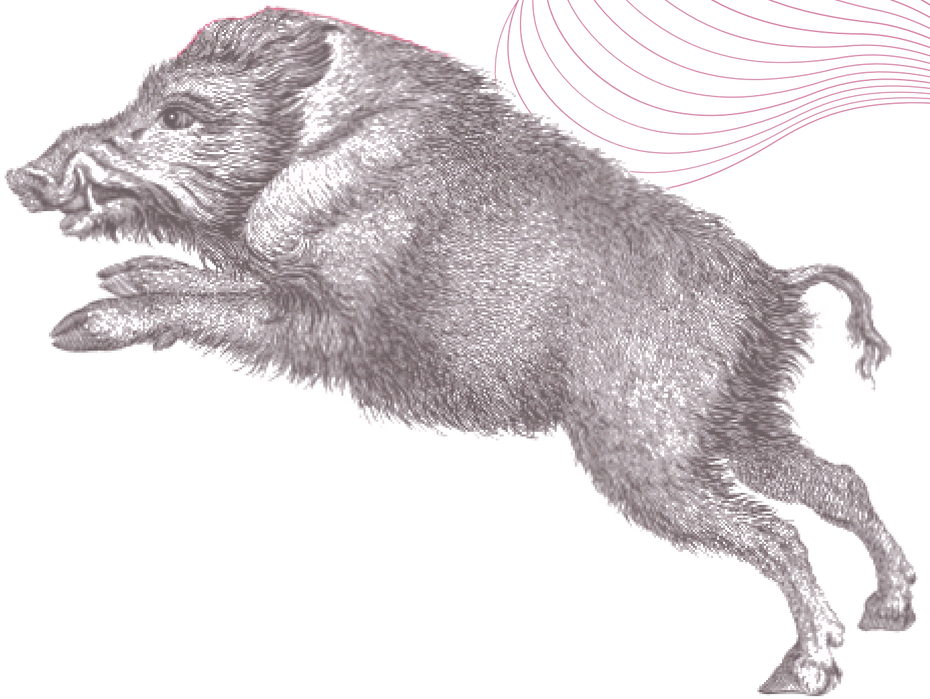
Designing Data-Intensive Applications

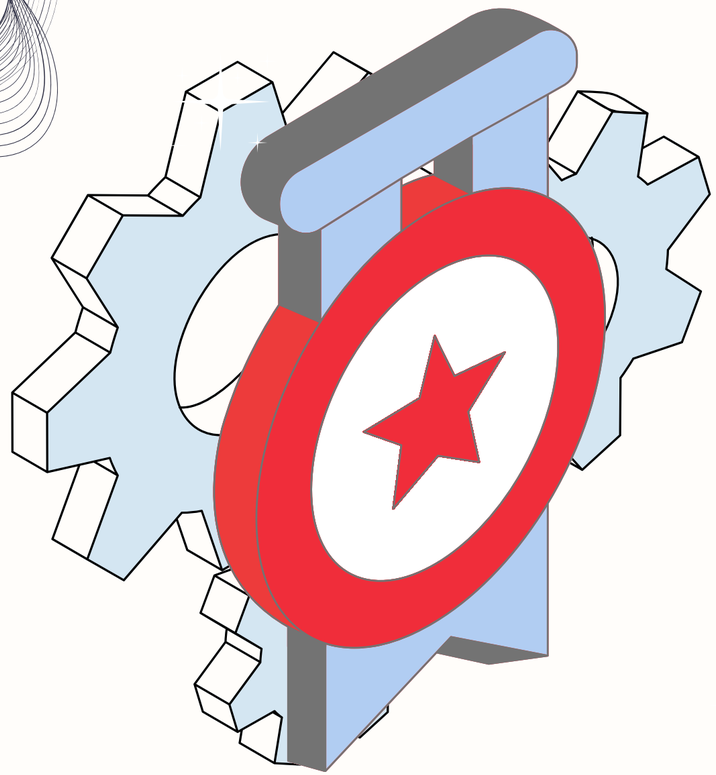


A book by:
Martin Kleppmann

Chapter 1

Reliable, Scalable, and Maintainable Applications

Reliability



Definition

Continuing to work correctly even when things go wrong

Why???

