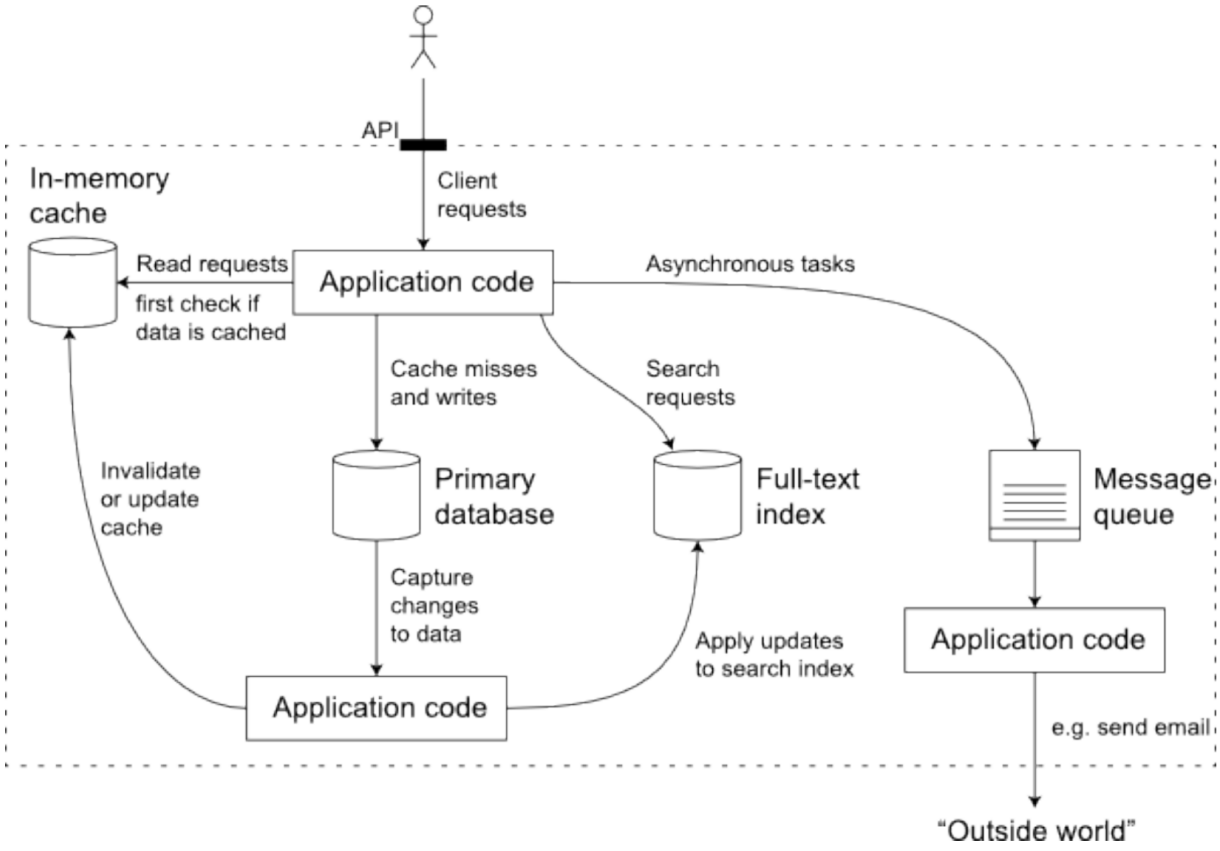
An application is data intensive if the **amount of data that it generates**/uses increases quickly or if the **complexity of data** that it generates/uses increases quickly or if the **speed of change in data** increases quickly.

A typical data-intensive application looks like this:



Main components in a data-intensive applications include

1. **Databases** for source of truth for any consumer. E.g. Oracle/MySQL
2. **Cache** for temporarily storing an expensive operation to speed up reads. E.g. Memcache
3. Full-text **Index** for quickly searching data by keyword or filter.e.g. Apache Lucene
4. **Message queues** for messaging passing between process. E.g. Apache Kafka
5. **Stream processing**. E.g. Apache Samza, Apache Spark
6. **Batch processing** for crunching large amount of data. E.g. Apache Spark/Hadoop
7. **Application code** which acts as the connective tissue between the above components.

Role of an application developer is to design the data system for **reliability**, **scalability** and **maintainability**.

Reliability is fault tolerance from human, software or hardware faults. The system can ensure there is no unauthorized access to systems, output is as per expectation and performance is good enough for the application to be usable. Deliberately doing Chaos testing helps with finding out issues. Hardware components have faults and thus the design of the system should be such that it can handle full machine failure. Bugs need to be fixed and caught at a much earlier phase and thus system needs to be testable and monitorable. Automating tests and staging/testing environment needs to present. And there needs to be a way to quickly roll-back when failure happens. Reliable systems are the basis of companies revenue, legal compliance and productivity of users of the system.

Scalability ensures the system can grow with higher traffic volume and complexity of new systems get added. Describing traffic load with peak #of reads,writes and simultaneous users helps model the system. Based on read heavy/write heavy patterns appropriate frontloading or backloading of processing needs to be built. Capacity planning is part of scalability and it is best to plan for capacity uplift regularly. Online systems care more about response time, offline systems are mainly around throughput. Remember end user response time includes your server response time + a lot of network in between. Monitoring for both system performance and perceived performance is thus quite critical. 90th,95th Percentile is a good way to measure and define SLO/A service level objectives/agreements. There are two major scaling techniques: scaling up (more powerful machine), scaling out (distributed many smaller machines).

Maintainability ensures that as we add new people to work on the software that productivity is not impacted for developers. There are three questions you can ask to check if your systems are maintainable:

1. **Operable**: Is is easy to get system up and running and monitor system health? Configurable and testable systems are easy to operate and maintain.
2. **Simple**: Is is easy to understand and ramp up on this system? Using good design patterns and having good documentation and clean code helps new people ramp up quickly.
3. **Evolveable**: Is is easy to change the system for issues and add new features? Refactor code regularly to make abstractions easier to understand and remove unintended code debt regularly.

There is no one magic solution that works for all data-intensive application use cases. Your solution will be custom to your needs, as long as you have a good idea of the above three pillars of Reliability, Scalability and Maintainability you will build the right system.