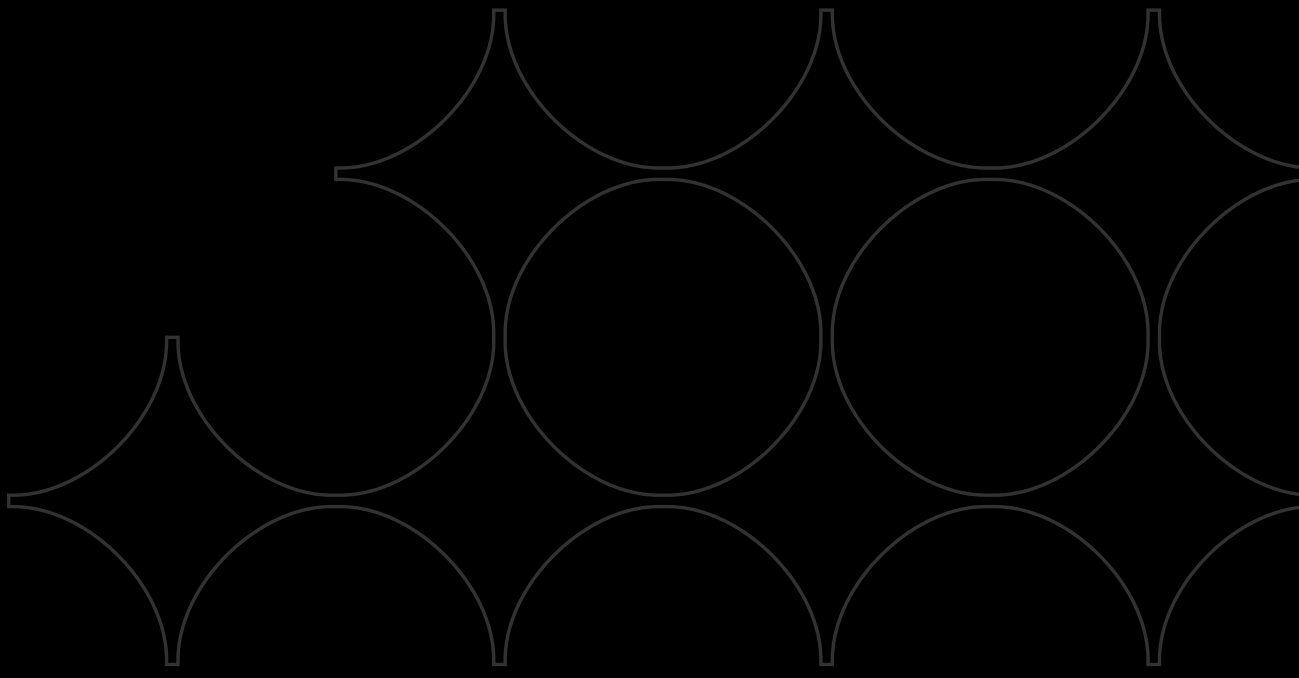
HAZELCAST

Seven Reasons Why Hazelcast Is the Best Caching Technology for You

White Paper



White Paper

Seven Reasons Why Hazelcast Is the Best Caching Technology for You

Hazelcast is a real-time, intelligent applications platform that lets you take advantage of computer memory to significantly accelerate data processing environments. One simple use case for Hazelcast is data caching. But Hazelcast offers you much more than caching. So one thing to keep in mind is that while basic caching might suit your immediate need, there are many other capabilities you should be adding to your IT infrastructure that can be powered by the Hazelcast Platform. After reading this paper, you will see how Hazelcast can be leveraged for building modern, high performance business applications.

As you also investigate technologies like Redis, Aerospike, or GridGain to boost the performance of your existing IT infrastructure, consider the other factors that should play a role in your decision. Ease of use and reliability should be important considerations. And for building real-time streaming applications, only Hazelcast provides the high powered stream processing engine that takes advantage of in-memory speeds.

Quick Review: What Is a Cache?

