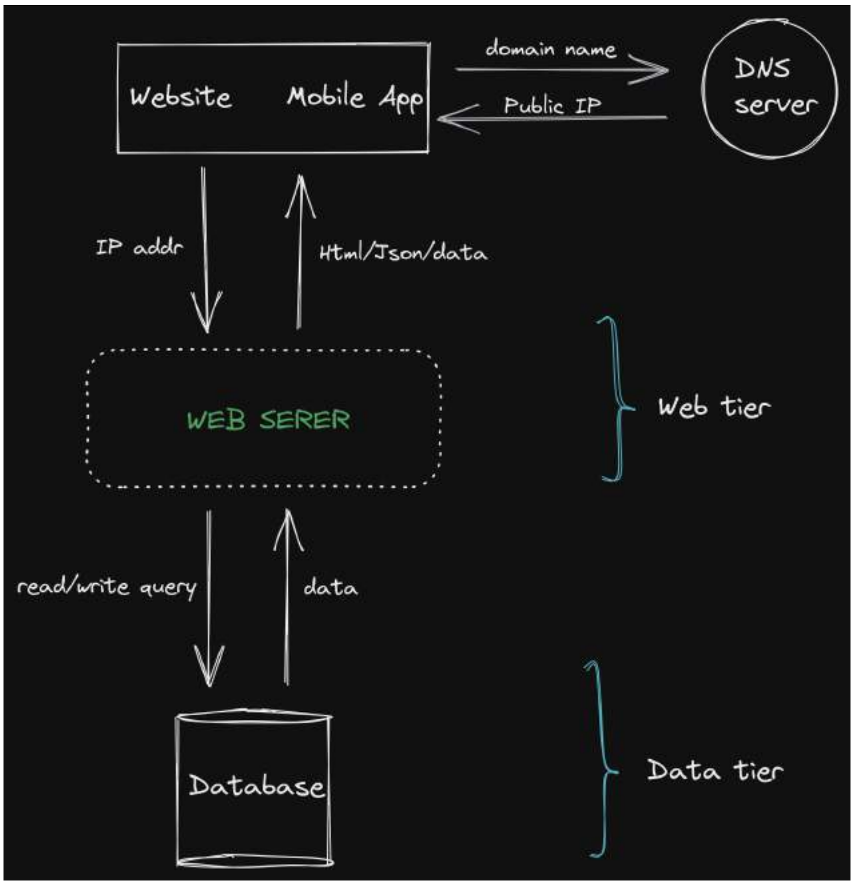
🖥️

Zero to Millions of User

We will start with the most basic setup and move forward to more advanced options.

Single Server Setup



Database types:

