

Altimetrik



Scale with Redis Cluster - Caching

Author/Architect

ArunSanthoshKumar Annamalai

White Paper



Index

Executive Summary	01	Access the Redis Cluster	13
Industry Trends	02	■ Create the Cluster Client	
Key Statistics Regarding Interest in Redis Clusters	03	■ Set the keys	
Understanding the Redis Cluster	04	■ Access the keys	
What is Redis Cluster			
Benefits of Redis Cluster		Best Practices	14
■ Horizontal Scaling		■ Data Sharding	
■ High Availability		■ Monitoring	
■ Automatic Failover nodes		■ Backup	
■ Data Sharding		■ Security	
Setting Up Redis Cluster	09	Limitation	17
■ Pre-Requisites		Use Cases	17
■ Installation		■ Ecommerce application	
■ Configure nodes		Key take-aways	18
■ Start the Instances		Conclusion	18
■ Create Cluster		References	18
■ Verify Cluster			
Scaling the Redis Cluster	11		
■ Adding the Nodes			
■ Removing the Nodes			

Executive Summary

Redis is an open-source, in-memory data structure store that can be used for caching, databases, and message brokering. As an in-memory store, it delivers significantly higher performance compared to traditional databases. Redis supports a wide range of use cases, including message brokering, key-value storage, caching, databases, Pub/Sub messaging, clustering, and ensuring high availability of data.

In this case study, we will focus on caching, one of the most common use cases of Redis. Redis offers high availability through replication, and its performance can be further enhanced by leveraging its clustering features.



Introduction Redis Clustering

Redis clustering enables scaling by distributing data across multiple Redis instances, allowing the system to handle larger datasets while increasing read and write throughput. Built-in fault tolerance is achieved by replicating data to secondary nodes, and data is partitioned based on hash slots.

Redis scales effectively through clustering, with the application managing redistribution in the event of node failures or additions. Redis also includes Sentinel, a built-in service that handles clustering management. Additionally, Redis clusters can be monitored using Redis Sentinel or various third-party tools.



Industry Trends

Redis is widely regarded as one of the most popular key-value stores in the IT industry, known for its robustness, scalability, and high availability. As the world becomes more competitive with advances in internet speed and technological upgrades, Redis supports high-speed client communication through its in-memory architecture.

Redis clustering has become a critical component of modern infrastructure, especially for real-time data processing in industries such as e-commerce, gaming, analytics, and finance. Organizations are increasingly adopting Redis clusters due to their scalability, performance, and resilience. Industry trends indicate a growing interest in Redis, as reflected in the increasing number of Docker image downloads and widespread adoption of Redis software.

Key Statistics Regarding Interest in Redis Clusters



The Bitnami/Redis-cluster images have been downloaded over 100 million times, reflecting the strong interest and widespread usability of Redis clusters in the software industry.

A significant number of downloads have been recorded for Redis-related APIs and programming package managers.

Redis is among the most downloaded images on Docker Hub.

This is a screenshot of the Docker Hub page for the 'bitnami/redis-cluster' image. The page header shows the Bitnami logo, the repository name 'bitnami/redis-cluster', and statistics: '100M+' downloads, '50' stars, and '1,561,519' pulls from 'Last week'. Below the header, it says 'By VMware · Updated 9 days ago' and 'Bitnami container image for Redis Cluster'. There are tags for 'DATABASES', 'STORAGE', 'MESSAGE QUEUES', and 'SECURITY'. A 'Learn more' link is visible on the right.

npm trends

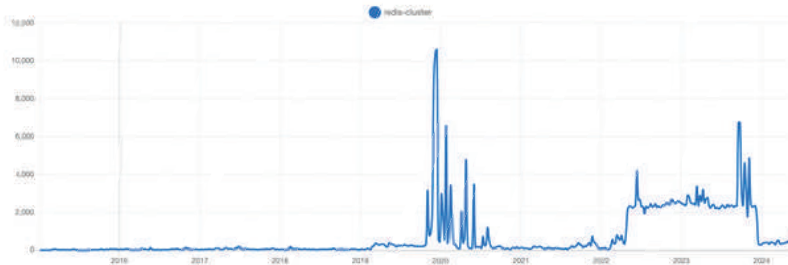
redis-cluster

Redis cluster client library

Enter an npm package...

