Remote Dictionary Server (Redis)

Waheed Iqbal

DevOps (Fall 2023)
Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan

Introduction

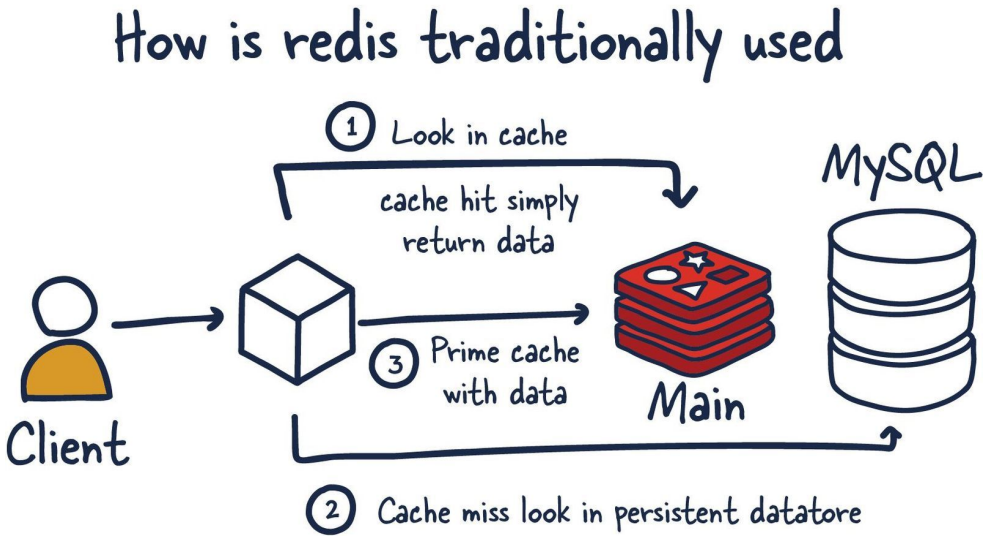
- Redis (short for Remote Dictionary Server) is an open-source, in-memory data store that is used as a database, cache, and message broker.
- It was designed to be highly performant and scalable, making it popular among developers who need to handle large amounts of data quickly and efficiently.
- Redis is a key-value database, which means that it stores data as key-value pairs.
- Each key is associated with a value, and values can be strings, hashes, lists, sets, or sorted sets.
- Redis is well-suited for applications that require high throughput and low latency, such as real-time applications, messaging systems, and web applications.

Introduction (Cont.)

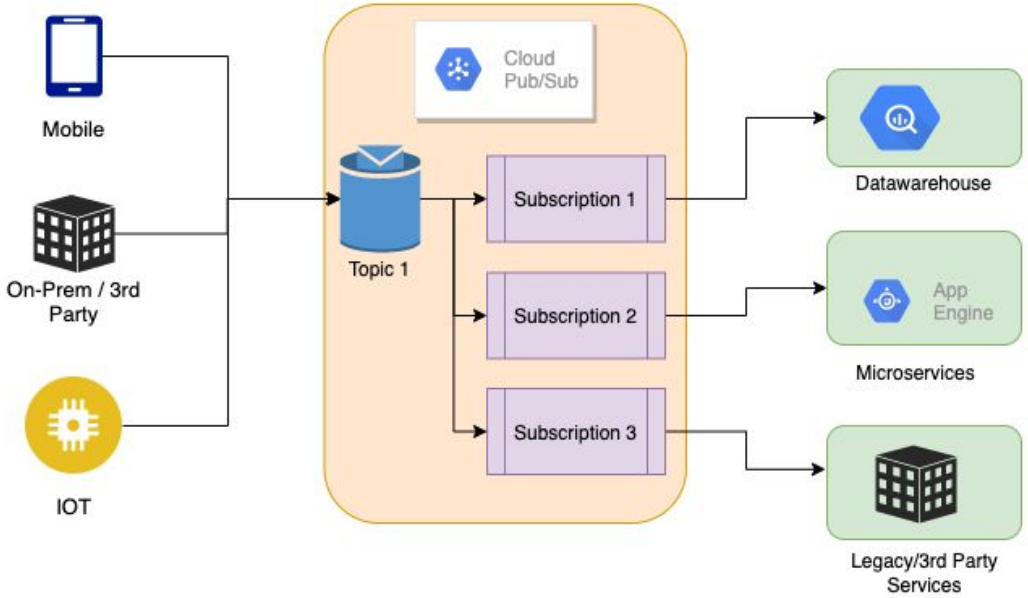
- Redis is an in-memory database, which means that all data is stored in RAM. However, Redis also supports persistence, which allows data to be written to disk periodically or on every write operation.
- Redis is often used as a cache because it can store frequently accessed data in memory, which can significantly improve the performance of an application.
- Redis also includes features such as transactions and pub/sub messaging, which makes it a popular choice for building real-time applications and implementing message queues.