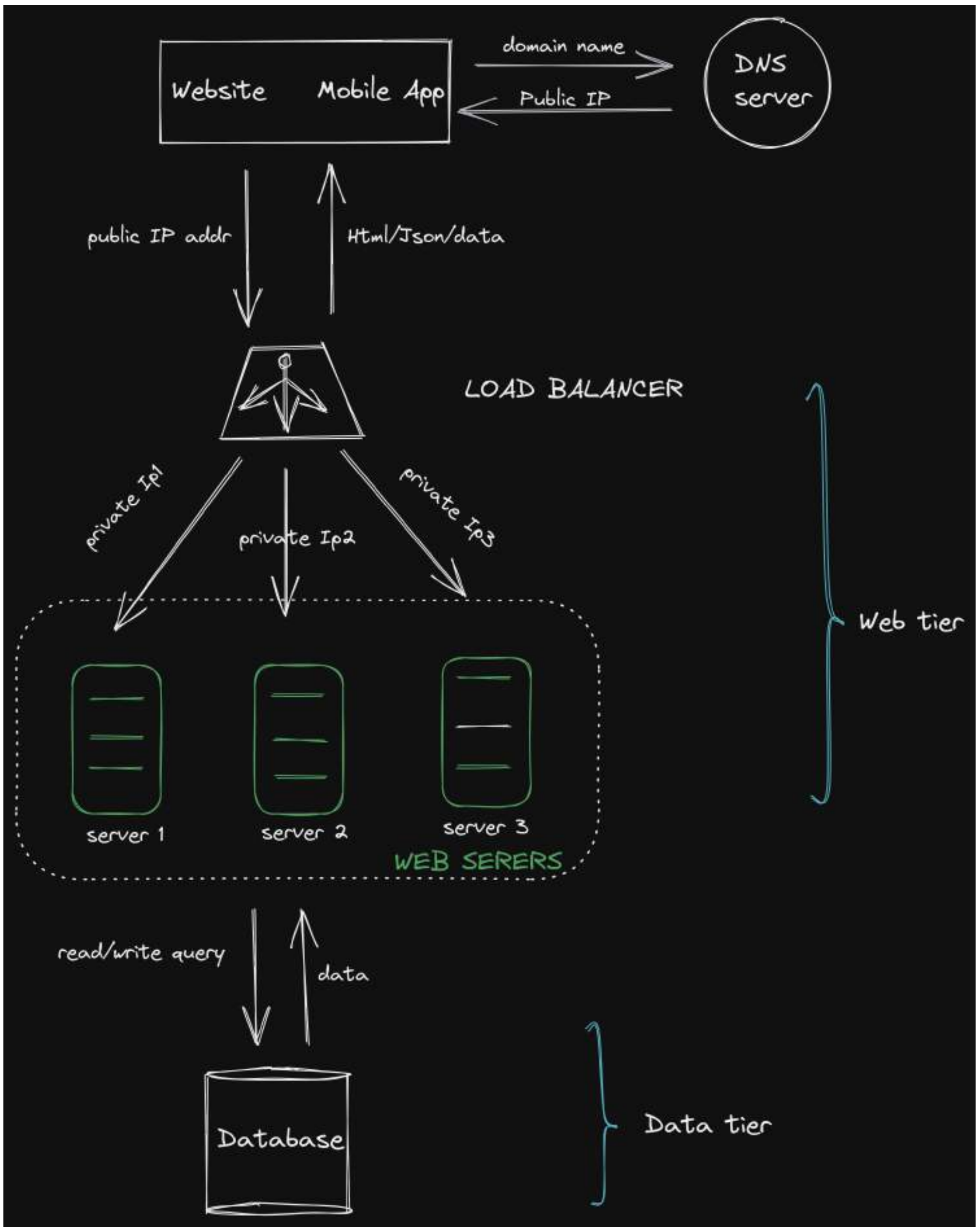
1. **Relational Databases (RDBMS)**
    a. Data is stored in rows of the table. We can perform join operations on tables.
    b. Ex: MySql, Oracle, etc.
2. **Non-relational Databases (NoSQL)**
    a. No fixed structure for stored data. These are stored in key-value pairs, documents, etc.

b. Ex: MongoDB, DynamoDB etc.

c. Used when we have large data and want super low latency

Adding Load Balancer

Load Balancer divides the traffic to multiple servers to avoid overloading any single server. This prevents the website from going offline.