redis-cluster · redis · newredis · redis · redis-cluster · redis-connection-pool · then-redis · hredis · node-redis · redis-js

Downloads in past All time



Stats

Package	Stars	Issues	Version	Updated	Created	Size
redis-cluster	53	30	0.0.8	9 years ago	12 years ago	1.2 MB



Understanding the Redis Cluster

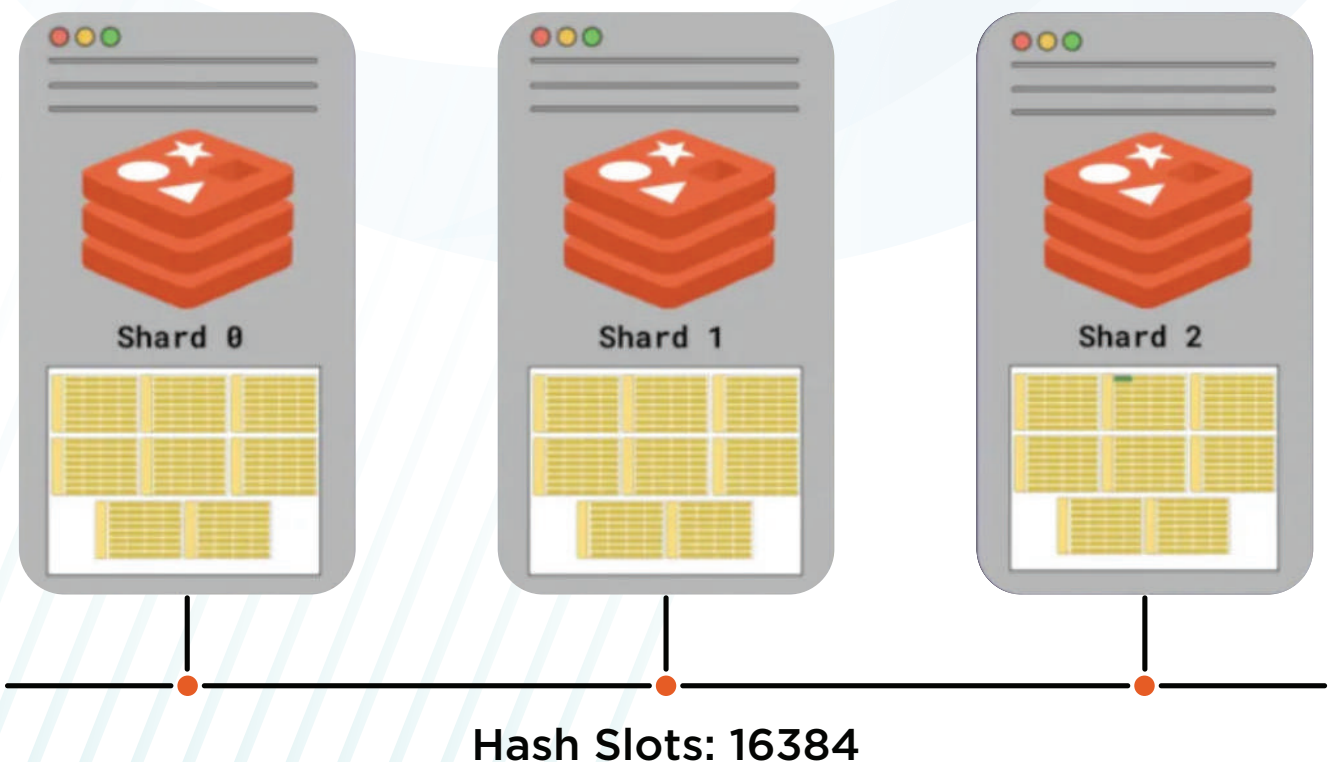
What is Redis Cluster

Redis Cluster is a distributed implementation of Redis that splits data storage across multiple Redis nodes, enabling horizontal scaling and ensuring high availability through automatic failover.

Benefits of Redis Cluster

Redis Cluster provides substantial benefits, including horizontal scaling, high availability, automatic failover, and data sharding. By distributing data and workloads across multiple nodes, it ensures continuous service through replicas and optimizes resource utilization through sharding. Redis Cluster delivers a best-in-class solution for modern, data-intensive applications.

