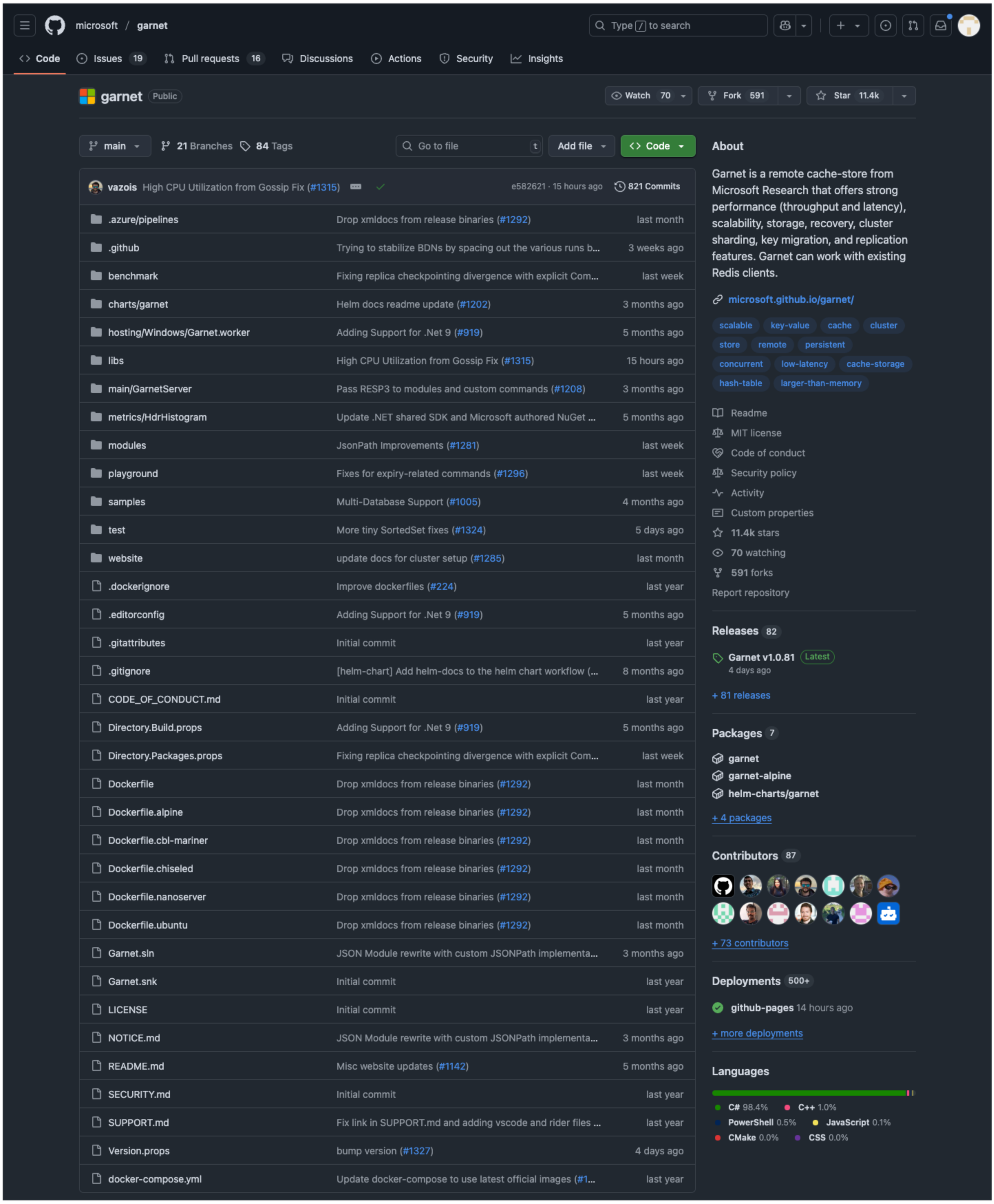
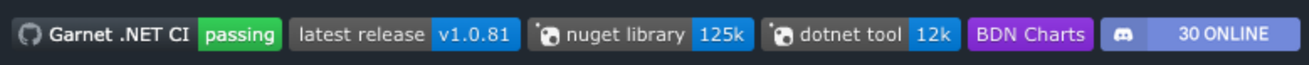
microsoft/garnet: Garnet is a remote cache-store from Microsoft Research that offers strong performance (throughput and latency), scalability, storage, recovery, cluster sharding, key migration, and replication features. Garnet can work with existing Redis clients.
https://github.com/microsoft/garnet



