01 Memorystore for Redis -KirkYagami 👨‍💻🕵️

Google Cloud Memorystore

> Memorystore for Redis Cluster is a fully managed Redis service for Google Cloud. Applications running on Google Cloud can achieve extreme performance by leveraging the highly scalable, available, secure Redis service without the burden of managing complex Redis deployments.

> **Redis** : Remote Dictionary Server
> https://cloud.google.com/memorystore/docs/redis/memorystore-for-redis-overview
> https://en.wikipedia.org/wiki/Redis

Redis in-depth:

1. https://medium.com/@ayushsaxena823/what-is-redis-and-how-does-it-work-cfe2853eb9a9
2. https://medium.com/nerd-for-tech/understanding-redis-in-system-design-7a3aa8abc26a
3. https://medium.com/codex/7-redis-features-you-might-not-know-bab8c9beb2c

1. Introduction to Memorystore

- **Definition**: Memorystore is a fully managed, in-memory data store service on Google that supports Redis and Memcached. It provides low-latency, high-throughput performance, making it ideal for caching, real-time analytics, session management, and other use cases where fast access to data is crucial.
- **Types of Memorystore**:
  - **Memorystore for Redis**: A Redis-compatible in-memory data store known for its simplicity and rich feature set, including persistence, replication, and built-in data structures.
  - **Memorystore for Memcached**: A Memcached-compatible service designed for high-speed caching of small chunks of data.

2. Key Features of Memorystore

- **Fully Managed**: Google handles the infrastructure, patching, scaling, and failure recovery, allowing you to focus on developing your application.
- **In-Memory Caching**: Memorystore provides extremely low-latency data access by storing data in memory, ideal for applications requiring quick lookups and fast data retrieval.
- **Deploy what fits your needs.** Memorystore for Redis allows you the flexibility to choose from different service tiers and sizes that fit your performance and operational needs.

With a few clicks, you have the option to deploy a Basic Tier standalone Redis instance or a Standard Tier high availability Redis instance up to 300 GB.

- **Easily scale to get blazing speed.** With Memorystore for Redis, you can easily achieve your latency and throughput targets by scaling up your Redis instances with minimal impact to your application's availability. Start with the lowest tier and smallest size, then grow your Redis instance as the needs of your application change. For applications that need scaling of read queries, you can scale the queries across five read replicas using the read endpoint.
- **Highly available and more secure.** Redis instances are protected from the internet using private IPs and are further secured using Identity and Access Management role-based access control and in-transit encryption. Standard high availability instances provide up to five replicas replicated across zones and provide a 99.9% availability SLA.
- **Focus on your application.** Memorystore for Redis automates the complex operational tasks that are required to deploy and manage Redis. Tasks like provisioning, replication, failover, and monitoring are all automated. Applications connect to a single endpoint, which simplifies management and operations. Additionally, integration with Cloud Monitoring makes it easy to monitor your Redis instances.
- **Redis Protocol Compatible.** Memorystore for Redis is fully Redis protocol compliant. You can move your applications using open source Redis to use Memorystore for Redis without any code changes. There is no need to learn new tools: all existing tools and client libraries just work.