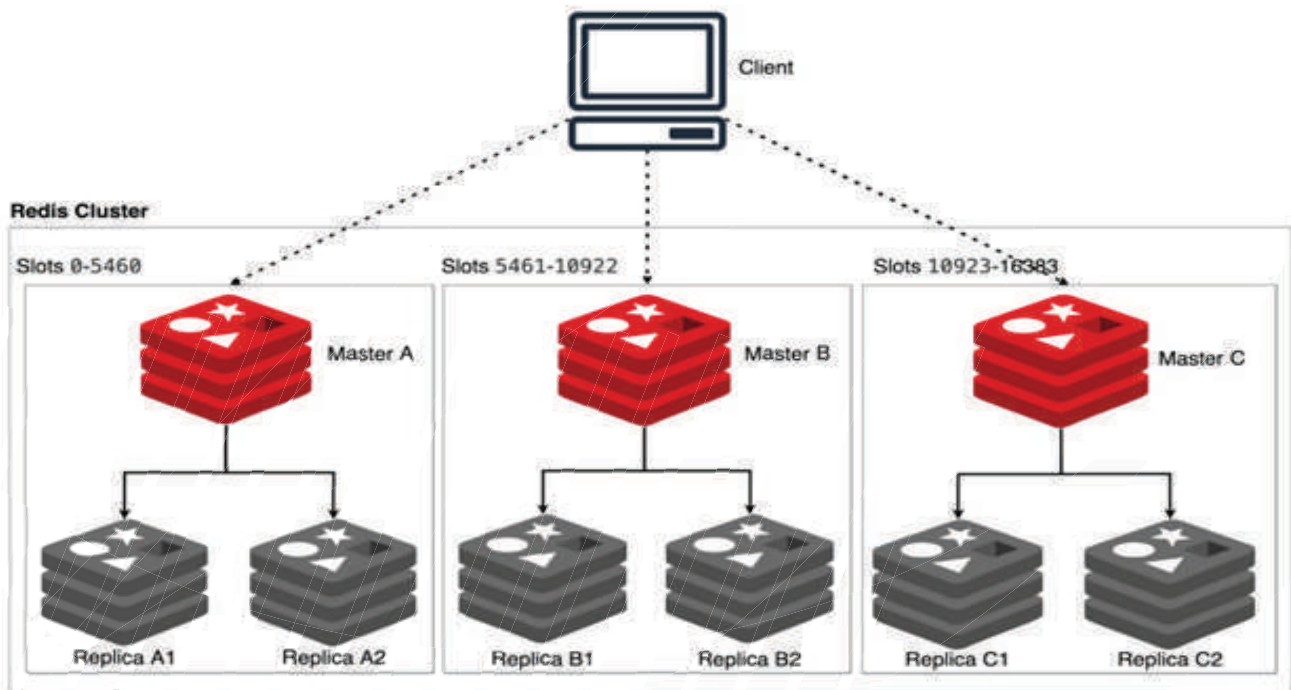
These features make Redis Cluster an excellent choice for applications requiring high performance, scalability, and resilience, particularly when handling large datasets with multiple write points.



Horizontal Scaling

Horizontal scaling architecture enables systems to expand by adding more nodes as needed to accommodate increased workloads and data volumes. By distributing data across multiple nodes, read and write operations are spread out, reducing the load on any single node and improving overall performance—especially in high-traffic enterprise systems, such as e-commerce or financial platforms that require real-time data access.

Nodes and data storage can be added without significant changes to the application architecture, allowing the system to easily scale up or down based on demand. This makes horizontal scaling both cost-effective and efficient for managing fluctuating workloads.