In computer systems, a cache is a technology for temporarily storing data in a location where it can be repeatedly accessed much more quickly than from its original location or form. In most cases, caches store data in a computer's main memory, aka "random access memory" (RAM). This type of "in-memory" architecture is especially useful for speeding up systems or applications that normally rely on data that is stored in a relatively slow medium, like across the network or on spinning disks (which is how most data is stored today). Caches are also useful for storing computed results, especially from complex calculations, so that subsequent references to the computed results do not require the time-consuming calculations to be repeated. The benefit of a cache is more pronounced in environments with larger cached objects, and/or with greater repetition of data access.

The biggest issue with in-memory caches is the cost of RAM. Since RAM is far more expensive than disk drives or solid-state drives (SSDs), it is cost-prohibitive to put all your data into an in-memory system. You need to weigh the speed advantage versus the extra cost. Fortunately, the trend of dropping RAM prices continues, making the advantage of in-memory technologies more realizable. Also, newer technologies like the Intel® Optane™ persistent memory (PMem) make in-memory usage more accessible to a broad audience, as Optane chips offer significant cost savings over RAM, while delivering a significant speed increase over SSDs.

How Caches Work

A cache typically works by having the calling application first make a request for specific data from the cache. Whenever the caller tries to access data, instead of normally connecting to the original data source, it will first request the data from the cache. If the data does not exist in the cache (a situation known as a "cache miss"), then the caller requests it from the original data source. When that data is retrieved, it is also inserted into the cache so that any subsequent requests for that same exact data can get it from the cache. The tradeoff for a cache is the extra work required to first check the cache, and then to insert new data into the cache. However, if cached data is frequently accessed, then the retrieval attempts from the original location are reduced, and the speed advantage of the cache medium more than makes up for the cache overhead.

When used as a cache, Hazelcast does not require you to include this logic in your code. You can configure Hazelcast to get data directly from the source whenever that data is requested by the calling application. Updates to data in Hazelcast can also be propagated back to the source.

Basic Versus Distributed Caching

If you're an application developer, you most likely incorporated a cache in your applications, even though you may not have done anything special to create it. For example, if you read data from a database, store it in an application-defined data structure, and then run several computations on that data, you are essentially using a cache. This, of course, is limited by the amount of RAM on your server.

Now suppose you could run computations on data sets that are larger than the amount of RAM on your computer. This hypothetical situation is not referring to the use of swap space, which is disk-based; it is referring to using physical RAM. This is where distributed caching can help. Distributed caching is a technology where a large RAM-based cache is created from the RAM of multiple computers that are coordinated together so applications have much more memory at their disposal.

Baseline Requirements for Large-Scale Caching

When working with distributed caches, there are some baseline expectations you should have. We will discuss the baseline requirements below. Note that these are not advantages or differentiators, so if your technology includes these characteristics, that is a good start, but not the final word for your evaluation.

In-memory core. To deliver the highest speed increases over disk-based stores, a caching system must be memory-based.

Speed optimizations. To efficiently scale on a multi-core hardware server, a caching system must be multi-threaded. Single-threaded systems cannot automatically leverage multiple cores, so as a workaround, you incur unnecessary administrative and processing overhead by running multiple caching server instances on the same hardware server.

Arbitrary data types. Caching systems should be able to accept any data format so that additional processing power is not expended on converting data to a specific format.

Key-based access. Caching data by keys is the most basic method for caching data, but is only one way of providing data access.

Multi-key operation. Caching systems must be able to look up multiple values in a single request. This is done by grouping keys together in certain access operations to consolidate the processing to improve performance.

Atomic operations and transactions. To eliminate conflicts, operations and transactions in caching systems must be atomic so that there are no "partial updates" in the system caused by two simultaneous operations.

Data expiration. Caching systems need multiple ways to expire data, including time-to-live (TTL) policies, to remove obsolete/stale data or to make room for newly cached data.

Eviction policies. Caching systems need ways to remove existing data in the cache to make room for newer data to be cached.

Intelligent caching. Caching systems should offer more than just read and write operations, and should provide operations on shared data that can be instantly seen by all authorized applications accessing the cache.

