Azure Redis Cache

vs.

NCache

Feature Level Comparison

For .NET Applications

Microsoft Azure Redis Cache v3.2.7 vs. NCache 4.8

Please note that this comparison is not against the general Open Source Redis v4.0.1 (download that comparison separately) but against Redis v3.2.7 that is being used by Microsoft Azure Redis Cache. Read this comparison to:

- Understand Azure Redis Cache and NCache major feature differences
- See how Azure Redis Cache and NCache compare on qualitative aspects such as performance, scalability, high availability, data reliability, and administration.

Table of Content

Disclaimer 1
1 Executive Summary 2
2 Qualitative Differences Explained 6
2.1 .NET Platform Support 6
2.2 Performance and Scalability 7
2.3 Cache Elasticity (High Availability) 9
2.4 Cache Topologies 12
2.5 WAN Replication 15
2.6 ASP.NET & ASP.NET Core Support 16
2.7 Object Caching Features 18
2.8 Managing Data Relationships in Cache 20
2.9 Cache Synchronization with Database 21
2.10 Event Driven Data Sharing 23
2.11 SQL-Like Cache Search 25
2.12 Data Grouping 25
2.13 Read-through, Write-through & Cache Loader 27
2.14 Big Data Processing 28
2.15 Third Party Integrations 29
2.16 Security & Encryption 30
2.17 Cache Size Management (Evictions Policies) 31
2.18 Cache Administration 32
3 Conclusion 34

Disclaimer

The comparison provided in this document is for the purpose of helping you get a better understanding of Azure Redis Cache versus NCache. Information obtained about Azure Redis Cache is from the freely available downloads, documents, and forums.

We did not conduct any scientific benchmarks for performance and scalability of Azure Redis Cache so our assessment about it may be different from yours. NCache benchmarks are already published on our website (http://www.alachisoft.com) for you to see.

Additionally, we have made a conscious effort to be objective, honest, and accurate in our assessments in this document. But, any information about Azure Redis Cache could be unintentionally incorrect or missing, and we do not take any responsibility for it.

Instead, we strongly recommend that you do your own comparison of NCache with Azure Redis Cache and arrive at your own conclusions. We also encourage you to do performance benchmarks of Azure Redis Cache and NCache both in your environment for the same purpose.

1 Executive Summary

This document compares Azure Redis Cache with NCache and contrasts their significant differences. This comparison focuses on all the major areas that a good in-memory distributed cache should provide.

Feature	Azure Redis Cache	NCache
.NET Platform Support		
- .NET Cache Client	Partial support	Supported
- .NET Core Cache Client	Partial support	Supported
- .NET NuGet Packages	Partial support	Supported
- .NET Cache Server	Not Supported	Supported
- .NET Core Cache Server	Not Supported	Coming Soon
- .NET Server-Side Code	Not Supported	Supported
- .NET Core Server-Side Code	Not Supported	Coming Soon
Performance and Scalability		
- Cache Performance	*Please verify yourself*	Super Fast
- Cache Scalability	*Please verify yourself*	Extremely Scalable
- Bulk Operations	Partial support	Supported
- Async Operations	Supported	Supported
- Compression	Not Supported	Supported
- Fast Compact Serialization	Not Supported	Supported
- Indexes	Not Supported	Supported
- Multiple NIC Binding	Not Supported	Supported
Cache Elasticity (High Availability)		
- Dynamic Cache Cluster	Partial support	Supported
- Peer to Peer Architecture	Not Supported	Supported
- Connection Failover	Partial support	Supported
- Dynamic Configuration	Partial support	Supported
- Multiple Clusters	Supported	Supported
- Named Caches	Supported	Supported
- Cluster Specific Events	Not Supported	Supported
- Split Brain	Not Supported	Partial support
Cache Topologies		
- Local Cache	Partial support	Supported
- Client Cache (Near Cache)	Not Supported	Supported

Feature	Azure Redis Cache	NCache
- Mirrored Cache	Supported	Supported
- Replicated Cache	Not Supported	Supported
- Partitioned Cache	Partial support	Supported
- Partitioned-Replica Cache	Supported	Supported
- Partitioned Data Balancing	Partial support	Supported
- Load Balancing	Supported	Supported
- Partitioned Data Affinity	Supported	Not supported
- Persistence	Supported	Supported
WAN Replication		
- Active – Passive	Not Supported	Supported
- Active – Active	Not Supported	Supported
- Conflict Resolution	Not Supported	Supported
- De-duplication	Not Supported	Supported
- Data Security	Not Supported	Supported
ASP.NET & ASP.NET Core Support		
- ASP.NET Core Session Persistence (basic)	Supported	Supported
- ASP.NET Core Session Persistence (advanced)	Not Supported	Supported
- ASP.NET Core Sessions (Multiple-site)	Not Supported	Supported
- ASP.NET Session State Caching (basic)	Supported	Supported
- ASP.NET Session State Caching (advanced)	Not Supported	Supported
- ASP.NET Sessions State (Multiple-site)	Not Supported	Supported
- ASP.NET View State Cache	Not Supported	Supported
- ASP.NET Output Cache	Supported	Supported
Object Caching Features		
- Get, Add, Insert, Remove, Exists, Clear Cache	Supported	Supported
- Expirations	Partial support	Supported
- Lock & Unlock	Supported	Supported
- Streaming API	Supported	Supported
- Transactions	Partial support	Partial support
- Data Portability	Not Supported	Supported
- Item Versioning	Not Supported	Supported
- Multiple Object Versions	Not Supported	Supported
Managing Data Relationships in Cache		