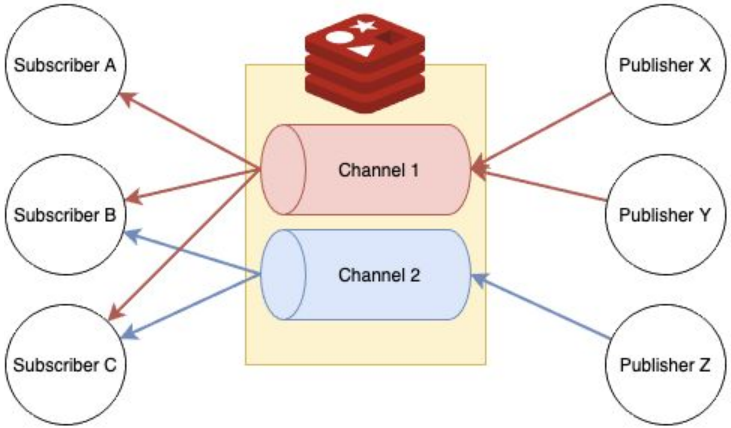
Redis Use Cases: Cache

1. When a user requests data from the application, the application checks if the data is available in Redis cache.
2. If the data is available in Redis cache, the application retrieves it from Redis and returns it to the user. This avoids the need to query the MySQL database, which can be slower and more resource-intensive.
3. If the data is not available in Redis cache, the application queries the MySQL database to retrieve the data, and then stores the data in Redis cache for future requests.
4. When the data in the MySQL database changes, the application updates the data in Redis cache to ensure that the cached data is up-to-date.



