1.memcached VS Redis

	Memcached	Redis
Sub-millisecond latency	Yes	Yes
Developer ease of use	Yes	Yes
Data partitioning	Yes	Yes
Support for a broad set of programming languages	Yes	Yes
Advanced data structures	-	Yes
Multithreaded architecture	Yes	-
Snapshots	-	Yes
Replication	-	Yes
Transactions	-	Yes
Pub/Sub	-	Yes
Lua scripting	-	Yes
Geospatial support	-	Yes

Redis uses a single core and shows better performance than Memcached in storing small datasets when measured in terms of cores. Memcached implements a multi-threaded architecture by utilizing multiple cores. Therefore, for storing larger datasets, Memcached can perform better than Redis.ElastiCache cluster supports these two engines.

2.Functionality of Redis

Has persistence mechanism

1. Support point-in-time snapshot.
2. AOF(append only file) logs every write operation..

3. LRU

```
package DatabaseDay1;
import java.util.HashMap;
import java.util.Map;
class LRU {
    class LinkNode {
        int key;
```

```
        int value;
        LinkNode prev;
        LinkNode next;
    }

    private void addNode(LinkNode node) {

        node.prev = head;
        node.next = head.next;

        head.next.prev = node;
        head.next = node;
    }

    private void removeNode(LinkNode node){

        LinkNode prev = node.prev;
        LinkNode next = node.next;

        prev.next = next;
        next.prev = prev;
    }

    private void moveToHead(LinkNode node){
        removeNode(node);
        addNode(node);
    }

    private LinkNode popTail() {

        LinkNode res = tail.prev;
        removeNode(res);
        return res;
    }

    private Map<Integer, LinkNode> cache = new HashMap<>();
    private int size;
    private int capacity;
    private LinkNode head, tail;

    public LRU(int capacity) {
```

```
        this.size = 0;
        this.capacity = capacity;
        head = new LinkNode();

        tail = new LinkNode();

        head.next = tail;
        tail.prev = head;
    }

    public int get(int key) {
        LinkNode node = cache.get(key);
        if (node == null) return -1;

        moveToHead(node);

        return node.value;
    }

    public void put(int key, int value) {
        LinkNode node = cache.get(key);

        if(node == null) {
            LinkNode newNode = new LinkNode();
            newNode.key = key;
            newNode.value = value;

            cache.put(key, newNode);
            addNode(newNode);

            ++size;

            if(size > capacity) {
                LinkNode tail = popTail();
                cache.remove(tail.key);
                --size;
            }
        } else {
            node.value = value;
            moveToHead(node);
        }
```

```
    }
}
public class LRUMain {
    public static void main(String[] args) {
        LRU lRUCache = new LRU(2);
        lRUCache.put(1, 1);
        lRUCache.put(2, 2);
        System.out.println(lRUCache.get(1));
        lRUCache.put(3, 3);
        System.out.println(lRUCache.get(2));
        lRUCache.put(4, 4);
        System.out.println(lRUCache.get(1));
        System.out.println(lRUCache.get(3));
        System.out.println(lRUCache.get(4));
    }

}
```

4.AWS: Elastic Cache

Amazon ElastiCache makes it easy to set up, manage, and scale distributed in-memory cache environments in the AWS Cloud. It provides a high performance, resizable, and cost-effective in-memory cache, while removing complexity associated with deploying and managing a distributed cache environment. ElastiCache works with both the Redis and Memcached engines.