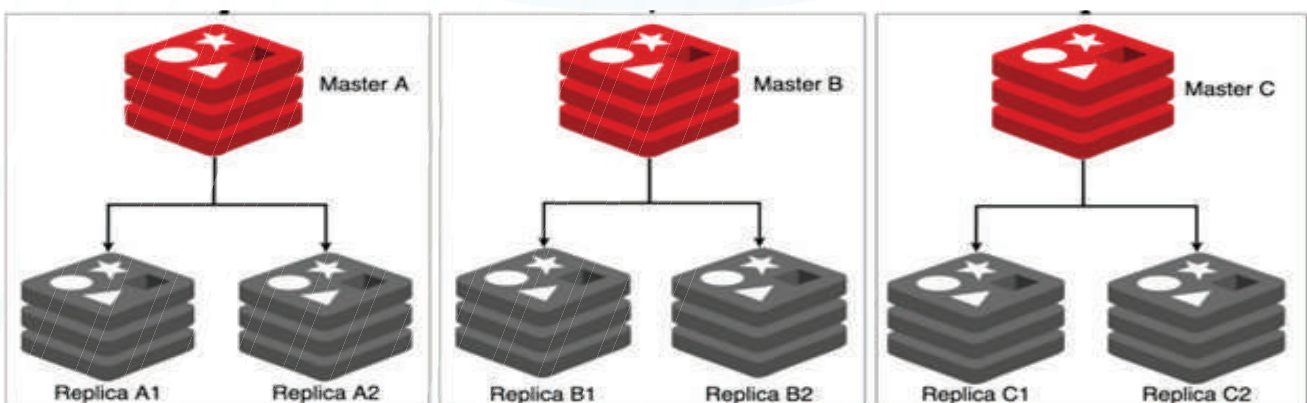


High Availability

Redis is built with replication capabilities, allowing additional instances to be added to ensure failover. Redis Cluster guarantees high availability by maintaining multiple replica nodes for each master node. These replicas serve as backups and can quickly take over if a master node fails. Redis provides synchronization mechanisms to keep replicas up to date, using either full or partial sync depending on data integrity needs.

In the event of a master node failure, replicas are promoted to master, ensuring uninterrupted service with minimal downtime.

Replicas consistently synchronize with the master node, protecting against data loss and maintaining data consistency. This architecture also handles node failures and network partitions, offering robust fault tolerance.

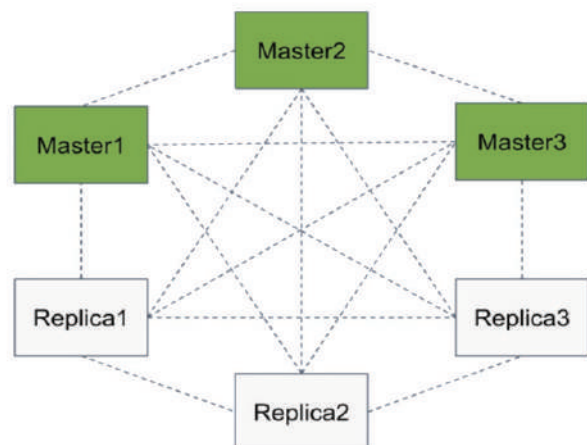




Automatic Failover nodes

Redis Cluster includes an automatic failover mechanism that promotes a replica (slave) to master if the current master node fails. This process occurs automatically, without requiring human intervention, ensuring minimal downtime and maintaining service responsiveness even during node failures.

By eliminating the risk of human error in failover situations, Redis Cluster significantly improves operational efficiency. Automatic failover enhances the reliability of applications by ensuring continuous data access and smooth operation during failures.