Distributed servers. Caching systems should be distributed across multiple computer servers and connected by a network, to provide horizontal scalability.

Request pipelining. Caching systems should incorporate a request pipelining capability that allows multiple requests in one transaction to reduce network traffic and thus increase performance.

High availability. Despite dealing with transient data, caching systems need to be highly available through redundancy and replication strategies to maintain the performance benefits that business-critical applications need.

Data persistence. When outages occur that disrupt nodes in the caching system, data persistence allows quick recovery of the state of the system prior to the outage.

Scalable shared-nothing architecture. Distributed caching systems must leverage a shared-nothing architecture to support elastic scalability. This lets you easily and efficiently expand or contract your caching cluster to meet your load requirements.

Local cache replicas. Optimizations like near-cache are necessary to align with application characteristics to deliver the highest performance and efficiency.

Seven Reasons Why Hazelcast Is Best for You

Hazelcast addresses all of the requirements above, and more importantly, it lets you build fast applications with in-memory capabilities (i.e., all data is stored in memory without the need to write to disk) to do more than what is possible with a simple cache. This section of the paper describes seven Hazelcast characteristics that make it the best in-memory technology for your modern applications.

1. Proven Performance

If you are looking to boost the performance of your applications and systems, it makes sense to go with the technology that can provide the greatest speed gains.

Hazelcast benchmarks show that our performance continually exceeds the competition. As part of our comparison efforts, we strive for fairness and transparency, and publish our work which describes the environment and configuration to encourage independent verification. We have observed other published benchmarks to be flawed with clear biases. We have documented our findings and have

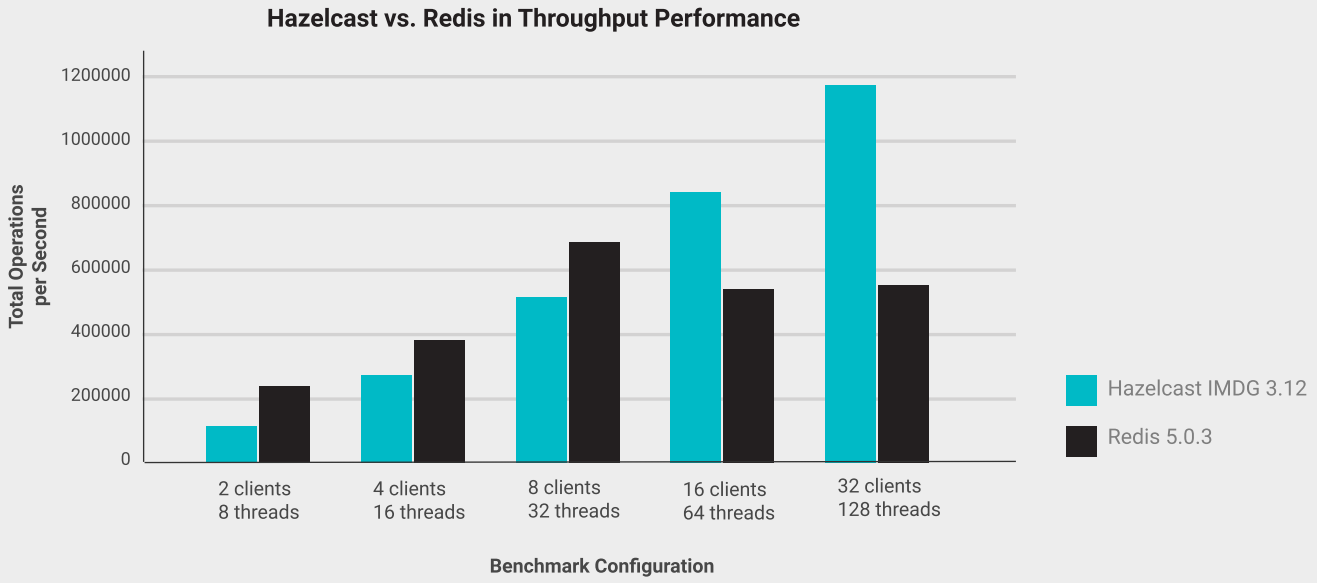
confirmed that in corrected benchmark tests, Hazelcast outperforms the competition in terms of both higher throughput and lower latency.