Feature	Azure Redis Cache	NCache
- Key Based Relationships	Not Supported	Supported
- Key Based Relationships Across Caches	Not Supported	Supported
Cache Synchronization with Database		
- SQL Dependency (SQL Server)	Not Supported	Supported
- Oracle Dependency (Oracle)	Not Supported	Supported
- Db Dependency (Any DB)	Not Supported	Supported
- File Dependency	Not Supported	Supported
- Custom Dependency (Any Source)	Not Supported	Supported
Event Driven Data Sharing		
- Item Level Events (onInsert / onRemove)	Supported	Supported
- Cache Level Events (Add/Insert/Remove)	Supported	Supported
- Custom Events (Fired by Apps)	Not Supported	Supported
- Continuous Query	Not Supported	Supported
- Topic (for Pub/Sub Messaging)	Supported	Supported
- Queue (for Pub/Sub Messaging)	Supported	Not Supported
SQL-Like Cache Search		
- SQL Search	Not Supported	Supported
- LINQ Queries	Not Supported	Supported
- SQL on Tags, Named Tags & Groups	Not Supported	Supported
Data Grouping		
- Groups/Subgroups	Not Supported	Supported
- Tags	Not Supported	Supported
- Named Tags	Not Supported	Supported
Read-through, Write-through, Cache Loader		
- Read-through	Not Supported	Supported
- Write-through & Write behind	Not Supported	Supported
- Auto Reload at Expiration & Database Synchronization	Not Supported	Supported
- Cache Startup Loader	Partial support	Supported
Big Data Processing		
- Map-Reduce Query	Not Supported	Supported

Feature	Azure Redis Cache	NCache
- Aggregators	Not Supported	Supported
- Entry Processor	Not Supported	Supported
Third Party Integrations		
- NHibernate 2nd Level Cache	No Official Support	Supported
- Entity Framework Core Cache (Extension Methods)	Not Supported	Supported
- Entity Framework Cache	Not Supported	Supported
- Memcached Protocol Server	Supported	Supported
- Memcached Smart Wrapper	Not Supported	Supported
Security & Encryption		
- Authentication (Active Directory/LDAP)	Partial support	Supported
- Authorization	Supported	Supported
- Data Encryption	Not Supported	Supported
- Secure Communication	Partial support	Supported
Cache Size Management (Evictions Policies)		
- Max Cache Size (in MBs)	Supported	Supported
- LRU Evictions (Least Recently Used)	Supported	Supported
- LFU Evictions (Least Frequently Used)	Not Supported	Supported
- Priority Evictions	Not Supported	Supported
- Do Not Evict Option	Supported	Supported
Cache Administration		
- Admin Tool (GUI)	Not Supported	Supported
- Monitoring Tool (GUI)	Not Supported	Supported
- PerfMon Counters	Not Supported	Supported
- Admin Tools (PowerShell)	Not Supported	Supported
- Admin Tools (Command Line)	Supported	Supported
- Administration and Monitoring (API)	Supported	Supported

2 Qualitative Differences Explained

2.1 .NET Platform Support

For .NET applications, it is important that your distributed cache is also native to .NET so your entire application stack is .NET. Otherwise, it unnecessarily complicates things for your development, testing, and deployment.

This section describes how Redis and NCache support .NET platform.

Feature Area	Azure Redis Cache	NCache
.NET Cache Client	Partial support Not officially supported. Third party .NET client exists.	Supported .NET Client is officially supported.
.NET Core Cache Client	Partial support Not officially supported. Third party .NET Core client exists.	Supported .NET Core Client is officially supported.
.NET NuGet Packages	Partial support Not officially supported. Third party NuGet packages exists.	Supported Full set of NuGet packages provided.
.NET Cache Server	Not Supported Redis server is written in C and usually supported on Linux. Even Azure Redis is deployed on Linux.	Supported NCache server is native .NET.
.NET Core Cache Server	Not Supported Redis server is written in C and usually supported on Linux. Even Azure Redis is deployed on Linux.	Coming Soon NCache server will soon be released in .NET Core.
.NET Server-Side Code	Not Supported Redis actually doesn't even support server-side code, let alone supporting it in .NET.	Supported Develop all server-side code like Read-through, Write-through, Write-behind, Cache Loader, Custom Dependency, and more in .NET.
.NET Core Server-Side Code	Not Supported Redis actually doesn't even support server-side code, let alone supporting it in .NET Core.	Coming Soon Once .NET Core NCache Server is released, you can then develop all server-side code like Read-through,

Feature Area	Azure Redis Cache	NCache
		Write-through, Write-behind, Cache Loader, Custom Dependency, and more in .NET Core.

2.2 Performance and Scalability

Performance is defined as how fast cache operations are performed at a normal transaction load. Scalability is defined as how fast the same cache operations are performed under higher and higher transaction loads. NCache is extremely fast and scalable.

See NCache benchmarks at Performance and Scalability Benchmarks.

Feature Area	Azure Redis Cache	NCache
Cache Performance	*Please verify yourself* But, please note that Redis converts everything into string for storage. And, this can be very costly for images, binary data, or large objects.	Super Fast NCache is extremely fast. Please see its performance benchmarks.
Cache Scalability	*Please verify yourself*	Extremely Scalable NCache provides linear scalability, means as you add more nodes to the cluster your performance increases in a linear fashion. Please see its performance benchmarks.
Bulk Operations	Partial support Bulk operations are not distributed to all the nodes. Instead, they're all sent to one shard and all the keys must be present on that node for success.	Supported Bulk Get, Add, Insert, and Remove. This covers most of the major cache operations and provides a great performance boost.
Async Operations	Supported	Supported Async add, insert, and remove provided. Async operation returns control to the application and performs the cache operation in the background. Improves application response time greatly.