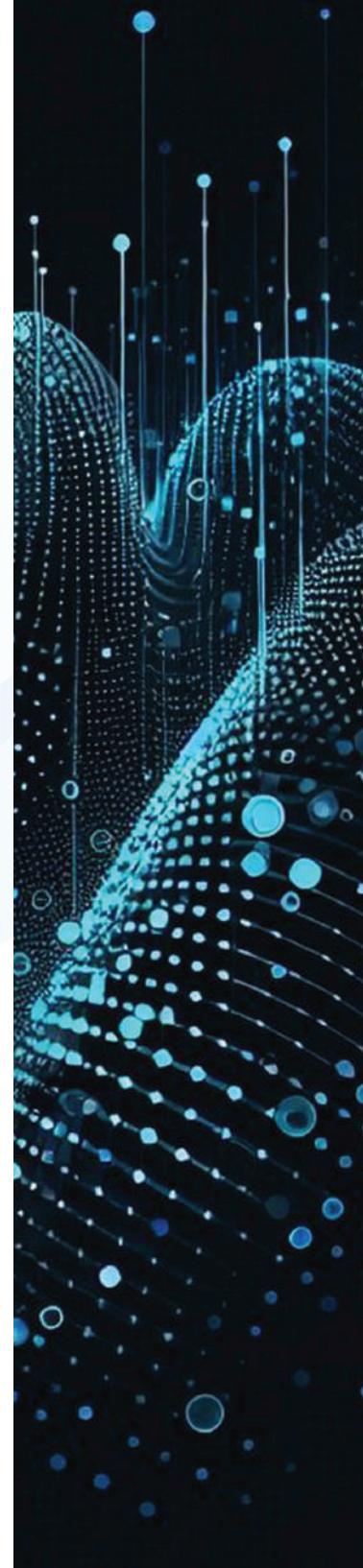
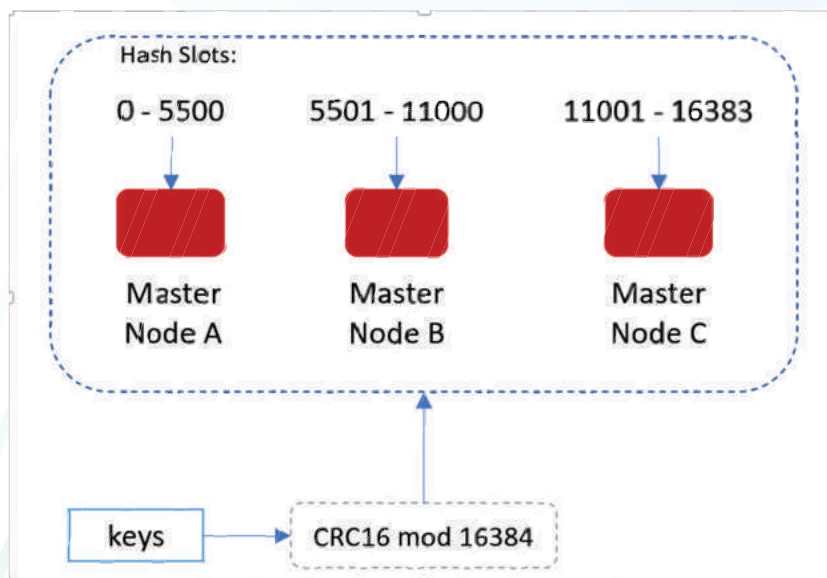


Data Sharding

Sharding involves distributing data across multiple nodes using consistent hashing, which partitions the data into smaller, more manageable chunks spread across the cluster. Redis Cluster employs hash slot-based partitioning, automatically handling sharding or resharding when nodes are added or removed.

By distributing data evenly across multiple nodes, Redis Cluster ensures no single node becomes overloaded, optimizing resource utilization.

Data sharding enables parallel processing of operations within the same application across different nodes, improving overall throughput and reducing latency. As data grows, the cluster can be expanded by adding more nodes, with each node managing a portion of the data, allowing the system to handle larger datasets effortlessly.



Setting Up Redis Cluster

Pre-Requisites

- Multiple instances or multiple nodes or multiple virtual machines.
- Redis should be installed in all machines which will be part of cluster.
- Network configuration to have the communication between nodes.

Installation

Install the same version of Redis sever into the nodes. This can be done by installing using binary download from the Redis releases.

This can be done in various ways. Manual install, Docker install, or the helm install.

Configure nodes

By default, Redis configuration for single instance application. To make it Redis cluster we need to modify to enable Redis cluster.

Minimally we need to make these config changes on the custom redis.conf file.

```
port 6379
```

```
cluster-enabled yes
```

```
cluster-config-file redis.conf
```

```
cluster-node-timeout 5000
```

```
appendonly yes
```

Start the Instances

Stat the Redis instance with parameterised the conf file updated with Redis cluster configuration.

Use the redis.conf file to start the redis instance in each node of the cluster.

```
redis-server /path/to/redis.conf
```

Create Cluster

Once you have all the nodes up and running, (with Redis server) create cluster in master server.

