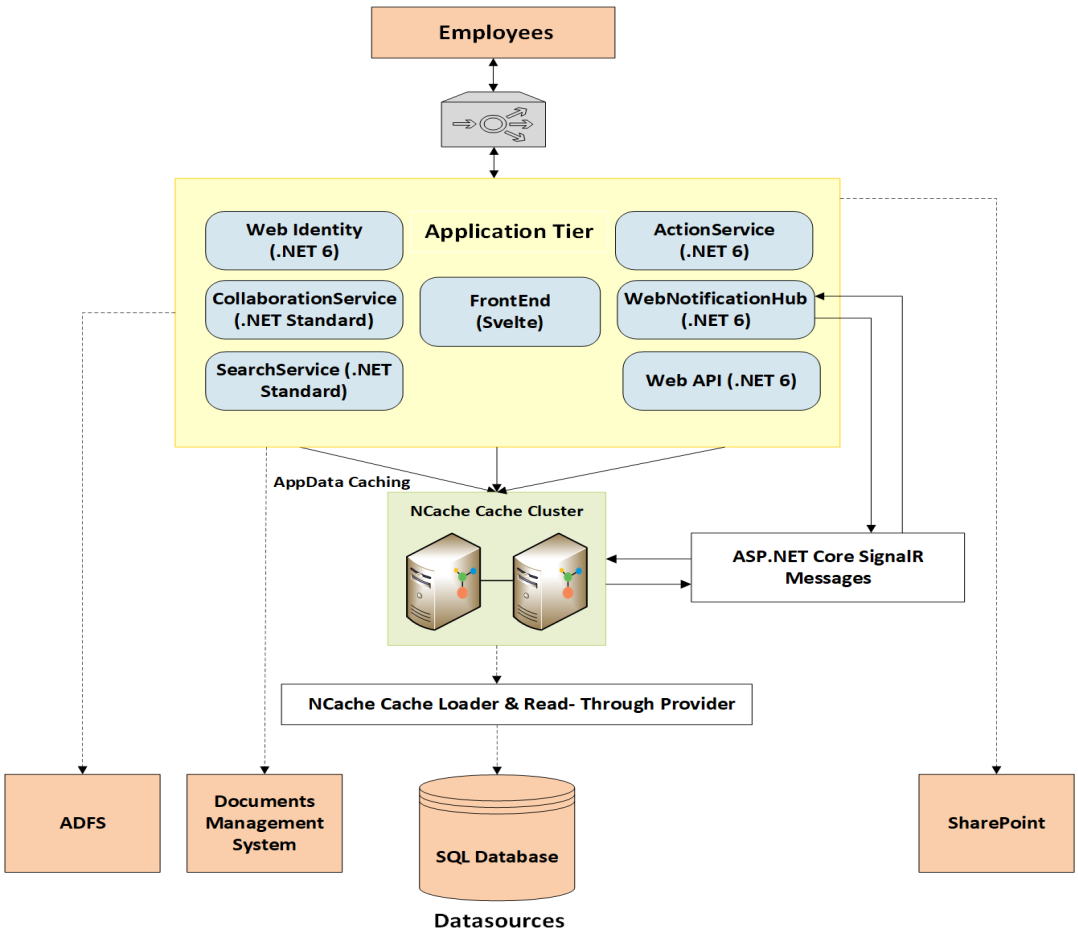


Figure 1 - Architectural Diagram using NCache

Currently, the application uses 2 load-balanced servers hosting the Front end, Logic layer, and NCache. Additionally, there is 1 database server to support the servers. They have 1 general cache where they store queues

that also caches some reference data. They also have 2 other caches for storing static data, i.e., “tasks”. The general cache and the static caches both have a Backing Source (Read-Through) and Loader implemented.

SignalR Backplane

The organization extends the application’s use with the SignalR back-plane. When multiple users are using the web app, they wait for feedback on the tasker object. The feedback is provided to them via a Notification Hub that uses SignalR. This has eliminated the need to refresh the web-page every time to request new messages.

Real-time ASP.NET web applications can be created with SignalR, where the server broadcasts updates to all logged-in users as soon as an update is triggered. By doing this, the response time for user requests for updates is reduced. NCache offers support for SignalR by providing an extension to the SignalR provider. The provider has records of all of the application's concerned web servers.

Queue Data Structure for Consistent Data

They have been excessively using the Queue data structure provided by NCache. Queues are stored in the general cache to maintain information being processed at runtime. The Web API adds a new item to the queue. Then Action Service polls that queue and if a new item is found then it performs required processes. The entire Queue is stored in NCache. Whenever they had to fetch an item, they don’t have to traverse the entire Queue as that is not only a slow but also a costly process. NCache uses referencing to get the required item that doesn’t hurt performance.

The Queue data structure comes with different APIs like GetQueue, which checks whether an item exists or not in the cache with the help of the Contains method. Similarly, getting the topmost item from a queue, copying an entire source queue to the one-dimensional array, removing items from the queue, registering events (key-based and data structures), locking and unlocking the queues for data consistency- all these are the other features that NCache offers with the Queue data structure.

Locking Queues for Data Integrity

The organization also brings in the use of locking offered by NCache. For instance, they may lock the queue whenever the user is reading or writing

About Alachisoft:

Alachisoft provides a popular high performance in-memory distributed cache called NCache. NCache is an Open Source middleware that runs in production environment and boosts performance and scalability of .NET web apps, SOA service apps, and general high traffic server apps. NCache has a 13 year proven track record with hundreds of customers all over the world and specially in US, UK, and Western Europe.

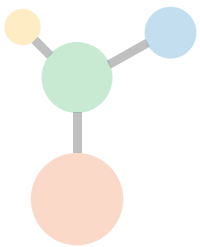
Visit our website at
www.alachisoft.com

You can download a free 60 day fully working trial of NCache from here:
www.alachisoft.com/ncache

Alachisoft
Corporate Headquarters
12005 Ford Road, Suite 520
Dallas, TX 75234

US: +1 (214) 764-6933
UK: +44 207 993-8327
Fax: +1 (925) 886 8361

Sales Email:
sales@alachisoft.com
Technical Support:
support@alachisoft.com



to it, let's say for about 20 seconds using the Lock API so that data integrity is maintained. If the required queue is free and ready to be in use again, the user unlocks it using the Unlock API. When we are talking about concurrent updates, NCache's locking feature is ready to serve and provide data integrity and data consistency.

Backing Source (Read-Through Caching)

They used the Read-Through provider implemented in the application's general, as well as static caches store - to fetch the data from the data source in case it is not present in the cache. This minimizes the additional network trips to the database. NCache offers a Read-Through provider that enables the users to communicate with the data source. In Read-Through Caching, in the event of a cache miss, NCache will contact the provider to load data behind the get call.

Cache Loader for Pre-Populating the Cache

NCache provides another powerful feature, the Cache Startup Loader, that they have been using in their tracking application. The application pre-populates the data in the cache on startup. This not only saves network costs but also makes the data highly available. The Cache Loader's purpose is to avoid latency even on the initial request, but since the Cache Loader runs on another service, it is technically asynchronous. The tracking application always keeps the most frequently used data in the cache on startup. This feature has helped improve the application performance, along with handling the request throughput.

Moving Forward with NCache

The organization is benefitting from NCache in numerous areas as explained above. As part of their future growth, they are planning to use NCache's Groups feature, along with its Event Notifications, and Pub/-Sub messaging, as the Action Service has to poll every time to check for updates. NCache is able to manage all this for them without lacking performance.