Feature Area	Azure Redis Cache	NCache
Compression	Not Supported	Supported Specify this along with item size threshold and only items larger than the threshold are compressed. Rest are cached uncompressed. This is provided because compressing smaller items often slows things down. And, you can configure "compression" at runtime through "Hot Apply".
Fast Compact Serialization	Not Supported In fact, Redis stores all data as string. So, if you need to store images or objects, they get transformed into a string first and then stored. And, this is quite costly as compared to binary serialized objects that NCache stores.	Supported Compact Serialization is extremely fast because it uses precompiled code to serialize and also because it stores type-ids instead of long type names in the serialized objects. This is almost 10 times faster. Once you register classes for Compact Serialization, NCache generates serialization code and compiles it in-memory all at runtime and uses this precompiled code for serialization. You can mix Compact Serialization with regular serialization on objects of your choice.
Indexes	Not Supported	Supported NCache allows you to define indexes on object attributes. NCache then generates data extraction code for these indexes at connection time, compiles it in-memory, and uses it at client-side for all data extraction. This is much faster than using Reflection. NCache also creates indexes automatically on Tags, Named Tags, Groups, and Subgroups. Expiration and Eviction Policies.

Feature Area	Azure Redis Cache	NCache
Multiple NIC Binding	Not Supported You can bind multiple IPs but cannot define which to use for cluster replication. Multiple binding is only for accessing Redis through different networks.	Supported You can assign two NICs to a cache server. One can be used for clients to talk to the cache server and second for multiple cache servers in the cluster to talk to each other. This improves your bandwidth scalability greatly. You can also assign a specific NIC for a cache client to use for talking to the cache server.

2.3 Cache Elasticity (High Availability)

Cache elasticity means how flexible is the cache at runtime. Are you able to perform the following operations at runtime without stopping the cache or your application?

1. Add or remove any cache servers at runtime without stopping the cache.
2. Make cache config changes without stopping the cache
3. Add or remove web/application servers without stopping the cache
4. Have failover support in case any server goes down (meaning are cache clients are able to continue working seamlessly).

This is an area where Redis is relatively weak. In fact, it doesn't provide support for some of these things. But, NCache is known for its strength in this area.

NCache provides a self-healing dynamic cache clustering that makes NCache highly elastic. Read more about it at Self-Healing Dynamic Clustering.

Feature Area	Azure Redis Cache	NCache
Dynamic Cache Cluster	Partial support Redis clusters with shards are rigid. If a Master without a Slave fails, the cluster becomes unusable because the slots from this shard are not picked up by other Masters. And, if a Master that has a slave goes down, the slave does take over. But it doesn't automatically rebalance (re-shard) itself.	Supported NCache is highly dynamic and simple to manage. A shard in NCache is a partition. And, a partition can also have a replica but always on separate server. And, similar to Redis, if a cache server goes down, its replica automatically takes over. But, unlike Redis, NCache replica

Feature Area	Azure Redis Cache	NCache
	And, if this Slave also goes down before rebalancing (re-sharding), you not only lose data but the cluster again enters an unusable state because other Masters don't pick up the slots from this shard. And, all rebalancing must be done manually and also requires you to specify which "slots" or buckets to rebalance. This also increases complexity for managing Redis.	automatically rebalances (re-shards) and merges itself into other partitions and all the partitions ensure they have corresponding replicas. This way, NCache is not vulnerable to data loss except during the rebalancing (state transfer) which is quite fast.
Peer to Peer Architecture	Not Supported Redis uses the Master/Slave concept which is more rigid than peer-to-peer. If a Master without a Slave goes down, the cluster becomes unusable since this Master's shard is not automatically rebalanced (re-sharded) to other Masters.	Supported NCache cache cluster has a peer to peer architecture. This means there is no "master/slave" and no "majority rule" in the cluster. All nodes are equal. There is a "coordinator" node that is the senior most node. If it goes down, next senior most node takes over this role automatically. This means if any server goes down, the cluster always remains functional and correct (even if there is data loss due to not having replicas).
Connection Failover	Partial support In Redis, if a shard with no replicas goes down, the entire cluster is halted and blocks any client requests. If a Redis shard node with a replica goes down, the replica automatically takes over. But, if later the replica also goes down (before any manual rebalancing), then the client	Supported NCache provides full connection failover support between cache clients and servers and also within the cache cluster. In case of cache server failure, NCache clients continue working with other servers in the cluster and without any interruption. Cluster auto-manages itself by rebalancing its data and recreating replicas where needed.
Dynamic Configuration	Partial support Although, Redis provides CONFIG SET	Supported NCache cluster configuration is not hard

Feature Area	Azure Redis Cache	NCache
	command, you have to separately run it against every server whereas in NCache, you can do this against the entire cluster at once.	coded and when you add or drop servers at runtime, all other servers in the cluster are made aware of it. NCache clients also learn about all the servers and a variety of other configuration at runtime from the cache cluster. Also, 'Hot Apply' feature allows you to change a lof of the configuration at runtime without stopping anything.
Multiple Clusters	Supported	Supported NCache allows you to create multiple cache clusters of either the same or different topologies on the same set of cache servers.
Named Caches	Supported	Supported NCache allows you to create multiple named caches on the same set of cache servers.
Cluster Specific Events	Not Supported Cluster Events are not supported. Redis only allows you to manually query about the cluster health.	Supported NCache provides events about changes in the cluster like: MemberJoined, MemberLeft, CacheStopped, etc. These events can be delivered to both .NET and Java applications natively.
Split Brain	Not Supported	Partial support Split brain detection is provided and you're notified through NCache events when that happens. But no auto recovery is provided yet. It is going to be added in the next version.

2.4 Cache Topologies

Cache Topologies determine data storage, data replication, and client connection strategy. There are different topologies for different type of use cases. So, it is best to have a cache that offers a rich variety of cache topologies.

Read more details on NCache caching topologies at in-memory distributed cache Topologies.