Create helm install the repo

```
[aannamalai@aannamalais-MacBook-Pro:~ $ helm repo list
NAME                URL
mybitnami           https://charts.bitnami.com/bitnami
datadog              https://helm.datadoghq.com
myairflow            https://airflow.apache.org
aannamalai@aannamalais-MacBook-Pro:~ $
```

Using the helm install the Redis-cluster.

```
([env] aannamalai@aannamalais-MacBook-Pro:~/Desktop/workspace/sco.redis.cluster $ helm install redis-cluster mybitnami/redis-cluster -n redis)
NAME: redis-cluster
LAST REPLOYED: Mon Jun  3 12:01:30 2024
NAMESPACE: yed19
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: redis-cluster
CHART VERSION: 10.8.2
APP VERSION: 7.2.4** Please be patient while the chart is being deployed **

To get your password run:
  osport REDIS_PASSWORD=$(kubectl get secret --namespace "redis" redis-cluster -o jsonpath="{.data.redis-password}" | base64 -d)

You have deployed a Redis®; Cluster accessible only from within you Kubernetes Cluster.INFO: The Job to create the cluster will be created.To connect to your Redis®; cluster:

1. Run a Redis®; pod that you can use as a client:
  kubectl run --namespace redis redis-cluster-client --rm --tty -i --restart='Never' \
  --env REDIS_PASSWORD=$REDIS_PASSWORD \
  --image docker.io/bitnami/redis-cluster:7.2.4-debian-12-r12 -- bash

2. Connect using the Redis®; CLI:

redis-cli -c -h redis-cluster -a $REDIS_PASSWORD

WARNING: There are "resources" sections in the chart not set. Using "resourcesPresets" is not recommended for production. For production installations, please set the following values according to your workload needs:
- redis.resources
- updateJob.resources
sinfo https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```





Verify Cluster

Verify the cluster using the cluster commands.

redis-cli -c cluster nodes

```

[[venv] @hostname:~$ ssh root@192.168.1.100 -i ~/.ssh/id_rsa -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null redis-cluster
Defaulted container 'redis-cluster-client' out of: redis-cluster-client, datadog-lib-luby-init (init), datadog-lib-docker-init (init), datadog-lib-python-in
it (init), datadog-lib-js-init (init), datadog-lib-java-init (init)
I have no fans@redis-cluster-client:/S ps -oat
  UID          PID     PPID    C  STIME  TTY          TIME CMD
  1001           1      0  0 00:33 pts/0      00:00:00 bash
  1001          13      0  0 00:34 pts/1      00:00:00 /bin/bssh
  1001          18     13  99 00:34 pts/1      00:00:00 ps -oat
I have no fans@redis-cluster-client:/S redis-cli -c -h redis-cluster -a $REDIS_PASSWORD
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
redis-cluster:63799 cluster nodes
733eao573rad782c560022798377f00a2aed750 10.42.0.188:6379@16379 slave 8009a0e90357fa1f98a968081264a50124fa2001 0 1717396497152 2 connected
8009a0e90357fa1f98a968081264a50124fa2001 10.42.0.188:6379@16379 myself,master - 0 1717396497000 2 connected 5461-18922
70121c3c08ad2800565f7c7856e0a06211a4220 10.42.0.187:6379@16379 master - 0 1717596496144 1 connected 8-5409
0915509ad0cccaed0ccca0010e20a48e0039d 10.42.0.192:6379@16379 slave f0121c9ca00d20e8364f92c90896b40b1e14012c 0 1717396498135 1 connected
302aed1770a5433278500027abc938324a0322e 10.42.0.193:6379@16379 slave 0fbac97a0db06cdea6c02e972af022798978039 0 1717396497000 3 connected
p764c97eb0ed0cde05e023972a7022798978039 10.42.0.189:6379@16379 master - 0 1717396498158 3 connected 18923-15393

```

Scaling the Redis Cluster

Here is the cluster with **6** replicas and each of the **3** masters to have one slave.

```

[[root] @hostname:~$ ssh root@192.168.1.100 -i ~/.ssh/id_rsa -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null redis-cluster
redis-cluster:63799 cluster nodes
NAME                REPOV  STATUS  RESTARTS  AGE
pod/redis-cluster-5  1/1    Running  0          10m
pod/redis-cluster-4  1/1    Running  0          1m
pod/redis-cluster-6  1/1    Running  0          1m
pod/redis-cluster-1  1/1    Running  0          10m
pod/redis-cluster-2  1/1    Running  0          1m
pod/redis-cluster-3  1/1    Running  0          1m
pod/redis-cluster-client  1/3    Running  0          1m

NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/redis-cluster-headless  ClusterIP    None             <none>            6379/ICP,16379/ICP  10m
service/redis-cluster             ClusterIP    10.65.148.112   <none>            6379/ICP          10m