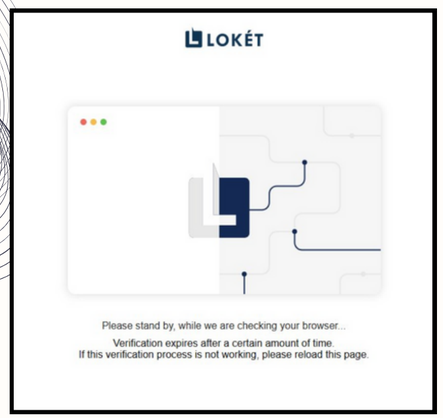
To find a solution or an alternative when something goes wrong

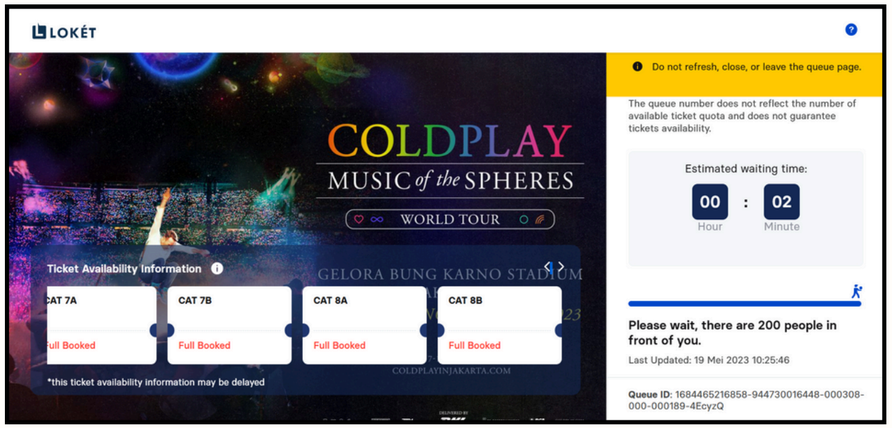
Reliability

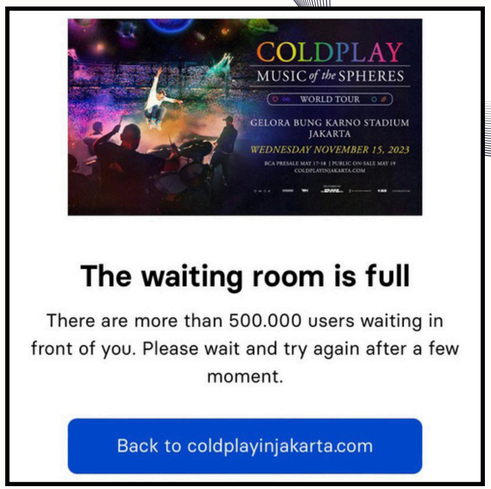
Typical Expectation

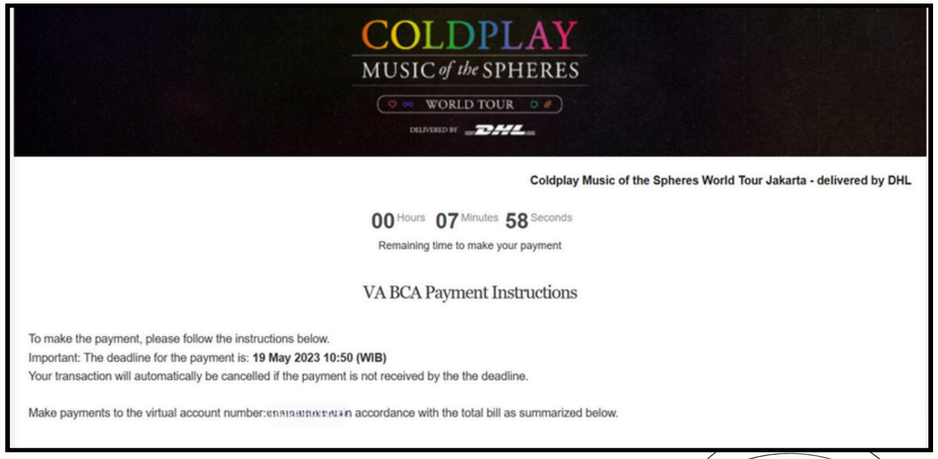
- Application performs the function the user expected
- Tolerate the user making mistakes or using the software in unexpected way
- Its performance is good
- The system prevents unauthorized access and abuse.

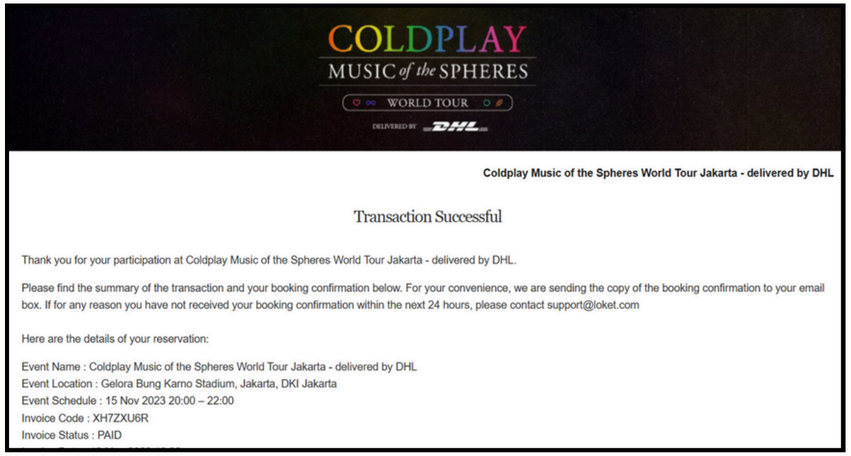
Examples of Good Reliability











Kinds of Faults

Hardware Faults

Software Faults

Human Errors

Hardware Faults

Physical failures in hardware that can disrupt the system. We can improve resilience to hardware failures by using hardware redundancy (RAID, dual power supplies, hot-swappable components)

