Let say you have an application which is running on a single web server, which in turn is also acting as a Database server.

When you access the URL of your application, myapp.test.com, the request goes to DNS (Domain Name System) which finds the IP addresses to this URL and redirects the request to the machine with that IP address, which is nothing but your web server.

Considerations for Scaling this Application –

1. Load Balancer – Instead of directly hitting the server, we can direct the request to the load balancer’s public IP address. This load balancer will internally talk to the web servers using their private IP address. Load balancer will help to direct the request to the server which is healthy and capable of handling the request.
2. Database – We need to move the DB out of web server so that we can scale them separately. A DB can be replicated to improve performance and provide better reliability and scalability. By replication we mean multiple copies of the DB. We can make one of them as a Master DB which handles the write operations and others as Slave DBs which handles the read operations. When Master is not available, one of the slave DB will become the Master.

We need to scale this DB too. There are two ways – vertical and horizontal scaling

Vertical scaling also means scale-up which means increasing the capacity of the server we have like RAM, CPU, etc. But there is a limit up to which we can scale vertically.   
Horizontal scaling means we can add more servers. This is called partitioning. A partitioning key is used to distribute the load. For example if user id is the partitioning key and a hash function is used for deciding which user info will be stored on which server, user\_id : 0,2,4 can be stored on server 1 and 1,3,5.. can be stored on server 2. Drawback is sometimes load can be unevenly distributed and sometimes there can be more read requests for a particular user on a particular server. Another drawback is performing joins can be really tough when data is partitioned like this.

1. Cache – Before hitting the DB, some frequently used information can be cached. This will improve the performance further. The cache servers can also be scaled and replicated.   
   Some considerations – cache expiry time, cache eviction policy, inconsistency, etc.
2. CDN – Content Delivery Network. Some static pages of a website can be stored in a CDN which is geographically distributed cache to load the website quickly
3. Session – We should make the web server stateless instead of stateful. We can have more than 1 web server and if each of the web server maintains the session info for some users, some servers will be overloaded. It is better to move this info out of web server and store in a separate global DB from where the web servers can request session info. This will also allow auto-scaling of these web servers based on the load.
4. Monitoring, Logging, Metrics – It is important to maintain logs to debug any issue. Keeping track of metrics help to analyze the load, performance , revenue, etc. of the servers
5. Data centers – This entire setup can be replicated many times over several DCs. GeoDNS can be used to route the traffic to the DC which is geographically closer to the user.

How will this flow work -?

1. Request comes to DNS from myapp.loadbalanced.test.com
2. DNS resolves the IP address and directs the request to the geographically closer DC. Also CDN will load the static content of the webpage
3. Load balancer will decide which web server to direct request to.
4. Web server will check the cache if information is present. It also checks the session information.
5. If info cannot be served from the cache , it is fetched from the DB