Redis Use Cases: Pub-Sub

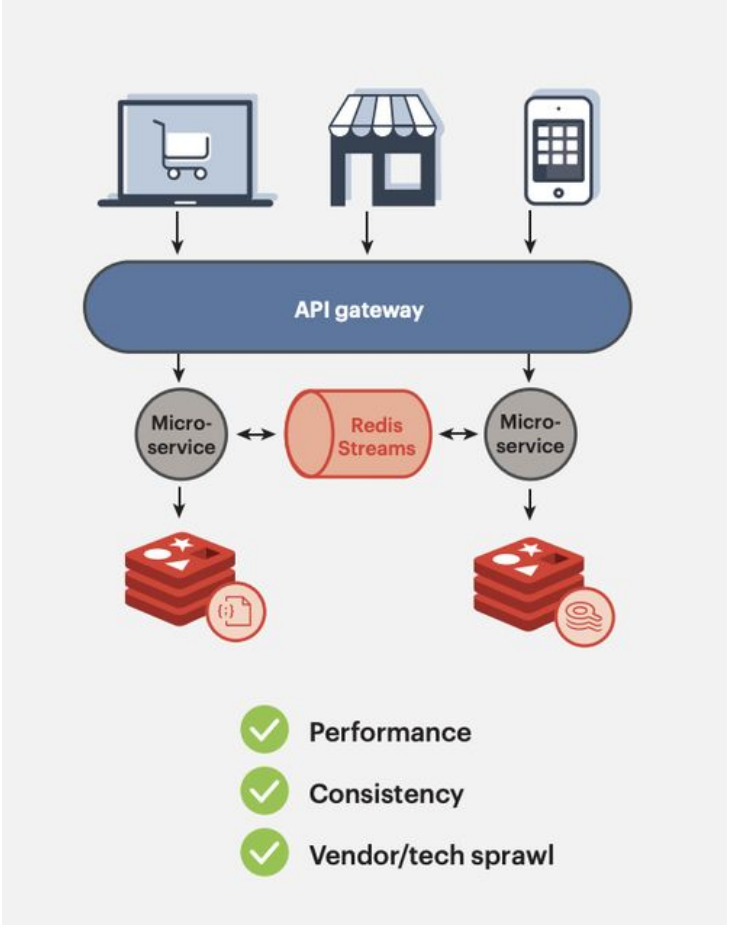




Redis Use Cases (Cont.)

Modern multi-channel retailers are turning to real-time inventory systems to optimize their inventory, yield, and supply-chain logistics, with the aim of improving the customer experience and supply chain.

Building and maintaining these complex systems is daunting for application developers.



Redis Use Cases: Data Ingest

Fast data ingest are complex and over-engineered for simple requirements such as streaming real-time data from the internet of things (IoT) and event-driven applications.

Running Redis Single Instance in Kubernetes

1. Lets create a pod:

   kubectl create deployment redis --image=redis:latest

2. Connect to the pod:

   kubectl exec -it [**redis-pod-name**] -- sh

3. Execute some queries:

- redis-cli
- info replication
- ping

Redis

Lets deploy redis and write a simple program to test the redis connectivity!

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis
          ports:
            - containerPort: 6379
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  type: NodePort
  selector:
    app: redis
  ports:
    - name: redis
      port: 6379
      targetPort: 6379
      nodePort: 30008
```

```python
import redis
import random
import string
# Create a Redis client
r = redis.Redis(host='192.168.49.2', port=30008, db=0)
# Set a value in Redis
r.set('key', 'value')
# Get a value from Redis
value = r.get('key')
print(value)
keys = r.keys('*')
print(keys)

# Set 10 random keys with random values in Redis
for i in range(10):
    # Generate a random key and value
    key = ''.join(random.choices(string.ascii_lowercase, k=5))
    value = ''.join(random.choices(string.ascii_lowercase, k=10))
    # Set the key-value pair in Redis
    r.set(key, value)