We have also observed that even if a benchmark is fair, the underlying behavior of the technology can create misleading results due to tradeoffs. For example, Redis automatically and silently stops replicating data under high load to keep up with the work. This creates artificially elevated performance figures because less work is done per transaction. The problem is that if a node fails, all unreplicated data on that node will be lost, making this behavior unacceptable for any production environment. Since the stoppage of replication is not an obvious or well-documented feature of Redis, users can lose data without knowing it, and thus get unexplainable results. More details of this risky behavior by Redis are discussed in the Hazelcast blog, Redis Load Handling vs Data Integrity: Tradeoffs in Distributed Data Store Design. Hazelcast does not make a "performance at any cost" design decision, so replication continues at high load, giving you superior performance while retaining data safety.

Our latest published benchmarks are available below:

- Redis vs. Hazelcast Benchmark
- Hazelcast Responds to Redis Labs' Benchmark
- GridGain/Apache Ignite vs Hazelcast Benchmark

In the chart below, you can see how Hazelcast shows a significant performance throughput advantage over another caching technology (Redis) at higher loads.

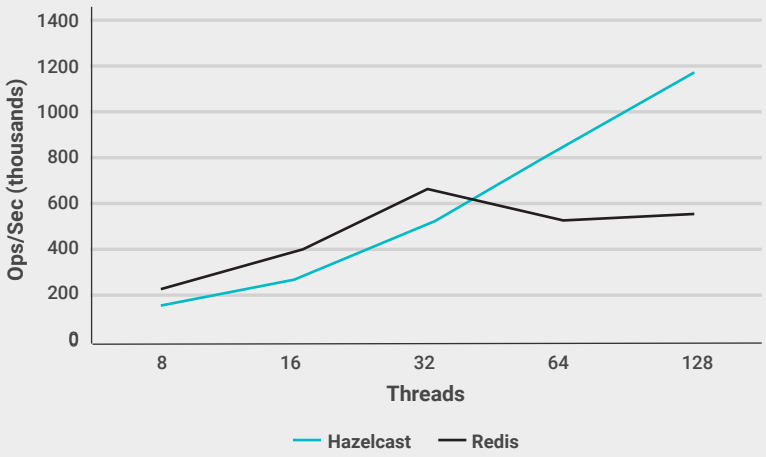


Hazelcast vs. Redis Comparing Total Operations per Second

2. Linear Scale

Hazelcast is architected to scale efficiently to handle increases in load and data volume without adding unnecessary administrative overhead. Redis, as an example, is a single-threaded technology that prohibits efficient scale for higher loads.

The chart below shows the scaling characteristics of throughput for both Hazelcast and Redis. You can see that both Hazelcast and Redis scale similarly at lower loads, but once higher workloads are involved which require more threads, Hazelcast continues to scale linearly, while Redis fails to scale further.



Hazelcast continues to scale to handle increases in load and data volume, while Redis tops out at 32 threads.