Feature Area	Azure Redis Cache	NCache
Local Cache	Partial support Only OutProc cache is supported	Supported Both InProc and OutProc. InProc is much faster but your memory consumption is higher if you have multiple instances on the same machine. OutProc is slightly slower due to IPC and serialization cost but saves you memory consumption because there is only one copy per machine.
Client Cache (Near Cache)	Not Supported	Supported Client Cache is a local cache on the cache client machine but one that is connected and synchronized with the cache cluster. Client Cache gives a local cache performance (specially InProc) but with the scalability of a distributed cache. NCache allows you to configure Client Cache without any code changes to the client application.
Mirrored Cache	Supported	Supported Mirrored Cache is a 2-node Active-Passive cache. All clients connect to the Active node and data mirroring is done asynchronously. In case Active node goes down, Passive node automatically becomes Active and all clients connect to it automatically.
Replicated Cache	Not Supported	Supported

Feature Area	Azure Redis Cache	NCache
		In Replicated Cache, the entire cache is replicated on all nodes in the cluster. You can have more than 2 nodes and all nodes are active meaning clients connect to them directly. Updates are done synchronously within the cluster and are therefore slower than other topologies. But, reads are super-fast. Each client connects to only one node. You can enable load-balancing or specify an ordered server list for the clients to use.
Partitioned Cache	Partial support No failover support in case a Master goes down. The whole cluster becomes unusable. Partitioned Cache means data partitioning without replication. It's good when you don't want the memory and transaction cost of replication because you can always load data from the database if some of it is lost due to cache server going down. Its equivalent in Redis is a cluster of Masters without any Slaves. But, if a Redis Master without a Slave goes down, the entire cluster becomes unusable. This is because slots from this shard are not picked up by other Masters.	Supported Full failover support if any server goes down (although there is data loss). Partitioned Cache is a very powerful topology. You can partition without replication to speed up the cache and also use less memory because you can always reload some data if lost in the cache. In Partitioned Cache, the entire cache is partitioned and each cache server gets one partition. All partitions are created or deleted and their buckets reassigned automatically at runtime when you add/remove nodes. Data re-balancing feature is provided even if no partition is added or removed but when any partition gets overwhelmed with too much data. Each client is connected to all cache nodes. This allows it to directly go where the data is (single hop).
Partitioned-Replica Cache	Supported Redis supports Shards with Replicas (essentially Partitioned-Replica Cache).	Supported Same as Partitioned Cache (read above). Also provides a replica for each partition

Feature Area	Azure Redis Cache	NCache
	Except Redis allows more than one Replica for each Master whereas NCache only supports one Replica for each Partition. Redis support async and sync replications like NCache.	kept at another cache server. This provides reliability against data loss if a node goes down. Async and Sync, both replications are supported. Just like partitions, replicas are also created dynamically. Data balancing also updates replicas.
Partitioned Data Balancing	Partial support Redis allows you to manually re-balance data (re-shard). But, it is not done automatically like NCache does.	Supported Data is automatically rebalanced when you add/remove cache servers from the cluster. Data is also rebalanced automatically when one cache server has a lot more data than other servers. You can configure the threshold of difference for this. You can turn off auto rebalancing in this case and manually do it if you wish. This applies to both Partitioned Cache and Partition-Replica Cache.
Load Balancing	Supported	Supported Clients are balanced among server nodes in case of Replicated Cache topology. For Partitioned Cache topologies clients are connected to all nodes for single hop operation and therefore balanced as well.
Partitioned Data Affinity	Supported Redis uses Range Partitioning through which you can specify which data should reside in which partition.	Not Supported
Persistence	Supported Redis provides RDB and AOF persistence. RDB is a snapshot of cache at a time. And, AOF logs every transaction.	Supported NCache has the equivalent of RDB persistence through Dump/Reload tools that take a snapshot of the cache and persist them to disk or reload the cache from a previous dump.

Feature Area	Azure Redis Cache	NCache
		NCache also provides the equivalent of AOF through its Read-thru/Write-thru providers. You can write any cache update also to a data source of your choice which is more flexible than AOF which only logs to a file. And, you can rebuild the cache from your data source by implementing a CacheLoader interface that NCache calls upon startup. This CacheLoader is run on multiple servers in the cluster and is highly scalable.

2.5 WAN Replication

WAN replication is an important feature for many customers whose applications are deployed in multiple data centers either for disaster recovery purpose or for load balancing of regional traffic.

The idea behind WAN replication is that it must not slow down the cache in each geographical location due to the high latency of WAN for propagating the data replication. NCache provides Bridge Topology to handle all of this.

Read more about it at WAN Replication of In-Memory Cache.

Feature Area	Azure Redis Cache	NCache
Active – Passive	Not Supported	Supported Bridge Topology Active-Passive You can create a Bridge between the Active and Passive sites. The Active site submits all updates to the Bridge which then replicates them to the Passive site.
Active – Active	Not Supported	Supported Bridge Topology Active-Active You can create a Bridge between two active sites. Both submit their updates to the Bridge which handles conflicts on "last update wins" rule or through a custom conflict resolution handler provided by you. Then, the Bridge ensures that both

Feature Area	Azure Redis Cache	NCache
		sites have the same update.
Conflict Resolution	Not Supported	Supported By default, "last update wins" algorithm is used to resolve conflicts. But, you can specify a "custom conflict resolution handler" that is called to resolve conflict by comparing the content of both objects and deciding.
De-duplication	Not Supported	Supported NCache Bridge optimizes replication queue by de-duplicating items. If the same key is updated multiple times, it only replicates the most recent update.
Data Security	Not Supported	Supported You can encrypt data with 3DES and AES algorithms before transportation. Otherwise, you can use a VPN between data centers for security.

2.6 ASP.NET & ASP.NET Core Support

ASP.NET Core applications can persist their Sessions in a distributed cache.

Similarly, ASP.NET applications need three things from a good in-memory distributed cache. And, they are ASP.NET Session State storage, ASP.NET View State caching, and ASP.NET Output Cache.

ASP.NET Session State store must allow session replication in order to ensure that no session is lost even if a cache server goes down. And, it must be fast and scalable so it is a better option than InProc, StateServer, and SqlServer options that Microsoft provides out of the box. NCache has implemented a powerful ASP.NET Session State provider. Read more about it at NCache Product Features.

ASP.NET View State caching allows you to cache heavy View State on the web server so it is not sent as "hidden field" to the user browser for a round-trip. Instead, only a "key" is sent. This makes the payload much lighter, speeds up ASP.NET response time, and also reduces bandwidth pressure and cost for you. NCache provides a feature-rich View State cache. Read more about it at NCache Product Features.