NAME                READY     AGE
redis-cluster-headless  6/6      30s

```


Set the keys

Based on the keys hash value, keys are redirected to relevant slots.

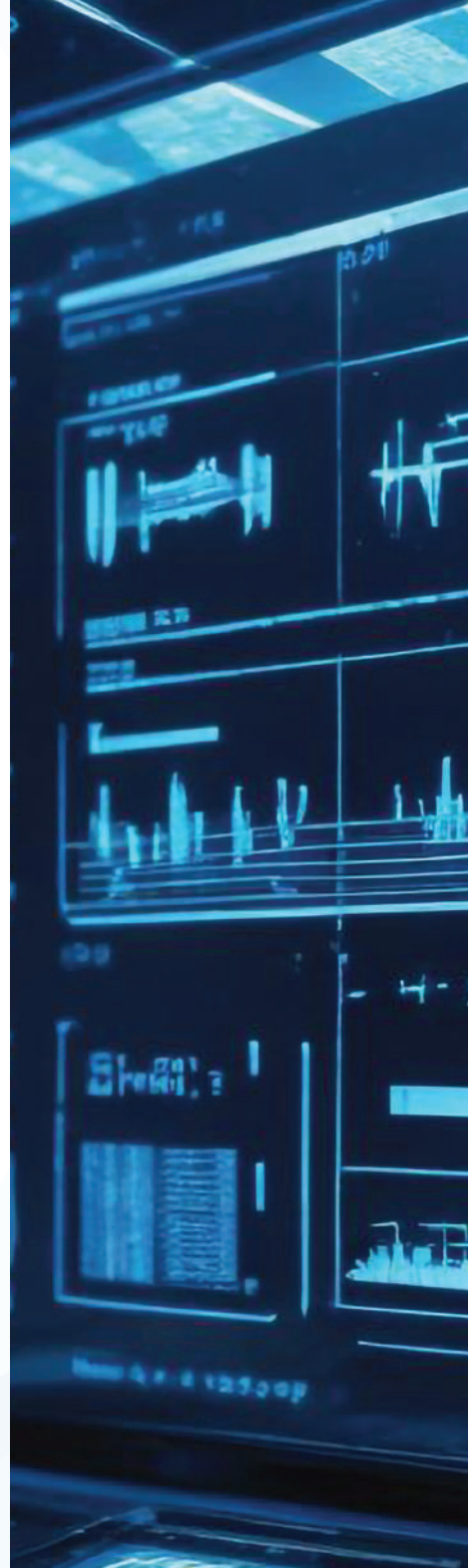
```
10.42.0.189:6379> set bar bar
-> Redirected to slot [6861] located at 10.42.0.187:6379
OK
10.42.0.187:6379> set car bmw
-> Redirected to slot [9461] located at 10.42.0.188:6379
OK
10.42.0.188:6379> set foo foo
-> Redirected to slot [12182] located at 10.42.0.189:6379
OK
10.42.0.189:6379> █
```

Access the keys

Keys are retrieved from the relevant slots.