3. Memorystore for Redis

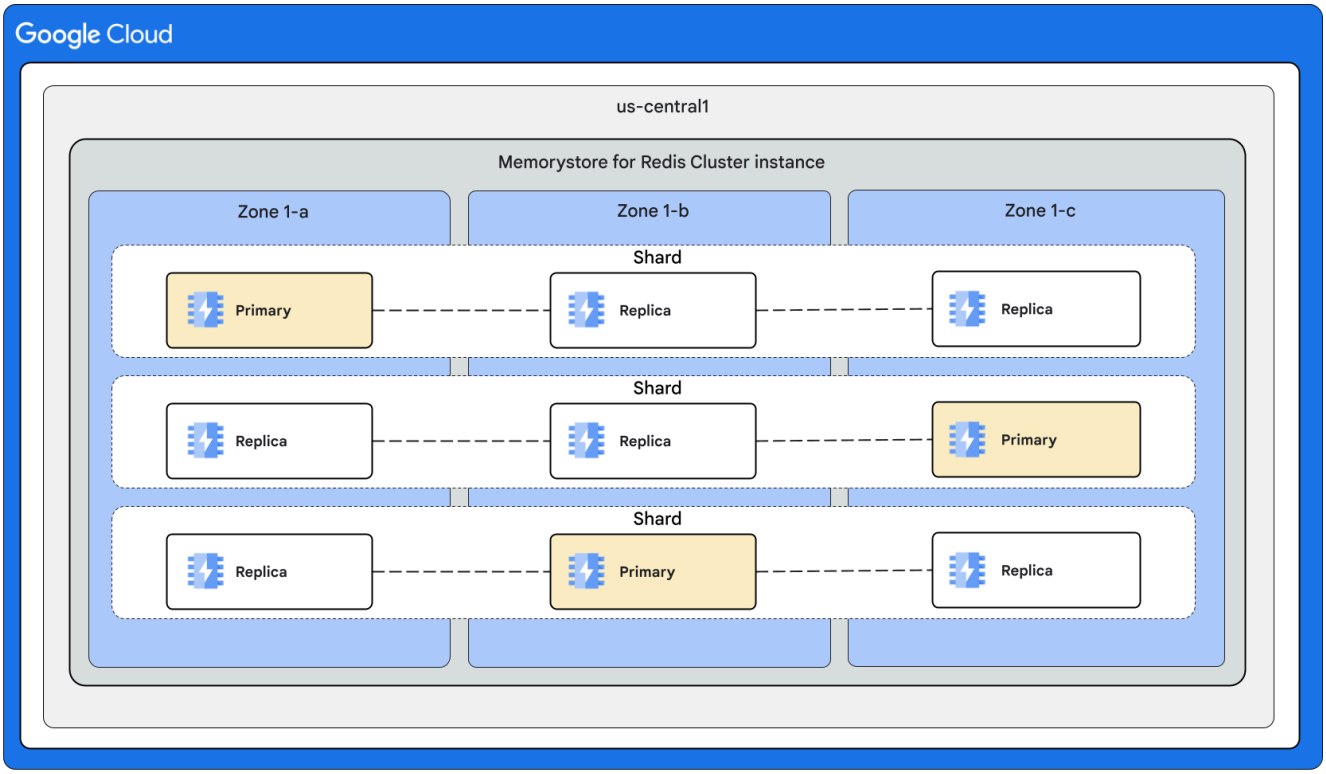
- **Key Data Structures**: Redis supports several advanced data structures such as strings, lists, sets, sorted sets, hashes, bitmaps, and hyperloglogs. These are used for a wide range of applications including caching, message brokering, and session management.
- **Persistence Options**:
  - **Snapshotting**: Redis can periodically take snapshots of the dataset and store them on disk (RDB – Redis Database Backup).
  - **Append-Only File (AOF)**: Redis can append every write operation to a log, ensuring durability with minimal data loss in case of failure.
- **Replication**: Redis supports replication to ensure data availability and reliability. Memorystore for Redis automatically manages replication across zones, providing fault tolerance and read scalability.
- **Failover**: With **Redis HA** in place, Memorystore can automatically failover to a replica in case the primary node goes down.
- **Eviction Policies**: Redis uses eviction policies to manage memory by expiring keys when memory is full. You can configure different strategies like LRU (Least Recently Used), LFU (Least Frequently Used), or no eviction.

4. Memorystore for Memcached

- **Simple Caching**: Memcached is a lightweight, high-performance caching system best suited for ephemeral caching (data that can be easily rebuilt) where persistence and advanced data structures are not required.
- **Horizontal Scaling**: Memorystore for Memcached allows you to scale your cache horizontally by adding or removing nodes. It's well-suited for large-scale distributed caching, such as caching results of expensive database queries or API calls.
- **Auto Discovery**: Memorystore for Memcached automatically manages cluster configurations, so clients can dynamically locate the correct Memcached node, ensuring high availability and distribution.

5. Memorystore Architecture

- **Redis**:
  - **Single and Multi-Zone Deployment**: For Redis instances, you can opt for either single-zone (lower cost, no failover) or multi-zone (automatic failover) deployments.
  - **Primary and Replica Nodes**: In multi-zone Redis configurations, Memorystore automatically promotes a replica to a primary in case of failure. Data is replicated synchronously to ensure consistency.
  - **Scaling**: Redis instances can scale vertically (increase in memory size) but do not support horizontal scaling (sharding) natively in Memorystore.
- **Memcached**:
  - **Clustered Architecture**: Memcached operates in a cluster of nodes, where each node is independent. Scaling is achieved by adding more nodes to the cluster.
  - **Data Distribution**: Memcached uses a consistent hashing algorithm to distribute data across multiple nodes in the cluster, making it easy to scale without affecting performance.