Third is ASP.NET Output Cache. Since .NET 4.0, Microsoft has changed the ASP.NET Output Cache architecture and now allows third-party in-memory distributed cache to be plug-in. ASP.NET Output Cache saves the output of an

ASP.NET page so the page doesn't have to execute next time. And, you can either cache the entire page or portions of the page. NCache has implemented a provider for ASP.NET Output Cache.

Feature Area	Azure Redis Cache	NCache
ASP.NET Core Session Persistence (basic)	Supported	Supported NCache has implemented an ASP.NET Core Sessions Provider. NCache provides intelligent session replication and is much faster than any database storage for sessions.
ASP.NET Core Session Persistence (advanced)	Not Supported	Supported NCache also provides flexible session locking options to handle robots flooding an ASP.NET website. You can also related sessions with view state so when a session expires, all the corresponding view state is also removed.
ASP.NET Core Sessions (Multi-Site)	Not Supported	Supported NCache allows you to share ASP.NET Core sessions across multiple data centers. This serves situations where you don't want to replicate all sessions to each data center but want the ability to overflow traffic from one data center to another without losing your ASP.NET Core sessions. The session moves from one data center to the next as the user moves.
ASP.NET Session Caching (basic)	Supported	Supported NCache has implemented an ASP.NET Session State Provider (SSP) for .NET 2.0+. You can use it without any code changes. Just change web.config. NCache provides intelligent session replication and is much faster than any database storage for sessions.
ASP.NET Session Caching (advanced)	Not Supported	Supported

Feature Area	Azure Redis Cache	NCache
		NCache also provides flexible session locking options to handle robots flooding an ASP.NET website. You can also related sessions with view state so when a session expires, all the corresponding view state is also removed.
ASP.NET Sessions (Multi-Site)	Not Supported	Supported NCache allows you to share ASP.NET sessions across multiple data centers. This serves situations where you don't want to replicate all sessions to each data center but want the ability to overflow traffic from one data center to another without losing your ASP.NET sessions. The session moves from one data center to the next as the user moves.
ASP.NET View State Cache	Not Supported	Supported (advanced). Yes. NCache has an ASP.NET View State caching module. Use it without any code changes. Just modify config file. Here are some advanced features supported by NCache: - Group-level policy - Associate pages to groups - Link View State to sessions - Max View State count per user - More
ASP.NET Output Cache	Supported	Supported NCache has an ASP.NET Output Cache provider implemented. It allows you to cache ASP.NET page output in an In-Memory Cache and share it in a web farm.

2.7 Object Caching Features

These are the most basic operations without which an in-memory distributed cache becomes almost unusable. These by no means cover all the operations a good Cache should have.

Feature Area	Azure Redis Cache	NCache
Get, Add, Insert, Remove, Exists, Clear Cache	Supported	Supported NCache provides more variations of these operations and therefore more control to the user.
Expirations	Partial support Redis supports EXPIRE command through which you can do Absolute Expiration. But, to implement Sliding Expiration, you must call EXPIRE every time you access the cache item. This is more work on your part and is also more costly as an extra cache call is being made.	Supported Absolute and Sliding expirations provided. Absolute expiration is good for data that is coming from the database and must be expired after a known time because it might become stale. Sliding expiration means expire after a period of inactivity and is good for session and other temporary data that must be removed once it is no longer needed.
Lock & Unlock	Supported	Supported NCache provides both. Lock is used to exclusively lock a cached item so nobody else can read or write it. This item stays locked until either the lock expires or it is unlocked. NCache also provides "GetAndLock()", that locks the item before fetching it, and "InsertAndUnlock()" that updates the item and then unlocks it, all in one call.
Streaming API	Supported	Supported For large objects, NCache allows the cache clients to fetch them in "GetChunk()" manner and update them in "AppendChunk()" manner. With this, NCache clients can stream in or out large objects from the cache.
Transactions	Partial support. Redis neither provides rollbacks nor supports the "all or nothing" proposition in a multi-command transaction.	Partial support. Explicit locking Implicit locking (item versioning) Entry Processor (are atomic)

Feature Area	Azure Redis Cache	NCache
Data Portability	Not Supported Redis does not provide any support for transforming C# objects into Java or vice versa. You have to do this yourself.	Supported .NET to Java and Java to .NET object conversion supported without going through JSON/XML transformation. Configurable using a user friendly GUI.
Item Versioning	Not Supported You have to manage it yourself	Supported This ensures that only one client can update an item and all future updates will fail unless cache clients first fetch the latest version and then update it.
Multiple Object Versions	Not Supported You have to manage it yourself	Supported NCache allows two different versions of the same class to be stored in the cache by different apps. Each app retrieves its own version and the cache keeps a superset.

2.8 Managing Data Relationships in Cache

Since most data being cached comes from relational databases, it has relationships among various data items. So, a good cache should allow you to specify these relationships in the cache and then keep the data integrity. It should allow you to handle one-to-one, one-to-many, and many-to-many data relationships in the cache automatically without burdening your application with this task.

Feature Area	Azure Redis Cache	NCache
Key Based Relationships	Not Supported	Supported You can specify that cached item A depends cached item B which then depends on cached item C. Then, if C is ever updated or removed, B is automatically removed from the cache and that triggers the removal of A from the cache as well. And, all of this is done automatically by the cache.

Feature Area	Azure Redis Cache	NCache
		With this feature, you can keep track of one-to-one, one-to-many, and many-to-many relationships in the cache and invalidate cached items if their related items are updated or removed.
Key Based Relationships Across Caches	Not Supported	Supported This is an extension of Key Based Cache Dependency except it allows you to create this dependency across multiple caches.

2.9 Cache Synchronization with Database

Database synchronization is a very important feature for any good In-Memory distributed cache. Since most data being cached is coming from a relational database, there are always situations where other applications or users might change the data and cause the cached data to become stale.