README | Code of conduct | MIT license | Security

Garnet



Garnet is a new remote cache-store from Microsoft Research, that offers several unique benefits:

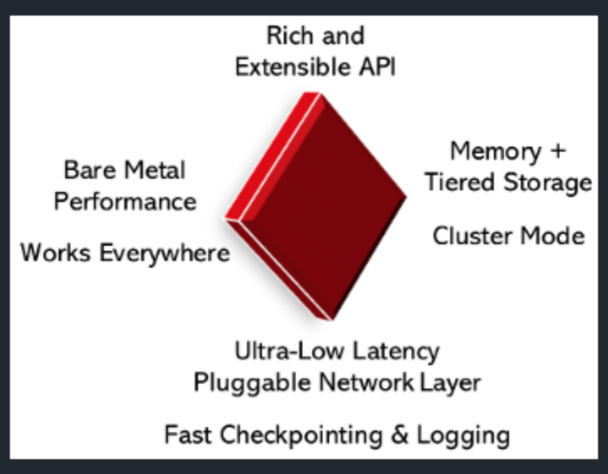
- Garnet adopts the popular RESP wire protocol as a starting point, which makes it possible to use Garnet from unmodified Redis clients available in most programming languages of today, such as StackExchange.Redis in C#.
- Garnet offers much better throughput and scalability with many client connections and small batches, relative to comparable open-source cache-stores, leading to cost savings for large apps and services.
- Garnet demonstrates extremely low client latencies (often less than 300 microseconds at the 99.9th percentile) using commodity cloud (Azure) VMs with Accelerated Networking enabled, which is critical to real-world scenarios.
- Based on the latest .NET technology, Garnet is cross-platform, extensible, and modern. It is designed to be easy to develop for and evolve, without sacrificing performance in the common case. We leveraged the rich library ecosystem of .NET for API breadth, with open opportunities for optimization. Thanks to our careful use of .NET, Garnet achieves state-of-the-art performance on both Linux and Windows.

This repo contains the code to build and run Garnet. For more information and documentation, check out our website at https://microsoft.github.io/garnet.

Feature Summary

Garnet implements a wide range of APIs including raw strings (e.g., gets, sets, and key expiration), analytical (e.g., HyperLogLog and Bitmap), and object (e.g., sorted sets and lists) operations. It can handle multi-key transactions in the form of client-side RESP transactions and our own server-side stored procedures and modules in C#. It allows users to define custom operations on both raw strings and custom object types, all in the convenience and safety of C#, leading to a lower bar for developing custom extensions. Garnet also supports Lua scripts.

Garnet uses a fast and pluggable network layer, enabling future extensions such as leveraging kernel-bypass stacks. It supports secure transport layer security (TLS) communications using the robust SslStream library of .NET, as well as basic access control. Garnet's storage layer, called Tsavorite, is built for high performance and includes strong database features such as thread scalability, tiered storage support (memory, SSD, and cloud storage), fast non-blocking checkpointing, recovery, operation logging for durability, multi-key transaction support, and better memory management and reuse. Finally, Garnet supports a cluster mode of operation with support for sharding, replication, and dynamic key migration.