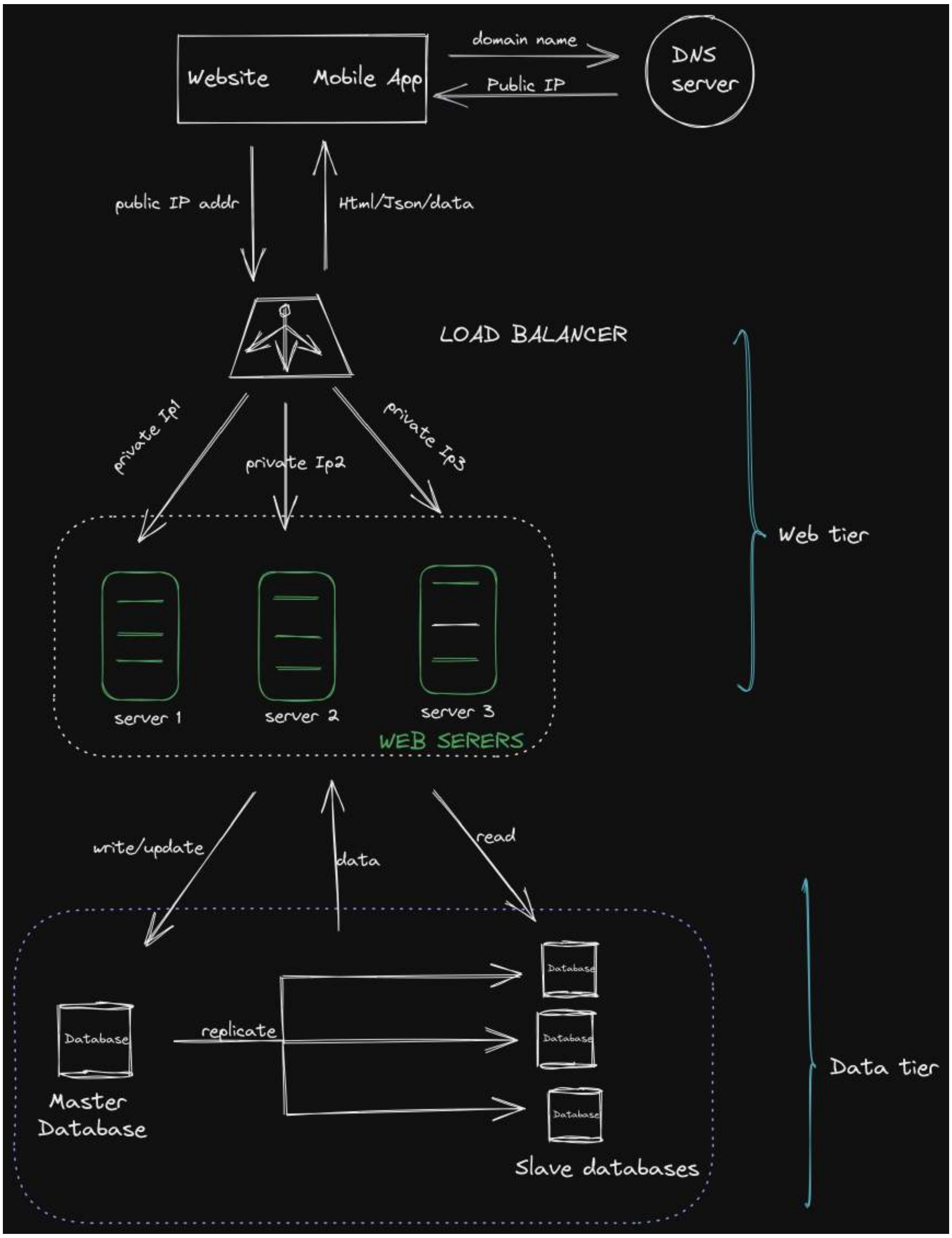


Now, web tier looks good, what about data tier

Master/Slave Database system

We replicate the master database into multiple slave databases, with this, we send all the **read** queries to slave databases and **write** to the master database.

In the real world, # read >>> # write, that's why we need a large number of slave databases compared to master databases.



Both tiers look fine. Now let's look at new methods which will be helpful in reducing the response time.

Caching And CDN (Content Delivery Network)

Cache is present between servers and database, which temporarily stores the frequently requested data from the database, to provide fast response time. Algorithms like LRU, and FIFO are used in caches.

CDN are used to provide static data (Html,Css,Js files) fastly to user. CDN servers are established around the whole world at different locations to provide faster response.

We need to set cache expiry time according to our requirements. If some data is time sensitive, then it should be low.

Web Tier types:

1. **State Less:**
    a. servers do not store user data/state. Instead, the session data of users is stored on shared storage (database or cache).
    b. scalable.
2. **State Full:**
    a. Servers store user session data/ state not in the database.
    b. The issue with this is, For example, if the user has some items in his cart and he wants to check out before that say server 1 (to which he was connected by a load balancer) goes down, then the user will be moved to other server but his state will we lost which means cart will get empty. **!! Big problem.**
    c. There is no benefit in horizontal scaling of the system/ web tier.
    d.