To handle these situations, a good In-Memory distributed cache should allow you to specify dependencies between cached items and data in the database. Then, whenever that data in the database changes, the cache becomes aware of it and either invalidates its data or reloads a new copy.

Additionally, a good distributed cache should allow you to synchronize the cache with non-relational data sources since real life is full of those situations as well.

NCache provides a very powerful database synchronization feature.

Feature Area	Azure Redis Cache	NCache
SQL Dependency (Sync with SQL Server) (Event based)	Not Supported	Supported NCache provides SqlDependency support for SQL Server. You can associate a cached item with a SQL statement based dataset in SQL Server database. Then whenever that dataset changes (addition, updates, or removal), SQL Server sends a notification to NCache and NCache invalidates this cached item or reloads it if you have enabled it with ReadThrough. This feature allows you to synchronize the cache with SQL Server database. If you have a situation where some applications or users are directly updating data in the

Feature Area	Azure Redis Cache	NCache
		database, you can enable this feature to ensure that the cache stays fresh.
Oracle Dependency (Sync with Oracle) (Event based)	Not Supported	Supported NCache provides OracleDependency support for Oracle. You can associate a cached item with a SQL statement based dataset in Oracle database. Then whenever that dataset changes (addition, updates, or removal), Oracle sends a data notification to NCache and NCache invalidates this cached item or reloads it if you have enabled it with ReadThrough. This feature allows you to synchronize the cache with Oracle database. If you have a situation where some applications or users are directly updating data in the database, you can enable this feature to ensure that the cache stays fresh.
Db Dependency (Sync with OLEDB) (Polling based)	Not Supported	Supported NCache provides support for you to synchronize the cache with any OLEDB database. This synchronization is based on polling. And, although it is not as real-time as a database notification, it is more efficient. It is more efficient because in one poll, NCache can synchronize thousands of cached items instead of receiving thousands of individual database notifications from Oracle in case of OracleDependency.
File Dependency (Sync with Non-Relational Source)	Not Supported	Supported NCache allows you to specify a dependency on an external file. Then NCache monitors this file for any updates and when that happens, NCache invalidates the corresponding cached item.

Feature Area	Azure Redis Cache	NCache
		This allows you to keep the cached item synchronized with a non-relational data source.
Custom Dependency (Sync with any custom source)	Not Supported	Supported NCache allows you to implement a custom dependency and register your code with the cache cluster. Then, NCache calls your code to monitor some custom data source for any changes. When changes happen, you fire a dependency update within NCache which causes the corresponding cached item to be removed from the cache. This feature is good when you need to synchronize the cached item with a non-relational data source that cannot be captured by a flat file. So, custom dependency handles this case.

2.10 Event Driven Data Sharing

Event Driven Data Sharing has become an important use for in-memory distributed caches. More and more applications today need to share data with other applications at runtime in an asynchronous fashion.

Previously, relational databases were used to share data among multiple applications but that requires constant polling by the applications wanting to consume data. Then, message queues became popular because of their asynchronous features and their persistence of events. And although message queues are great, they lack performance and scalability requirements of today's applications.

NCache provides very powerful features to facilitate Event Driven Data Sharing. They are discussed below and compared with Redis.

Feature Area	Azure Redis Cache	NCache
Item Level Events (onInsert / onRemove)	Supported	Supported NCache can fire events to its clients whenever specific cached items are updated or removed based on client interest. You can register Java and .NET callbacks with NCache client and your callbacks are

Feature Area	Azure Redis Cache	NCache
		called in these cases. NCache uses its own socket-level protocol for this event propagation so it is super-fast.
Cache Level Events (Add/Insert/Remove)	Supported	Supported If turned on, NCache sends event notifications to all clients whenever any item is added, updated, or removed from the cache. You can register Java and .NET callbacks with NCache client and your callbacks are called in these cases.
Custom Events (Fired by Apps)	Not Supported	Supported NCache allows your applications to fire custom events into the cache cluster. And, other applications can register to be notified for these events. This feature allows you to coordinate a pub/sub scenario with asynchronous event driven coordination between various clients.
Continuous Query	Not Supported	Supported NCache provides a powerful Continuous Query (CQ) feature. CQ lets you specify a SQL query against which NCache monitors the cache for any additions, updates, or deletes. And, your application is notified through events whenever this happens.
Topic (Pub/Sub Messaging)	Supported	Supported NCache allows your applications to do Pub/Sub style messaging through Topic.
Queue (Pub/Sub Messaging)	Supported	Not Supported

2.11 SQL-Like Cache Search

In-Memory distributed cache is frequently used to cache objects that contain data coming from a relational database. This data may be individual objects or collections that are the result of some database query.

Either way, applications often want to fetch a subset of this data and if they have the ability to search the distributed cache with a SQL-like query language and specify object attributes as part of the criteria, it makes the In-Memory distributed cache much more useful for them.

NCache provides powerful SQL-like searching capability of the cache.

Feature Area	Azure Redis Cache	NCache
SQL Search	Not Supported	Supported NCache provides a rich SQL based searching capability. You can search the cache based on object attributes instead of just keys. You can also include Group, Subgroup, Tags, and Named Tags in your SQL query.
LINQ Queries	Not Supported	Supported NCache allows you to search the cache with LINQ queries from .NET applications. LINQ is a popular object querying language in .NET and NCache has implemented a LINQ provider. So, if you're comfortable using LINQ, you can search the cache the same way you would with NCache SQL.
SQL Search on Tags, Named Tags & Groups	Not Supported	Supported NCache allows you to include Tags, Named Tags, and Group names as part of you SQL search criteria.

2.12 Data Grouping

An in-memory distributed cache should be much more than a Hashtable with a (key, value) pair interface. It needs to meet the needs of real life applications that expect to fetch and update data in groups and collections. In a relational database, SQL provides a very powerful way to do all of this.

We've already explained how to search an in-memory distributed cache through SQL and LINQ. Now let's discuss Groups, Tags, and Named Tags. These features allow you to keep track of collections of data easily and even modify them.

