Chapter 1 - Design Scalable systems

Type	Designing Data-Intensive Applicaitons, Resource
Content	Design, Principles

💡 Today, many applications are data-intensive, as opposed to compute expensive. Raw CPU is rarely a limiting factor, and a bigger problem is usally the amount of data, the complexity of the data and the speed at which is changing.

✏️ **Message queue:** A form of asynchronous service-to-service communication used in serverless architectures. Messages are stored and queued until they are processed and deleted.

Three concerns that are important in most data-intensive software systems:

- Reliability: The system should still work correctly even in the face of adversity, wether it is human error, software faults, and hardware faults.
- Scalability: The system should be able to handle rapid growth in it's demands.
- Maintainability: As more people interact and work on the system, it can become more complex, the ability for these people to work productively should be maintained.

Reliability

✏️ Reliability means making systems work correctly, even when faults occur. Fault-tolerance techniques can hide ceetain types of faults from the end user.

- A bug can remain dormant for as long as their assumptions are met, however a change in the envioronment can trigger the bug and bring the system down. This is called a software fault, and is very common. A solution to reduce these types of bugs is by decoupling the places where people make the most mistakes away from the people. Provide sandbox environments where people can explore and work with real data without affecting real users.
- Test thorougly at all levels, unit tests, whole-system integration tests, and manual tests.
- Set up detailed and clear monitoring, such as performance metrics, and error rates. Monitoring can show us early warning signs and allow us to check wether our assumptions are validated.

Scalability

✏️ Scalability means having strategies for keeping performance good, even when load increases. To discuss scalability we must define load. Under a scalable system you can add processing capacity in order to remain reliable under high load.

- Could the system perform as well with 10,000 users as it could with 1,000,000.
- We must first describe load, this can be described with load parameters:
    - The best choice of load parameters can be described by load parameters
    - Load parameters choices depends on the architecture of the system
    - An example would be writes per second
- With a load parameter described you will have to experiement with it to understand how they intereact with your system. How much do to system resources increase

with an increase in your load parameter?

⚠️ Latency vs response time: Response time is what the client sees, includes network delays, and queueing delays. Latency is the duration that a request is waiting to be handled. During which it is latent, awaiting service.

- We use percentiles to monitor latency and response times, as they can change randomly and the mean does not represent the average waiting time for the average user.
- An architecture that scales well for an application is built around assumptions of which operations will be common and which will be rare- the load parameters. If those assumptions are wrong then a lot of effort is wasted, and at worst counterproductive.

Maintainability

✏️ Maintainability is about making life better for the engineering and operations team who work with the system. Good abrastraction can reduce complexity and make the system easier to modify and adapt for new use cases. Good operability means good visibility into the system's health, and having effective ways of managing it.

Design principles for maintainability:

1. **Operability**: Make it easy for teams to keep the system running smoothly
2. **Simplicity**: Make it easy for new engineers to understand the system, by removing as much complexity as possible
3. **Evolvability**: Make it easy for the engineers to make changes to the system in the future, adapting it for unanticipated use cases as requirements change.

Good operability means making routine tasks easy. This can include:

- Providing visibility into the runtime behavior
- Avoiding dependency on individual machines

Abstract away complexity is key.