Message Queue

A message queue is a durable component, stored in memory, that supports asynchronous communication. It serves as a buffer and distributes asynchronous requests. The basic architecture of a message queue is simple. Input services, called producers/publishers, create messages, and publish them to a message queue. Other services or servers, called consumers/subscribers, connect to the queue, and perform actions defined by the messages. The model is shown in Figure 1-17.

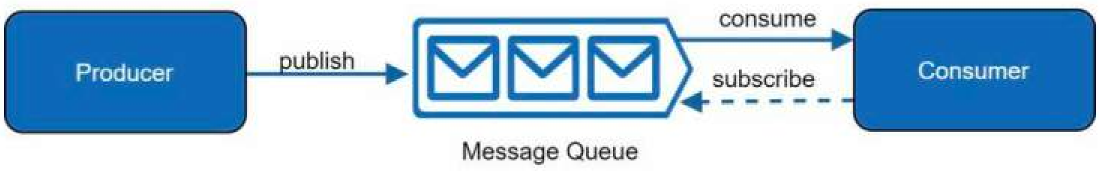
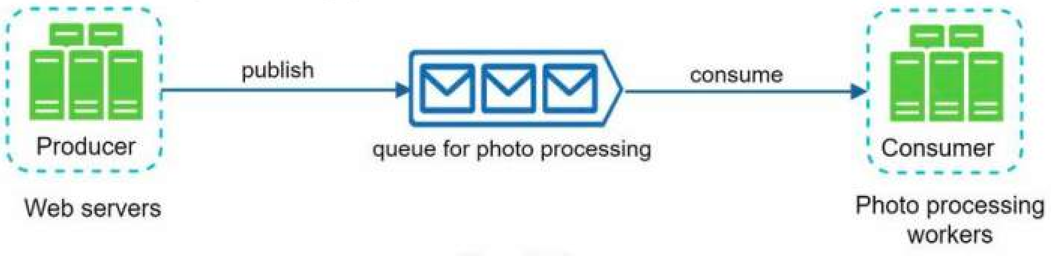


Figure 1-17

Decoupling makes the message queue a preferred architecture for building a scalable and reliable application. With the message queue, the producer can post a message to the queue when the consumer is unavailable to process it. The consumer can read messages from the queue even when the producer is unavailable.

Consider the following use case: your application supports photo customization, including cropping, sharpening, blurring, etc. Those customization tasks take time to complete. In Figure 1-18, web servers publish photo processing jobs to the message queue. Photo processing workers pick up jobs from the message queue and asynchronously perform photo customization tasks. The producer and the consumer can be scaled independently. When the size of the queue becomes large, more workers are added to reduce the processing time. However, if the queue is empty most of the time, the number of workers can be reduced.



Logging, Metrics, and Automation

Logging: Monitoring error logs to identify system errors and problems. Logs can be monitored per server or aggregated to a centralized service.

Metrics: Collecting metrics provides business insights and system health status. Examples include host-level (CPU, memory), aggregated (database, cache), and business metrics (active users, revenue).

Automation: Automation tools improve productivity in complex systems. **Continuous integration** verifies code check-ins, automating processes like build, test, and deploy for enhanced developer productivity.

Database Scaling

1. **Vertical scaling:** It is scaling up the CPU power by adding more RAM, disk, etc to the existing system. But it is limited due to hardware limitations. And it is too costly.