Feature Area	Azure Redis Cache	NCache
Groups/Subgroups	Not Supported	Supported NCache provides the ability for you to group cached items in a group-subgroup combination (or just group with no subgroup). You can later fetch or remove all items belonging to a group. You can also fetch just the keys and then only fetch subset of them.
Tags	Not Supported	Supported NCache provides a concept called Tags. A Tag is a string that you can assign to one or more cached items. And one cached item can be assigned multiple Tags. And, later, you can fetch items belonging to one or more Tags in order to manipulate them. You can also include Tags in SQL or LINQ search as part of the criteria.
Named Tags	Not Supported	Supported NCache provides Named Tags feature where you can assign a "key" and "tag" to one or more cached items. And, a single cached item can get multiple Named Tags. Later, you can fetch items belonging to one or more Named Tags. You can also use Named Tags in SQL and LINQ queries as part of the criteria.

2.13 Read-through, Write-through & Cache Loader

Many people use in-memory distributed cache as "cache on the side" where they fetch data directly from the database and put it in the cache. Another approach is "cache through" where your application just asks the cache for the data. And, if the data isn't there, the in-memory distributed cache gets it from your data source.

The same thing goes for write-through. Write-behind is nothing more than a write-through where the cache is updated immediately and the control returned to the client application. And, then the database or data source is updated asynchronously so the application doesn't have to wait for it.

NCache provides powerful capabilities in this area.

Feature Area	Azure Redis Cache	NCache
Read-through	Not Supported	Supported NCache allows you to implement multiple read-through handlers and register with the cache as "named providers". Then, the client can tell NCache to use a specific read-through upon a "cache miss". NCache also allows you to add read-through handlers at runtime without stopping the cache.
Write-through & Write behind	Not Supported	Supported NCache allows you to implement multiple write-through handlers and register with NCache as "named providers". Then, whenever application updates a cached item and tells NCache to also call write-through, NCache server calls your write-through handler. If you've enabled write-behind, then NCache updates the cache immediately and queues up the database update and a background thread processes it and calls your write-through handler.
Auto Reload at Expiration & Database Synchronization	Not Supported	Supported If you've implemented a read-through handler, NCache allows you to use it to specify that whenever a cached item expires, instead of removing it from the cache, NCache should call your read-through handler to read a new copy of

Feature Area	Azure Redis Cache	NCache
		that object and update the cache with it. You can specify the same when database synchronization is enabled and a row in the database is updated and a corresponding cached item would have been removed from the cache but is now reloaded with the help of your read-through.
Cache Startup Loader	Partial support Redis persists everything in a file. Upon restarts, Redis re-loads the state of the cache. Some data loss is to be expected since the replication is asynchronous. But, you can't write a custom cache loader.	Supported NCache lets you implement a Cache Loader and register it with the cache cluster. NCache then calls it to prepopulate the cache upon startup. Cache Loader is your code that reads data from your data source/database.

2.14 Big Data Processing

For analysis and processing large amount of data becomes faster if done in-memory A distributed cache is a scalable in-memory data store. And, if it can support the popular Map/Reduce style processing, then you're able to speed up your work greatly.

Feature Area	Azure Redis Cache	NCache
Map-Reduce Query	Not Supported	Supported. NCache provides a MapReduce framework where you program can run on cache servers for parallel processing of Big Data.
Aggregators	Not Supported	Supported. NCache provides Aggregator that works with MapReduce framework and provides you statistical data.
Entry Processor	Not Supported No entry processor like capability provided even though a very basic LUA scripting engine allows scripts to be executed on servers.	Supported. NCache fully supports Entry Processor execution on cache nodes in parallel.

2.15 Third Party Integrations

Memcached is an open-source in-memory distributed caching solution which helps speed up web applications by taking pressure off the database. Memcached is used by many of the internet's biggest websites and has been merged with other technologies. NCache implements Memcached protocol to enable users with existing Memcached implementations to easily migrate to NCache. No code required for this.

NHibernate is a very powerful and popular object-relational mapping engine. And, fortunately, it also has a Second Level Cache provider architecture that allows you to plug-in a third-party cache without making any code changes to the NHibernate application. NCache has implemented this NHibernate Second Level Cache provider. See NHibernate Second Level Cache for details.

Entity Framework from Microsoft is also a very popular object-relational mapping engine. And, although Entity Framework doesn't have a nice Second Level Cache provide architecture like NHibernate, NCache has nonetheless implemented a Second Level Cache for Entity Framework.

Feature Area	Azure Redis Cache	NCache
NHibernate 2nd Level Cache	No Official Support Not officially supported. Open source projects only	Supported NCache provides an NHibernate L2 Cache provider that you can plug-in through web.config or app.config changes. NCache has also implemented database synchronization feature in this so you can specify which classes should be synchronized with the database. NCache lets you specify SqlDependency or DbDependency for this.
Entity Framework Core Cache	Not Supported Not supported by Redis.	Supported Extension Methods NCache has implemented EF Core Extension Methods for caching to make it really simple for EF applications to use caching. It also gives full control to the application about how to cache data.
Entity Framework Cache	Not Supported Not officially supported or even recommended by Redis. Small open source projects are available without support.	Supported Custom ADO.NET Provider NCache has implemented a behind-the-scene second level cache for Entity Framework. You can plug-in NCache to your EF application, run it in analysis mode, and quickly see all the queries

Feature Area	Azure Redis Cache	NCache
		being used by it. Then, you can decide which queries should be cached and which ones skipped.
Memcached Protocol Server	Supported	Supported NCache has implemented Memcached protocol fully. This means you can plug-in NCache as an in-memory distributed cache as a replacement of Memcached. Two ways are offered to use Memcached applications with NCache. Memcached Pug-In: All the popular Open Source .NET Memcached client libraries have been implemented for NCache. Memcached Gateway: Using this you can store your application data from any application that use the Memcached.
Memcached Smart Wrapper	Not Supported	Supported NCache has implemented the popular .NET and Java Memcached client libraries which in-turn calls NCache. This allows you to plug-in Memcached client library to your application without any code change or recompilation. This wrapper does not require you to go through a Memcached Protocol Server which is an extra hop.