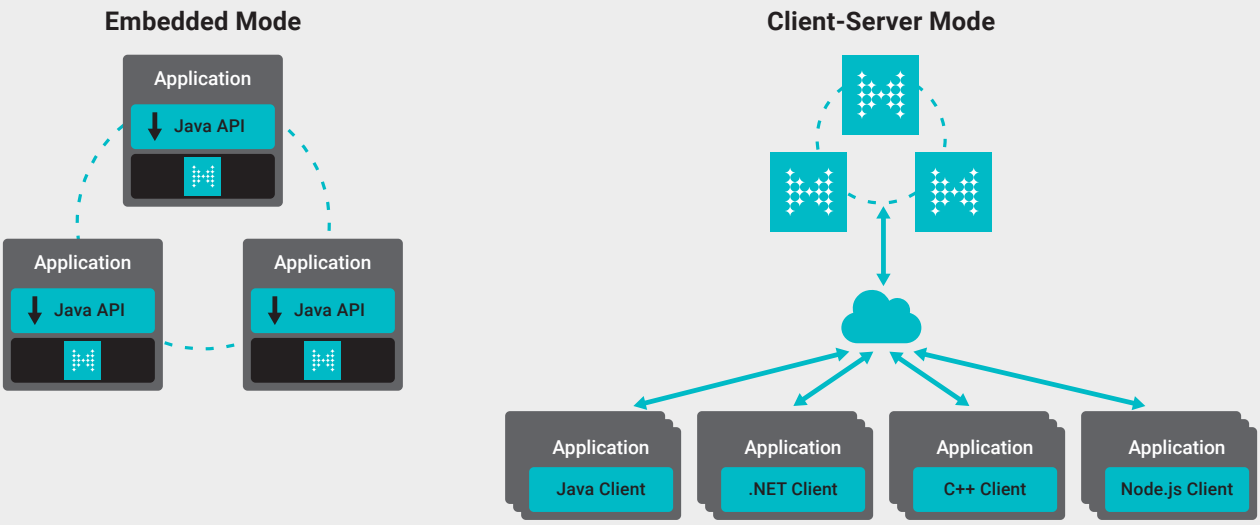
3. Ease of Use

Hazelcast is easy to get started with, easy to deploy, and easy to maintain. It was architected to minimize the effort that is typically required for using large-scale data processing systems. To make it easy to get started, Hazelcast offers:

- Free software downloads with accompanying getting-started guides
- The simplicity of a single JAR file to start development on your laptop
- Free online training
- Support for a variety of popular programming languages (C++, Go, Java, .Net, Node.js, Python, Scala)
- Support for popular APIs and integrations (e.g., JCache, jclouds, Hibernate, Spring, Memcached)
- Easy setup with no external dependencies necessary prior to using Hazelcast

Once you have applications that are ready for production deployment, Hazelcast features the following characteristics to simplify the DevOps effort:

- Packaged as a single JAR file, with no complex systems to install across a cluster
- Lightweight runtime for flexibility in deployments, to run at the edge, embedded in your applications, in a client/server topology, in a Kubernetes cluster, or in the cloud
- Embedded security capabilities (pluggable authentication, authorization controls, customizable security APIs, encryption over the wire) to protect your data from theft/tampering



Hazelcast can be embedded in your apps, or be deployed in a client-server model.

As shown in the diagram above, Hazelcast can be deployed in either embedded mode or client-server mode. In embedded mode, you do not have to maintain a system running in a cluster, as the "cluster coordination" intelligence is packaged completely in the Hazelcast library. Each application will contain the coordination logic without a separate underlying software installation.

- Once you have Hazelcast running in production, the ease of use continues in the maintenance stages with the following characteristics:
- No required external dependencies, with less maintenance complexity, simplifying the DevOps maintenance lifecycle
- API stability to avoid application rewrites upon Hazelcast upgrades
- Rolling upgrades to minimize downtime when upgrading Hazelcast
- Support for blue-green deployments to reduce downtime and upgrade risk, by redirecting all clients to the green cluster (formerly the test cluster, promoted to the live cluster) while the current live (blue) cluster is set as the offline/test (green) cluster
- Data persistence to minimize downtime when a restart is needed (e.g., after planned downtime or hardware failure)
- Authentication certificates update and encryption keys rotate with no downtime

4. Reliability

Even though cached data is permanently stored elsewhere and can be re-retrieved when needed, that does not mean that cache reliability is not a critical requirement. Cache reliability should be considered just as important as the reliability of other components in your system. Since caching is used to accelerate application performance, a cache failure can mean a performance drop that puts the system at risk of missing SLAs. Hazelcast delivers reliability in the areas described below.