2. **Horizontal scaling / Sharding:** To increase the number of servers present in the system

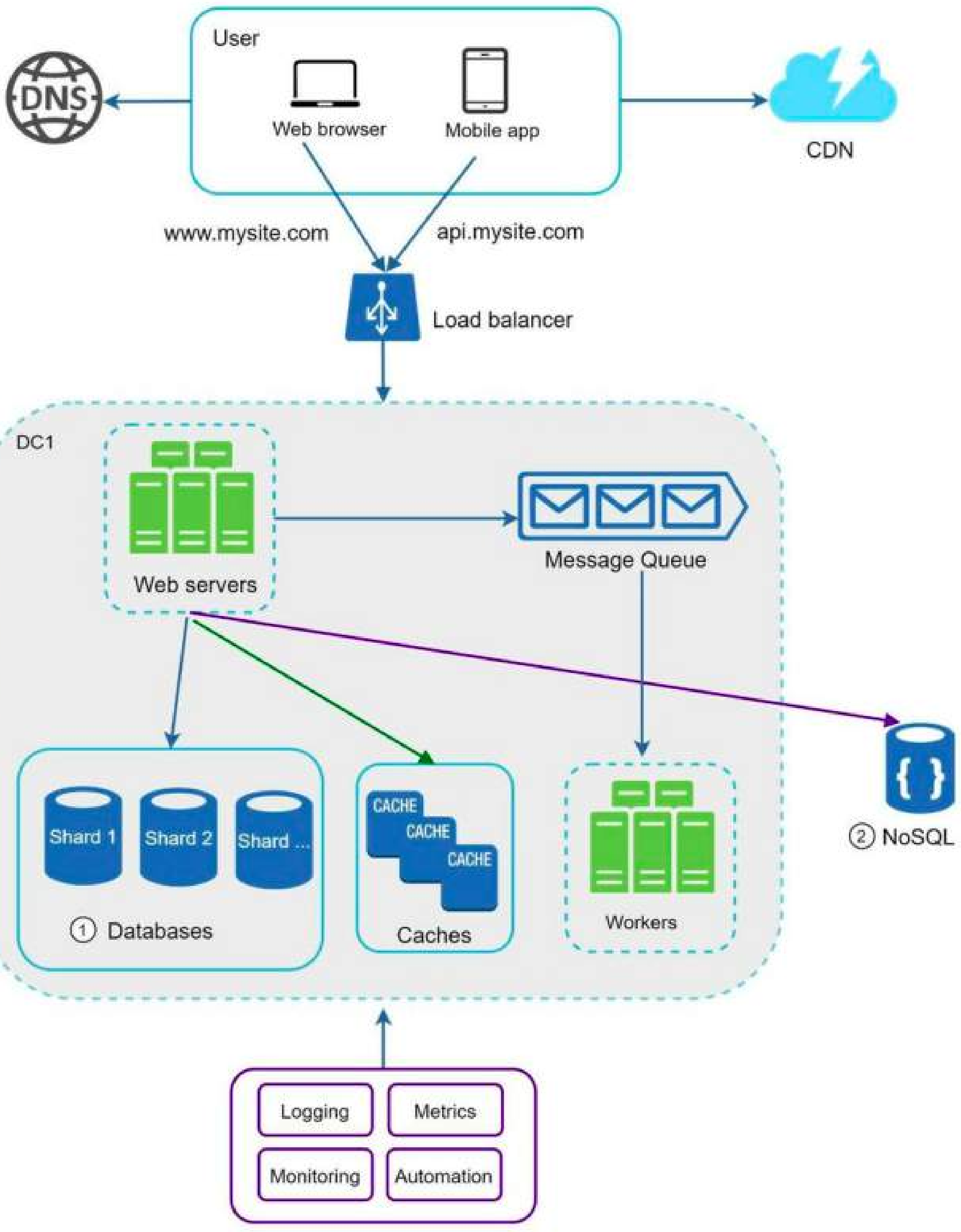
   In sharding, we divide the whole set of servers in to small groups called shards.

   Each shard in the system holds unique data. When a user registers on the website, they are assigned to a specific shard based on a sharding algorithm/hash function. The shard detail is included in the request header when the user makes a request to the server. This allows us to route the request to the user's assigned shard efficiently.

   Problem: When a shard becomes overloaded, it needs to be resharded, which involves making changes to the hash function. This can be a complex and challenging process.

   Solution: Consistent hashing is used to address this issue.

Final System design



Tools Figure 1-23