2.16 Security & Encryption

Many applications deal with sensitive data or are mission critical and cannot allow the cache to be open to everybody. Therefore, a good In-Memory distributed cache provides restricted access based on authentication and authorization to classify people in different groups of users. And, it should also allow data to be encrypted inside the client application process before it travels to the distributed cache.

NCache provides strong support in all of these areas.

Feature Area	Azure Redis Cache	NCache

Feature Area	Azure Redis Cache	NCache
Authentication (Active Directory / LDAP)	Partial support No support for Active Directory or LDAP authentication. Redis provides only text based authentication where administrator keeps a list of users and passwords in a configuration file.	Supported You can authenticate users against Active Directory or LDAP. If security is enabled, nobody can access the cache without authentication and authorization.
Authorization	Supported	Supported You can authorize users to be either "users" or "admins". Users can only access the cache for read-write operations while "admins" can administer the cache.
Data Encryption	Not Supported	Supported You can enable encryption and NCache automatically encrypts all items one client-side before sending them to the cache. And, this data is kept encrypted while in the cache. And decryption also happens automatically and transparently inside the client process. Currently, 3DES and AES128 / AES196 / AES256 encryptions are provided and more are being added.
Secure Communication	Partial support No SSL support. Redis is meant to be used within a trusted network without outside access. User must install their own protection mechanism to secure their data.	Supported NCache provides SSL support for client/server communication. Additionally, strong encryptions are provided by NCache so you can encrypt data over an unsecured connection.

2.17 Cache Size Management (Evictions Policies)

An in-memory distributed cache always has less storage space than a relational database. So, by design, an in-memory distributed cache is supposed to cache a subset of the data which is really the "moving window" of a data set that the applications are currently interested in.

This means that an in-memory distributed cache should allow you to specify how much memory it should consume and once it reaches that size, the cache should evict some of the cached items. However, please keep in mind that if you're caching something that does not exist in the database (e.g. ASP.NET Sessions) then you need to do proper capacity planning to ensure that these cached items (sessions in this case) are never evicted from the cache. Instead, they should be "expired" at appropriate time based on their usage.

Feature Area	Azure Redis Cache	NCache
Specify Cache Size (in MBs)	Supported	Supported
LRU Evictions (Least Recently Used)	Supported	Supported
LFU Evictions (Least Frequently Used)	Not Supported	Supported
Priority Evictions	Not Supported	Supported NCache also lets you specify a "do not evict" priority for some cached items and then they are not evicted.
Do Not Evict Option	Supported	Supported NCache lets you specify "do not evict" option for the entire cache. Then, nothing is evicted even when cache is full. Instead, the client applications receive an error stating that the cache is full when they try to add data to the cache.

2.18 Cache Administration

Cache administration is a very important aspect of any distributed cache. A good cache should provide the following:

1. GUI based and command line tools for cache administration including cache creation and editing/updates.
2. GUI based tools to monitor cache activities at runtime.
3. Cache statistics based on PerfMon (since for Windows PerfMon is the standard)

NCache provides powerful support in all these areas.

Read more about it at Administration and Monitoring Tools.

Feature Area	Azure Redis Cache	NCache
Admin Tool (GUI)	Not Supported	Supported NCache Manager is a powerful GUI tool for NCache. It gives you an explorer style view and lets you quickly administer the cache cluster from a single place. This includes cache creation/editing and many other functions.
Monitoring Tool (GUI)	Not Supported	Supported NCache Monitor is a powerful GUI tool that lets you monitor NCache cluster wide activity from a single location. It also lets you monitor all of NCache clients from a single location. And, you can incorporate non-NCache PerfMon counters in it for comparison with NCache stats. This real-time comparison is often very important.
PerfMon Counters	Not Supported	Supported NCache provides a rich set of PerfMon counters that can be seen from NCache Manager, NCache Monitor, or any third party tool that supports PerfMon monitoring.
Admin Tools (PowerShell)	Not Supported	Supported NCache provides a rich set of PowerShell admin tools. You can create a cache, add remote clients to it, add server nodes to it, start/stop the cache, and much more. Use these tools from your PowerShell scripts and automate various cache admin operations.
Admin Tools (Command Line)	Supported	Supported NCache provides a rich set of command line tools. You can create a cache, add remote clients to it, add server nodes to it,

Feature Area	Azure Redis Cache	NCache
		start/stop the cache, and much more. Use these tools from your scripts and automate various cache admin operations.
Admin & Monitoring (API)	Supported	Supported NCache provides Java and .NET API to manage and monitor the caches & client. Using this API you can stop/start the cache, get the statistics of the connected clients or get the health info of the cache cluster.

3 Conclusion

As you can see in a very detailed fashion, we have outlined all of NCache features and all the corresponding Redis features or a lack thereof. I hope this document helps you get a better understanding of Redis versus NCache.

In summary, Redis is a popular free cache mainly on Unix/Linux platform with clients running on either Unix or Windows. And, the Windows port of Redis server done by Microsoft OpenTech group not very stable and therefore not as reliable. In fact, Microsoft itself is using the Linux version of Redis in Azure. So, if you want to use Redis, you'll probably want to run it on Unix and then access it from your Windows app servers.

But, the true cost of ownership for an in-memory distributed cache is not just the price of it. It is the cost to your business. The most important thing for many customers is that they cannot afford unscheduled downtime (especially during peak hours). And, this is where an elastic cache like NCache truly shines.

Additionally, all those caching features that NCache provides are intended to give you total control over the cache and allow you to cache all types of data and not just simple data. This is something Redis cannot do.

Please read more about NCache and also feel free to download a fully working 60-day trial of NCache from:

- NCache details.
- Download NCache.

