To scale a application from o to 1 million users we have to follow some steps. Initially there was single server means application server and database is in the same server.

Next we will separate application server and database in different serves. We will call application server as mid tier and database server as data tier. We sperate them so that we can scale up them individually if needed. In the application server there is all the logic of the system located.

Next we will create multiple application server and add load balancer. We will create multiple application server in the mid tier so that too many requests can be distributed. So that network call traffic can be low. We will add load balancer between client and mid tier. It will decide which application server will handle the request. Load balancer also gives some sort of security because it communicate with application servers through private ip and client can not directly talk to the server. Data can be encrypted.

Next we have to replicate database. We can use here master db and slave db. There can be only one master db. But slave db can be multiple. Write operation happened in master db and read operation happened in slave dbs. All the dbs will be in sync. If one slave db fails, other slave will handle the request. If master database fails, one slave db will become master.

Next we will add cache. It is a very expensive call from mid tier to database. So to reduce this calls we can add caching. We can do caching using redis. Cache has the same data as the db. Before make any request to database first the mid tier will try to get data from cache if it gets the data, it will be called cache hit and if it does not get the data, it will be called cache miss. If it misses the cache, then it will try to get the data from database. Cache is used with mid tier. Cache has something called ttl. TTL means time to live. A certain time is set after that time mid tier will get data from database and update the cache with new ttl. If mid tier misses cache then it will also update the cache with new data and new ttl.

CDN is content delivery network. It can be consider as cache for static content or less frequently changed content. Static contents are html, css, video etc. Lets say there is data center in a particular region. There are various users of the whole system from various locations. So because of one data center the user who is located near to that data center will get the content very quickly. The user who is far away from the region of data center will get latency. So solve this problem CDN is used. There is various cdn node in various locations which are copy of the data center. If user makes any request, he/she will get the content from his/her nearest cdn node. If the nearest cdn node does not have the content then that cdn node will ask for the content to its nearest cdn node. If again fails then it will ask data from the data center. It increases performance and also ensures security. For example DDOS attack. As the request directly does not go to the data center, so it is more secured. Moreover we can make the CDN nodes intelligent enough so that it can track bots. We can add CDN with client.

Next we can add multiple data centers. Each data center is consist of application server, cache and databases means mid tier, cache and data tier. Load balancer sends a request to a data center based on geo request. So the traffic gets distributed. If one data center face faults. The request will be fulfilled from another data center. Each data center has the similar copy of data tier.

Next we can add messaging queue. Messaging queue can be created using rabitMQ or kafka. It is used to handle asynchronous task which takes lots of time like notification sending or email sending. There is a producer who add message to the message queue. Then the subscriber of that queue will get the notification. If the message fails, it will add to the requeue. You can again send the message from the requeue means you can retry. In messaging queue, there is an exchange and some queues. Exchange is connected to the queues through bindings. There are difference ways to pass the message from exchange to queue. The first way is direct. In this way the routing key is equal to whichever binding key of queue the message will be pass to that queue. And the subscriber of that queue will get the message. Another way is fan out. In this way the message will be sent from exchange to all the queues. Users subscribe to those queues will get the message. They may ignore if they want. The last way is called topic. It is a bit similar to the first way. In this way message will be sent to multiple queues. For example whichever binding keys of queues partially match with the routing key sent by the producer, the exchange will push the message to those queues. The subscribers of those queues will get the notification. It will add with application server and cache.

Next we can do database scaling. We can do database scaling using two ways one is vertical scaling and another one is horizontal scaling. In vertical scaling we can increase the capacity of existing database but there is certain limitations of ram and cpu. On the other hand horizontal scaling is adding more database nodes. We can do horizontal scaling through shading. Shading can be possible in two ways. One is horizontal and another one is vertical. In horizontal shading we will divide a large table into rows. For example from row 1 to 10 will be in one table and from row 11 to 20 will be on different table. In vertical shading the table is divided into columns. For example from column 1 to 5 in one table and column 6 to 10 is on another table. Generally horizontal shading is better. You can also do vertical shading. There are some drawbacks of horizontal shading. For example we do shading between rows based alphabet of a particular column. But lets say after shading some alphabets has more data. Then we have to again do shading. In this manner it may create a shading. We don't know how to do shading. Another drawback is as the data is divided into rows we can not do joining of the table. We can solve this using denormalization. To solve the drawback of shading tree we can use consistent hashing.

=======================

Questions

=======================

1. Deeply understand load balancer, cache and cdn

2. Deeply understand master slave, message queue

3. Deeply understand database scaling

4. Deeply understand shading