High availability. Hazelcast provides a distributed architecture that replicates data across nodes in the cluster to ensure high availability despite hardware failure. When a node fails, Hazelcast will ensure that the replicas of the data in that failed node will be available to maintain continuity in the system. If there is a local network outage that creates a network partition, in which a portion of the cluster cannot

communicate with another, Hazelcast has the logic to handle this "split-brain" situation. Each sub-cluster will continue to operate, including updating data. When the network outage is resolved and the two sub-clusters can communicate with each other again, Hazelcast will use the configured policy to merge the sub-clusters so a cohesive view of the cached data is achieved in the full, restored cluster.

Disaster recovery. Hazelcast also safeguards against a complete site-wide disaster with WAN replication. This lets you create a complete copy of the cache to a geographically remote location, to work in conjunction with disaster recovery (DR) strategies you might have in place for your other data management platforms. You can deploy active-active or active-passive configurations depending on your needs. Hazelcast optimized the WAN replication capabilities to make replication faster and more efficient, to achieve smaller recovery point objectives (RPOs). Also, Hazelcast provides an automatic failover feature built into the client libraries so that if the primary cluster goes down, client requests are automatically routed to the backup cluster, to achieve smaller recovery time objectives (RTOs).

Data safety. Data reliability is a core strength of Hazelcast, and ongoing testing shows that it continues to support data safety, even at high load. Hazelcast can support industry-leading performance without increasing the risk of data loss. Contrast this to other technologies that tradeoff data safety capabilities for the sake of handling higher load. As mentioned earlier in this paper, one example is discussed in the Hazelcast blog, Redis Load Handling vs Data Integrity: Tradeoffs in Distributed Data Store Design.

Strong consistency and data correctness. Unlike other caching technologies, Hazelcast gives you the option to build applications that guarantee data consistency. This is ideal for environments where the correctness of data in the cache is a critical business requirement. Any use case that involves financial transactions, mission-critical data, or time-sensitive data requires a consistent, correct view of data. Other technologies cannot guarantee strong consistency since they lack the required built-in intelligence. Redis, for example, clearly calls out that it is not able to guarantee strong consistency, and explains why here: (https://redis.io/topics/cluster-tutorial). See the section on **Data Consistency** below for more information on how Hazelcast is able to guarantee consistency.

Reducing planned downtime. The capabilities described above can reduce unplanned downtime, and Hazelcast also offers capabilities to reduce downtime associated with planned maintenance.

- **Blue-green deployments.** Hazelcast provides support for blue-green deployments so that all clients can be easily redirected from the live production cluster (the "blue" cluster) to the upgraded/tested/validated secondary cluster (the "green" cluster). This lets you do any maintenance work including software upgrades on the secondary cluster and quickly switch over to it when it is ready. This switchover promotes the green cluster to be the new live blue cluster, and switches the former blue cluster into the secondary cluster on which the next round of maintenance can be done. If any problems are discovered with the new blue cluster, Hazelcast lets you quickly switch back to the original blue cluster to avoid downtime.
- **Persistence.** Hazelcast lets you persist cached data to disk, so if there is a node that needs to be shut down temporarily, it can be quickly recovered to its fully operational state by reading the saved state from the data persistence files. Hazelcast partners with Intel to optimize the persistence feature on Intel Optane PMem to dramatically increase the recovery speed compared to SSDs.
- **Rolling upgrades.** Hazelcast supports rolling upgrades (within a major version number) to keep a cluster live while software updates are performed on a node-by-node basis.

5. Data Consistency

In many situations, data consistency in a distributed environment is crucial for ensuring your system reflects the absolute up-to-date state of your data. As mentioned earlier, any use case that involves financial transactions, mission-critical data, or time-sensitive data requires a consistent, correct view of data. But per the CAP Theorem, which states that for a distributed system that can tolerate network partitions (the "P" in CAP), the system can provide either consistency ("C") or availability ("A"), but not both at the same time. The tradeoff of consistency versus availability is beyond the scope of this paper, but suffice it to say that many systems do not give you a choice on this tradeoff. Hazelcast lets you choose between data consistency or availability to suit your use case's specific requirements.