```
10.42.0.189:6379> get foo
"foo"
10.42.0.189:6379> get bar
-> Redirected to slot [6861] located at 10.42.0.187:6379
"bar"
10.42.0.187:6379> get car
-> Redirected to slot [9461] located at 10.42.0.188:6379
"bmw"
10.42.0.188:6379> █
```

Keys are retrieved from the relevant slots.



Best Practices

Redis is a high-performance application with sub-millisecond response times. Due to its performance capabilities, careful consideration is required when configuring it for production. It's essential to implement consistent algorithms for data sharding, maintain real-time observability to monitor patterns, establish a proper backup strategy to ensure data integrity, and secure access to the application.

Data Sharding

- Design Redis keys to distribute data evenly across all shards.
- Use consistent hashing to minimize data movement when adding or removing nodes.
- Utilize hash tags (delimited by “{ }”) in keys so that related keys are stored on the same shard.



Monitoring

Monitoring is critical for maintaining the health and performance of a Redis Cluster. With proper monitoring, issues can be detected early on.

- Use Redis Sentinel, a high-availability component, to automate monitoring and manage the failover process.
- Configure exporters and integrate Redis with external monitoring systems like Prometheus, Grafana, or Datadog to track key metrics such as CPU usage, memory consumption, cache hit/miss ratios, and latency.
- Implement alerting for critical events such as node failures, high memory usage, and increased latency to proactively address potential issues

Backup

Redis is an in-memory data store, where application data is stored in RAM. However, Redis provides file-based backup options.

- Leverage Redis snapshots to create point-in-time copies of the dataset. You can adjust the configuration to trigger backups based on time intervals or the number of writes.
- Utilize the Append Only File (AOF) to log every write operation, ensuring durability and offering a complete recovery mechanism.
- Define Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO), and regularly test your backup and recovery processes.

Security

Protecting Redis instances from unauthorized access, data breaches, and security threats is crucial.

- Apply firewall restrictions at the port level and allow only trusted IP addresses and networks to connect to Redis.
- Implement role-based access control to define user roles and permissions, restricting sensitive commands based on user roles.
- Keep Redis software up to date with the latest security patches and updates, following Redis documentation and release notes.
- Enable authentication for Redis instances to safeguard against unauthorized access.



Limitations

- Redis clustering only supports db0, compared to the default Redis instance, which supports db0 to db15, providing 16 databases in total.
- Redis Cluster does not support Pub/Sub due to potential issues with hashing patterns. It's not recommended to use Pub/Sub commands on cluster instances.
- Multi-key operations are restricted in Redis Cluster.

Use Cases

E-commerce Application

One use case involves an e-commerce client with a large product catalog featuring multiple SKUs and product add-ons. E-commerce traffic is often driven by events, marketing campaigns, and product launches, leading to fluctuating demand. The client can scale the service by adding more nodes to handle increased traffic. If the web server uses load balancing, it funnels parallel requests to caching, databases, or external APIs.

Typically, web servers and databases respond within 1 to 100 milliseconds, while in-memory caching provides response times as fast as 1 to 100 nanoseconds. During high-traffic conditions, when multiple users access various products simultaneously, requests are queued, and caching helps reduce the load on databases. With a caching system backed by clustering, data reads are distributed across multiple nodes, utilizing different CPU cores and data storage, instead of relying on a single instance.

After implementing Redis Cluster, the client's system was upgraded to handle more traffic during peak demand, redefining application scalability.





Key Takeaways

Redis Cluster provides high availability and horizontal scalability to accommodate growing data and traffic. By implementing Redis Cluster for caching, frequently accessed data is stored in-memory and distributed, reducing database load and improving response times. Additionally, Redis manages replication and high availability, making it an ideal solution for modern IT applications.

Conclusion

Redis Cluster offers a robust solution for scalable and resilient caching. By distributing data across multiple nodes, it enhances performance and ensures high availability. For optimal results, it is essential to follow best practices and implement strong security measures. Enabling Redis Cluster can significantly boost application performance and reliability.

References

- Redis Documentation: redis.io/documentation
- Redis Cluster Tutorial: redis.io/topics/cluster-tutorial
- Redis GitHub Repository: github.com/redis/redis

This whitepaper will help as guide for Analysing and implementation of the Redis cluster for scalable caching solutions, providing detailed steps and best practices to ensure a successful deployment.

About Altimetrik

Altimetrik is a pure play digital business company focused on delivering business outcomes with an agile, product-oriented approach. Our digital business methodology provides a blueprint to develop, scale, and launch new products to market faster. Our team of 5,000+ employees with software, data, cloud engineering skills help create a culture of innovation and agility that optimizes team performance, modernizes technology, and builds new business models. As a strategic partner and catalyst, Altimetrik quickly delivers results without disruption to the business.

Our unique Digital Business Methodology centers on business led ownership aligned to company goals. It is comprised of three pillars: experienced team of practitioners, an incremental approach, and an end-to-end self-service digital business platform. These combine to facilitate collaboration and agility between business and engineering teams to co-create products and solutions faster without disruption to the business. This is powered by a single source of truth and a culture of innovation that brings unlimited growth within reach.

We cater to companies of all sizes from Fortune 100 to digital disruptors and start-ups. We are a people-centric organization and talent is one of the central pillars of our business model and success. Employee engagement, diversity & inclusion, well-being and empowerment are central themes to our ethos. This project has been instrumental in taking our digital culture to the next level and brings our globally spread employee base on a unified platform.