6. Use Cases for Memorystore

- **Caching**: Memorystore is widely used for caching database query results, API responses, and HTML fragments to reduce latency and improve performance.
- **Session Management**: Web applications can use Redis to store session data, which requires fast, frequent access with minimal latency.
- **Real-Time Analytics**: Memorystore's low-latency data access makes it suitable for applications that require real-time data processing, such as recommendation engines or event-driven systems.
- **Leaderboard Systems**: Redis is ideal for implementing real-time leaderboards for games or applications due to its support for sorted sets.
- **Distributed Locking**: Redis can be used to implement distributed locks to ensure that only one process accesses a critical section of code at any given time in a distributed environment.
- **Message Queues**: Redis can serve as a lightweight message broker, supporting pub/sub messaging patterns for real-time event notifications or data streams.

What it's good for

Memorystore for Redis provides a fast, in-memory store for use cases that require fast, real-time processing of data. From simple caching use cases to real time analytics, Memorystore for Redis provides the performance you need.

- **Caching:** Cache is an integral part of modern application architectures. Memorystore for Redis provides low latency access and high throughput for heavily accessed data, compared to accessing the data from a disk based backend store. Session management, frequently accessed queries, scripts, and pages are common examples of caching.
- **Gaming:** Gaming is about capturing and keeping the user's attention. One key aspect that keeps users hooked on a game is the leaderboard. Everyone wants to see how they are progressing and where they stand. Making this experience snappy is critical, and with its in-memory store and data structure like Sorted Set, Memorystore for Redis makes it easy to maintain a sorted list of scores while providing uniqueness of elements. Player Profile is another piece of information that can be accessed frequently. Redis hash makes it fast and easy to store and access profile data.
- **Stream Processing:** Whether processing a Twitter feed or stream of data from IoT devices, Memorystore for Redis is a perfect fit for streaming solutions. Combined with Dataflow, Memorystore for Redis provides a scalable, fast in-memory store for storing intermediate data that thousands of clients can access with very low latency.

7. Memorystore Pricing

- **Pricing Factors**:
  - **Memory Size**: Pricing is largely based on the amount of memory allocated to the Redis or Memcached instance.
  - **Network Egress**: Charges apply for data transferred between regions or to external networks.
  - **Instance Type**: Redis instances with high availability (multi-zone) typically cost more than single-zone deployments.
- **Free Tier**: Google Cloud offers a free tier for Redis with 1 GB of memory, which can be beneficial for small-scale applications or development purposes.

8. Security in Memorystore

- **VPC Service Controls**: Memorystore can be integrated with VPC to control network access. You can restrict access to specific IP ranges and ensure that only trusted networks can interact with your instances.
- **IAM Roles**: Memorystore leverages Google Cloud IAM roles to manage permissions. By assigning specific roles to users or service accounts, you can control who can create, manage, or delete instances.
- **Encryption**: Data in transit between applications and Memorystore is encrypted using **TLS** to protect against interception. Memorystore does not offer encryption at rest since data is stored in memory.

9. Best Practices for Memorystore

- **Use Caching Strategically**: Use Memorystore to cache frequently accessed data such as database query results, API responses, or computed results to reduce expensive

computations.

- **Choose Appropriate Eviction Policies**: Select the appropriate eviction strategy (e.g., LRU or LFU) based on your use case to ensure memory is used efficiently.
- **Monitor and Optimize**: Use Cloud Monitoring and Logging to track the performance of your Memorystore instances. Set up alerts for memory usage, latency, and connection errors to preemptively address potential issues.
- **Right-Size Your Instances**: Start with an instance size that fits your expected workload, then scale vertically as needed. Avoid over-provisioning to minimize costs.
- **Leverage Redis Data Structures**: Use Redis-specific features like sorted sets, lists, and hyperloglogs to efficiently store and manipulate complex data structures.

10. Real-World Example: Improving Web Application Performance with Memorystore

- **Scenario**: A web application fetches product listings from a database. Every query incurs database I/O and increases latency for end users.
- **Solution**:
  - Use Memorystore to cache the results of frequent queries in Redis.
  - Configure Redis with an appropriate eviction policy to automatically remove outdated data.
  - Implement session management using Redis to track user sessions efficiently.
- **Outcome**: The application's latency drops significantly as the majority of requests are served from the Redis cache, reducing the load on the primary database.

11. Memorystore Performance Considerations

- **Memory Size**: Choose the appropriate instance size based on your workload. Larger memory sizes can store more data and reduce eviction rates.
- **Latency and Throughput**: For low-latency, high-throughput applications, Redis is more suitable due to its sophisticated data structures and built-in replication.
- **Persistence**: If persistence is critical, enable Redis' AOF or snapshotting features to ensure data is recoverable after restarts. Memcached does not support persistence.
- **Scaling**: Redis supports vertical scaling, so plan your instance size carefully. Memcached supports horizontal scaling, which is better for large distributed systems that need more memory capacity spread across multiple nodes.