The Hazelcast CP Subsystem lets you build systems with strong data consistency, when that capability is critical in your cache. This feature is proven to be reliable, as demonstrated by the renowned and rigorous Jepsen test suite (https://jepsen.io). This testing is described in the Hazelcast blog, Testing the CP Subsystem with Jepsen.

As mentioned earlier, other technologies cannot guarantee strong consistency since they lack the required built-in intelligence. Redis, for example, clearly calls out that it is not able to guarantee strong consistency and explains why here: (https://redis.io/topics/cluster-tutorial).

Hazelcast offers other capabilities to support data consistency, including support for split-brain detection and conflict resolution (mentioned in the **Reliability** section), pessimistic locking, XA transaction support, and atomic operations. All of these capabilities give you more flexibility in how you leverage in-memory data as part of your overall system.

6. Security

Hazelcast includes security controls to protect your data from unauthorized use. And while security mechanisms usually add overhead that impacts performance, Hazelcast optimizes the security capabilities to ensure high performance. For example, the end-to-end TLS encryption that protects transmitted data across all clients and cluster members is optimized to exhibit only a small percentage of performance degradation, and is much faster throughput than the Java SSLEngine. And as an alternative to TLS, Hazelcast provides symmetric encryption with support for algorithms such as DES/ECB/PKCS5Padding, PBEWithMD5AndDES, Blowfish, and DESede.

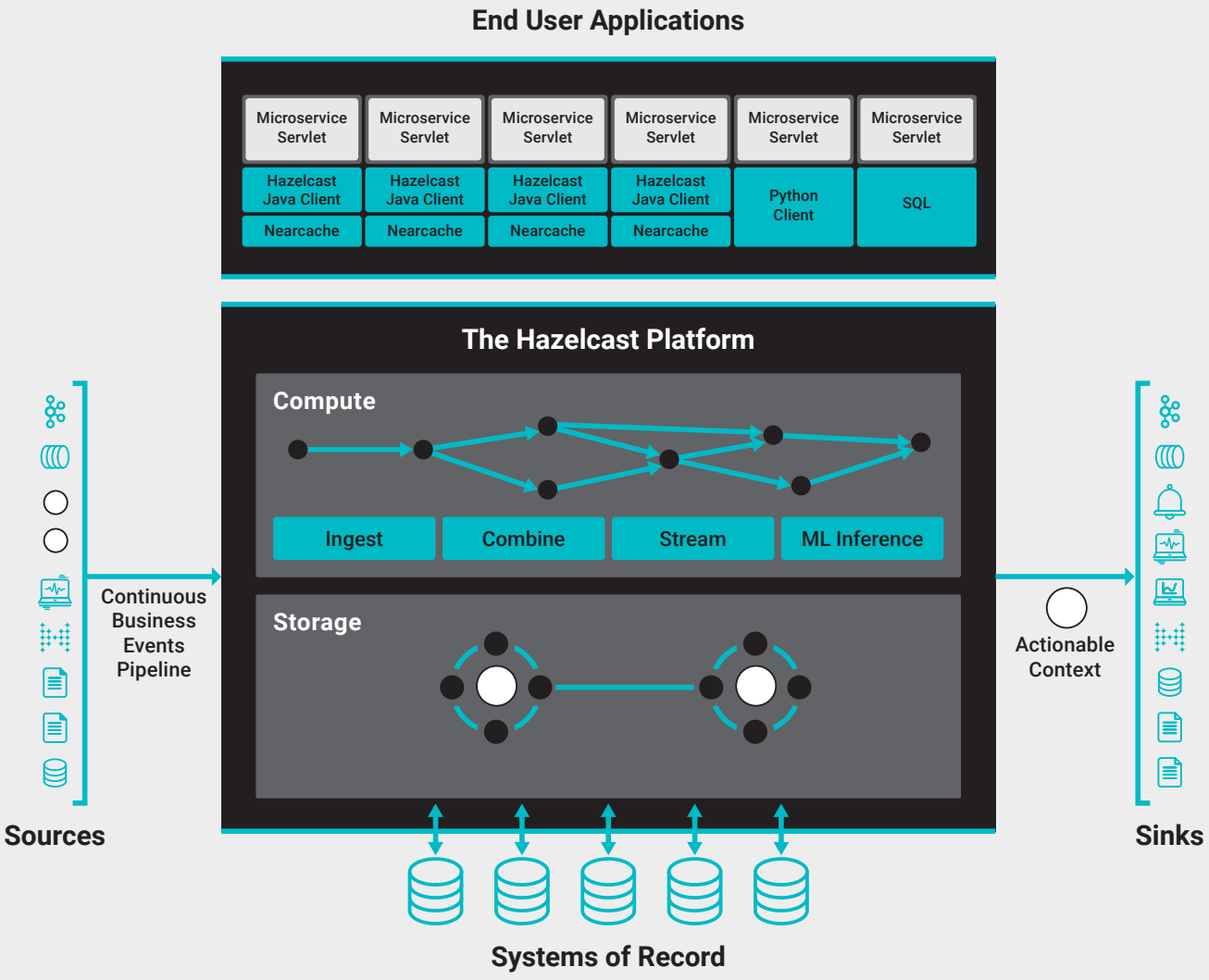
Hazelcast also provides JaaS-based pluggable authentication (to connect with LDAP/Active Directory or Kerberos), X.509 certificate-based mutual authentication, and role-based authorization controls. It also offers Socket Interceptor and Security Interceptor, which are hooks to allow custom security controls. Socket Interceptor is used for custom connection procedures, like certificate exchange as an extra measure to prevent rogue processes from joining the cluster. Security Interceptor lets you intercept all API calls to filter results sets, and also can be used for auditing.