# Print all the keys and values in Redis
for key in r.keys():
    value = r.get(key)
    print(f"{key.decode('utf-8')}: {value.decode('utf-8')}")
```

Redis Cluster

- Redis Cluster is a distributed implementation of Redis that allows for horizontal scaling and high availability.
- It allows you to distribute data across multiple nodes while maintaining performance and reliability.
- Redis Cluster uses consistent hashing to distribute keys across the cluster, and provides automatic failover and replication for high availability.
- Unlike standalone Redis instances, Redis Cluster allows you to scale horizontally by adding or removing nodes to the cluster.
- Redis Cluster also supports partitioning of data to ensure that data is distributed evenly across the nodes, and load balancing to ensure that each node is utilized efficiently.

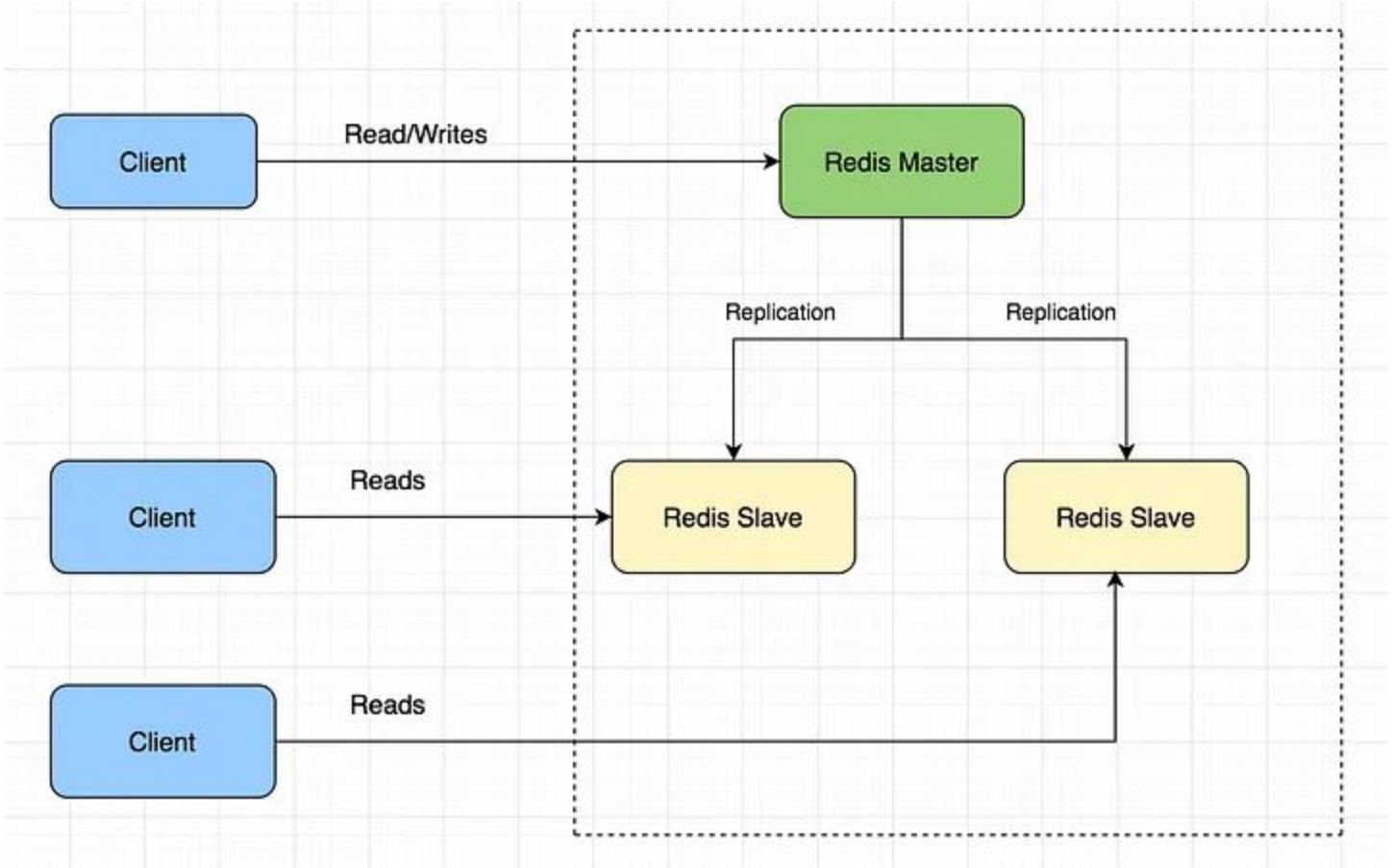
Redis Cluster (Cont.)

Redis Cluster uses several TCP ports for various functions. Here are the TCP ports used by Redis Cluster:

1. Cluster bus port (default 16379): This port is used for inter-node communication, such as cluster configuration updates and node failure detection. It uses the Redis Cluster bus protocol, which is a binary protocol based on TCP.
2. Cluster TCP port (default 6379): This port is used for client communication with the Redis Cluster. Clients connect to this port to send Redis commands and retrieve data.

Redis Cluster data sharding

- Redis Cluster uses data sharding to distribute data across the cluster, which means that each piece of data is stored on a specific node in the cluster.
- Redis Cluster partitions the keyspace into a fixed number of hash slots (by default, 16384) and assigns each slot to a specific node.
- When you write data to Redis Cluster, the data is hashed to determine which slot it belongs to. The node responsible for that slot is then responsible for storing and serving that data.
- Resharding is the process of moving hash slots from one node to another in order to balance the load across the cluster.
- Redis Cluster automatically performs resharding when a node becomes too heavily loaded or when new nodes are added to the cluster.
- Each key in Redis must be mapped to a single hash slot, which means that you need to ensure that the key's hash is mapped to a single slot.


Client
Read/Writes
Redis Master
Replication
Replication
Client
Reads
Redis Slave
Redis Slave
Client
Reads

Reference

https://redis.io/docs/management/scaling/

https://github.com/sobotklp/kubernetes-redis-cluster