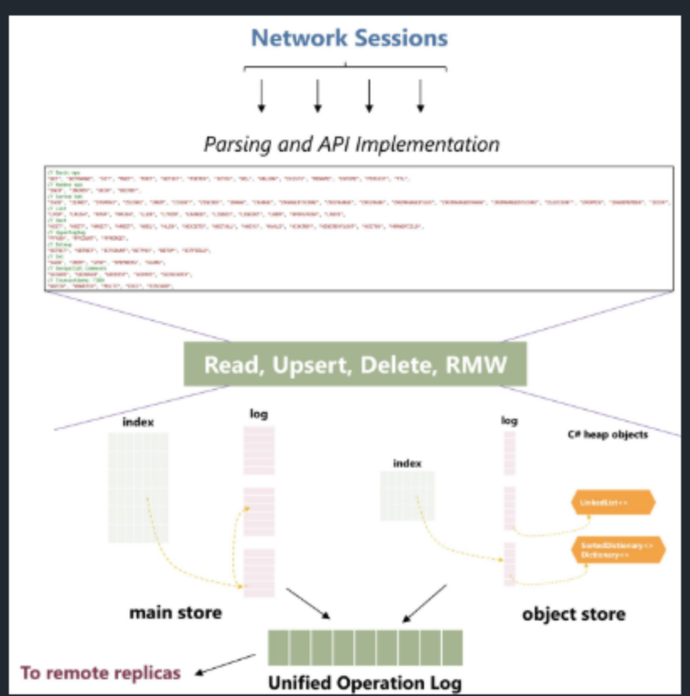


Performance Preview

We illustrate a few key results on our website comparing Garnet to leading open-source cache-stores.

Design Highlights

Garnet's design re-thinks the entire cache-store stack – from receiving packets on the network, to parsing and processing database operations, to performing storage interactions. We build on top of years of our prior research. Below is Garnet's overall architecture.



Garnet's network layer is based on a shared memory design, with TLS processing and storage interactions performed on the network IO completion thread, avoiding thread switching overheads in the common case. This approach allows CPU cache coherence to bring the data to the processing logic, instead of traditional shuffle-based network designs, which require data movement to the appropriate shard on the server.

Garnet's storage design consists of two Tsavorite key-value stores whose fates are bound by a unified operation log. The first store, called the "main store," is optimized for raw string operations and manages memory carefully to avoid garbage collection. The second, and optional, "object store" is optimized for complex objects and custom data types, including popular types such as Sorted Set, Set, Hash, List, and Geo. Data types in the object store leverage the .NET library ecosystem for their current implementations. They are stored on the heap in memory (which makes updates very efficient) and in a serialized form on disk. In the future, we plan to investigate using a unified index and log to ease maintenance.

A distinguishing feature of Garnet's design is its narrow-waist Tsavorite storage API, which is used to implement the large, rich, and extensible RESP API surface on top. This API consists of read, upsert, delete, and atomic read-modify-write operations, implemented with asynchronous callbacks for Garnet to interject logic at various points during each operation. Our storage API model allows us to cleanly separate Garnet's parsing and query processing concerns from storage details such as concurrency, storage tiering, and checkpointing. Garnet uses two-phase locking for multi-key transactions.

Cluster Mode

In addition to single-node execution, Garnet has a fully-featured cluster mode, which allows users to create and manage a sharded and replicated deployment. Garnet also supports an efficient and dynamic key migration scheme to rebalance shards. Users can use standard Redis cluster commands to create and manage Garnet clusters, and nodes perform gossip to share and evolve cluster state. Garnet's cluster mode design is currently *passive*: this means that it does not implement leader election, and simply responds to cluster commands issued by a user-provided *control plane*; see this link for details.

Next Steps

Head over to our documentation site, or jump directly to the getting started or releases section.

License

This project is licensed under the MIT License, see the LICENSE file.

Privacy

Privacy information can be found at https://privacy.microsoft.com/en-us/.

Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit https://cla.opensource.microsoft.com.

When you submit a pull request, a CLA bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., status check, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

Trademarks

This project may contain trademarks or logos for projects, products, or services. Authorized use of Microsoft trademarks or logos is subject to and must follow Microsoft's Trademark & Brand Guidelines. Use of Microsoft trademarks or logos in modified versions of this project must not cause confusion or imply Microsoft sponsorship. Any use of third-party trademarks or logos are subject to those third-party's policies.

Redis is a registered trademark of Redis Ltd. Any rights therein are reserved to Redis Ltd. Any use by Microsoft is for referential purposes only and does not indicate any sponsorship, endorsement or affiliation between Redis and Microsoft.