7. Added Capabilities: Stream and Batch Processing

Hazelcast is more than a simple data store (e.g., an in-memory database). It is a high-speed application infrastructure that operates on data structures residing in a seamless, shared pool of RAM and processing power across many nodes in a cluster. In other words, Hazelcast is a computing cluster that lets you easily build distributed, parallelized applications, where the Hazelcast framework deploys your code to all nodes in the cluster.

If your immediate needs entail caching data from a data store to boost performance, your short-term future needs will probably include application enhancements that will simplify ongoing maintenance. Instead of writing application logic that treats a cache as a separate store, your applications can operate directly on data structures that appear to be in local memory (but are in fact shared across the pool of RAM). This simplifies the coding effort on aggregations, transformations, and other computations, and allows the development of a new generation of applications that natively incorporate in-memory speeds.

Hazelcast also provides stream processing capabilities to process data in real time as it arrives, going beyond simple data storage. This gives you a versatile data processing environment that lets you complement batch processing with real-time streaming. You can build comprehensive data applications that leverage both historical data and real-time data. The core functionality lets you process high-speed event streams from a variety of data sources, and then deliver the outputs to a variety of destinations.



Hazelcast provides stream processing capabilities as part of the in-memory computing platform.

The integrated processing capabilities simplifies the infrastructure for building these combined historical/real-time applications. It reduces the number of moving parts by colocating ingestion, processing, and the serving layer. You get lower architectural and operational complexity in your large-scale systems.

Hazelcast is also integrated with Apache Beam (the Beam capability matrix is here: https://beam.apache.org/documentation/runners/capability-matrix/), giving you more options for your data processing needs. If you are building data pipelines in Beam, you can leverage Hazelcast stream processing capabilities to execute stream processing pipelines for use cases that require the highest levels of performance.

Conclusion

With so many strategic data initiatives that businesses plan, it is important to have the best technologies in place. It is often difficult to determine which technologies are best aligned with your requirements, so hopefully this document has given you ideas on what capabilities to prioritize, and has helped you to understand how Hazelcast can provide an advantage. As RAM prices continue to drop, increased use of in-memory technologies will be more practical. Newer technologies like the Intel Optane PMem that deliver high speed and cost-effective deployments will further encourage the use of in-memory technologies to build high-speed applications.

If you are looking for a caching technology, or even a broader applications platform to accelerate your systems, please visit us at www.hazelcast.com or contact us at info@hazelcast.com to get more information.