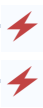
Specifications	Basic Tier	Standard Tier (read replicas disabled)	Standard Tier (read replicas enabled)
Description	Provides a cache with no replication	Provides redundancy and availability using replication	Provides redundancy and availability using replication to backup data, and multiple Read replicas to increase read throughput
Max Redis primary size	300 GB	300 GB	300 GB
Max network bandwidth	16 Gbps	16 Gbps	16 Gbps total for writes. 16 Gbps per node for reads.[1]
I/O threads	Yes[2]	Yes[2]	Yes[2]
Scale primary size	Yes	Yes	Yes
Scale number of read replicas	No	No	Yes
Cross-zone replication	No	Yes	Yes
Automatic failover	No	Yes	Yes
Read replicas	No	No	Yes
In-transit encryption	Yes	Yes	Yes
Maintenance window	Yes	Yes	Yes
Cloud Monitoring	Yes	Yes	Yes

[1] The maximum write throughput is 16 Gbps. Read throughput is dependent on the number of nodes (read replicas) in the instance, including the primary node. For example, if you have an instance with 1 primary node and 2 read replicas, the total read throughput is 48 Gbps.

[2] I/O threads are only available on M3 instances or higher running Redis version 6.x. For more information, see Redis version 6.x.

12. Conclusion

Memorystore, with its Redis and Memcached options, is a powerful in-memory data store that helps deliver low-latency, high-performance applications. Its flexibility, scalability, and ease of use make it a valuable tool for use cases like caching, session management, and real-time data processing. As a managed service, it abstracts the complexities of infrastructure management, allowing developers to focus on building efficient and fast applications while benefiting from high availability, security, and seamless integration with Google Cloud.



https://cloud.google.com/memorystore/docs/redis/create-instance-gcloud

Creating a Memorystore for Redis Instance

To create a Memorystore for Redis instance:

1. Open a terminal window.
2. Set the project you'd like to create your instance in as the default project in gcloud by entering the following command:

```
gcloud config set core/project PROJECT_ID
```

3. Enter the following command to create a 2 GiB Basic Tier Redis instance in the `us-central1` region:

```
gcloud redis instances create myinstance --size=2 --region=us-central1 \
    --redis-version=redis_6_x
```

4. After the instance is created, use the `describe` command to get the IP address and port of the instance:

```
gcloud redis instances describe myinstance --region=us-central1
```

If successful, gcloud returns the following information:

```
authorizedNetwork: projects/my-project/global/networks/default
createTime: '2018-04-09T21:47:56.824081Z'
currentLocationId: us-central1-a
host: 10.0.0.27
locationId: us-central1-a
memorySizeGb: 2
name: projects/my-project/locations/us-central1/instances/myinstance
networkThroughputGbps: 2
port: 6379
redisVersion: REDIS_6_X
reservedIpRange: 10.0.0.24/29
state: READY
tier: BASIC
```

Connecting to the Redis instance from a Compute Engine VM

You can connect to the Redis instance from any Compute Engine VM that uses the Redis instance's authorized network with a supported RFC 1918 IP address.

1. If you don't already have a Compute Engine VM that uses the same authorized network as your Redis instance, create one and connect to it by following Quickstart using a Linux VM.
2. Install `telnet` using `apt-get`:

```
sudo apt-get install telnet
```

3. From the terminal, telnet to the IP address of the Redis instance, replacing variables with appropriate values:

```
telnet instance-ip-address 6379
```

4. If successful, the command will return this result:

```
Trying instance-ip-address…
Connected to instance-ip-address
```

5. In the telnet session, enter some Redis commands:

   - Ping the Redis instance:

     ```
     PING
     ```

     Result:

     ```
     PONG
     ```

   - Set a key:

     ```
     SET HELLO WORLD
     ```

     Result:

     ```
     +OK
     ```

   - Get the key:

     ```
     GET HELLO
     ```

     Result:

     ```
     $5
     WORLD
     ```

Clean up

To avoid incurring charges to your Google Cloud account for the resources used, follow these steps:

1. Delete the instance by entering the following command:

```
gcloud redis instances delete myinstance --region=us-central1
```

2. Enter `Y` to confirm the deletion:

```
You are about to delete instance [myinstance] in [us-central1].
Any associated data will be lost.
Do you want to continue (Y/n)? Y
```

3. If successful, gcloud returns the following response:

```
Deleted instance [myinstance].
```