Short Description of mongoDB:

mongoDB is a nonrelational document-based database. Documents consist of key-value pairs, with values being various kinds of data.
Due to mongoDB's being nonrelational database, it has a dynamic schema that allows users to add additional data to the key.

- CAP:
    - Consistency simply means, when you write a piece of data in a system/distributed system, the same data you should get when you read it from any node of the system.
    - Availability means the system should always be available for read/write operation.
    - Partition-tolerance means, if there is a partition between nodes or the parts of the cluster in a distributed system are not able to talk to each other, the system should still be functioning.
    - We generally categorize RDBMS in **CA**. Because Relational databases are a single node system and hence we do not need to worry about partition tolerance and hence if RDBMS server is up and running, it will always respond success for any read/write operation.

mongoDB with CAP:
Not with A:
MongoDB supports a “single master” model. This means you have a master node and a number of slave nodes. In case the master goes down, one of the slaves is elected as master. This process happens automatically but it takes time, usually 10-40 seconds. During this time of new leader election, your replica set is down and cannot take writes.
MongoDB can always be Consistent based on how you configure your client and the way you write data(using write options) and can be always available for reads but you can never make write always available, there will always be downtime when:
a) the new leader is getting elected
b) the client driver disconnects from the leader

Range partitioning with mongoDB:

MongoDB uses the shard key associated to the collection to partition the data into chunks. A chunk consists of a subset of sharded data. Each chunk has a inclusive lower and exclusive upper range based on the shard key.
MongoDB splits chunks when they grow beyond the configured chunk size. Both inserts and updates can trigger a chunk split.

Does mongoDB support ACID?
MongoDB 4.0 introduces multi-document transactions for replica sets and can be used across multiple operations, collections, and documents. The multi-document transactions provide a globally consistent view of data, and enforce all-or-nothing execution to maintain data integrity.
The transactions will have following properties

- When a transaction is committed, all data changes made in the transaction are saved.
- If any operation in the transaction fails, the transaction aborts
- When a transaction aborts/aborted, all data changes made in the transaction are discarded.
- Until a transaction is committed, no write operations in the transaction are visible outside the transaction.

In short, Yes for Atomicity.
In MongoDB, Transactions (also called multi-documents transactions) are associated with a session. That is, you start a transaction for a session. At any given time, you can have at most one open transaction for a session.
You can not lock the entire collection for writes. You may want to create multiple transactions to ensure that writes are not interlacing/overriding between your processes. MongoDB uses Optimistic Locking

- Optimistic Locking
    - is a concurrency control
        - begin: record a timestamp of transaction
        - modify: read data, tentatively write changes
        - Validate: Check whether other transactions have modified data that this transaction has used (read or written). This includes transactions that completed after this transaction's start time, and optionally, transactions that are still active at validation time.
        - Commit/Rollback: If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction

Memcached vs Redis

- Both memcached and Redis are open-source in-memory key-value data store, both belong to the NoSQL family

    - in-memory data store: database that keeps the whole dataset in RAM, each time we query a database, we don't need to access the disk, thus vastly decreasing the response time. But at the same time in-memory databases also risk losing the data when a process or server is down or rebooted, we can minimize the risks by persisting write operations in disks using snapshots or taking logs.
        - For non-change operations (retrieve data), in-memory databases do not use the disks.
        - For change operations (update, add or delete), in-memory databases write to main memory and also persist on disks.
            - Writing to disks: Transactions are applied to the transaction log in an append-only way. (When addressed in this append-only manner, disks are pretty fast. 100mb/sec)
- Data Partitioning
    - Both Redis and Memcached are able to distribute the data across multiple nodes
- Value data structures
    - In addition to strings, Redis supports lists, sets, sorted sets, hashes, bit arrays, and hyperloglogs.
- Multithreading
    - Memcached supports multithreading, and it can scale up the computing usage.
- Snapshots
    - Redis supports snapshots, in case when the server/process is down or rebooted.
- Replication
    - Redis lets you create multiple replicas of a Redis primary. This allows you to scale database reads and to have highly available clusters.
- Transactions
    - Redis supports transactions which let you execute a group of commands as an isolated and atomic operation.

AWS ElastiCache

From AWS documentation:

Amazon ElastiCache is a web service that makes it easy to set up, manage, and scale a distributed in-memory data store or cache environment in the cloud. It provides a high-performance, scalable, and cost-effective caching solution. At the same time, it helps remove the complexity associated with deploying and managing a distributed cache environment.

High Availability:

AWS ElastiCache achieves high availability through automatic failover detection and mitigation. Primary node failures, for instance, are immediately resolved by standby replicas. The same applies to reading operations — if the primary one is busy, the read replicas swing into action to serve the data and keep your application running smoothly.