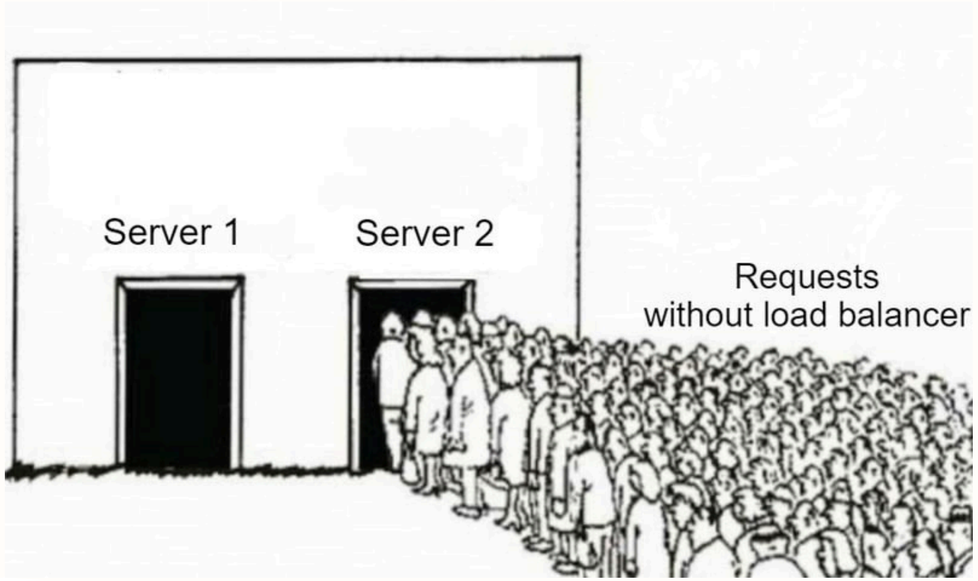
Example with hardware redudancy

- **Initial State:** Server has 2 PSUs, one connected to the main power, the other to a UPS/generator.
- **Issue Occurs:** PSU 1 fails, but the server keeps running on PSU 2
- **Action Taken:** Team gets a notification and replaces PSU 1 without shutting down the server (hot-swap).
- **Recovery:** New PSU is installed, and the server returns to using both power sources.
- **Result:** No downtime, data remains safe, and services

Hardware Faults

Software fault tolerance techniques

- Modern systems adopt software fault-tolerance techniques (data replication, load balancing, rolling upgrades)



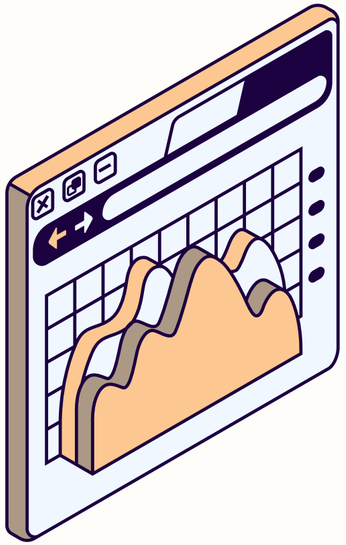
Example

- **Initial State:** The app runs on 3 servers with a load balancer.
- **Issue:** 1 server crashes, but the other 2 handle requests.
- **Action:** The load balancer redirects traffic to the active servers.
- **Recovery:** After repair, the server is reintegrated automatically.
- **Result:** Zero downtime, uninterrupted service, and system stability.

Software Faults

Characteristic

- Difficult to detect
- Systematic & cascading failures
- No quick fix



Example

- **Initial State:** A video streaming app with multiple microservices.
- **Issue:** A memory leak in the recommendation service spikes CPU and RAM, disrupting playback.
- **Action:** Engineers isolate, fix, test, and redeploy the service.
- **Recovery:** The service is gradually restored with performance monitoring.
- **Result:** Core services stay online, downtime is minimized, and the bug is resolved.

Tolerating Faults

Human Errors

Causes

- Misconfiguration settings
- Lack of Understanding
- Pressure & Fatigue

Solution

- Minimize Opportunities for Mistakes
- Decouple High-Risk Actions
- Enable Quick & Easy Recovery
- Use Detailed & Clear Monitoring
- Provide Good Management & Training

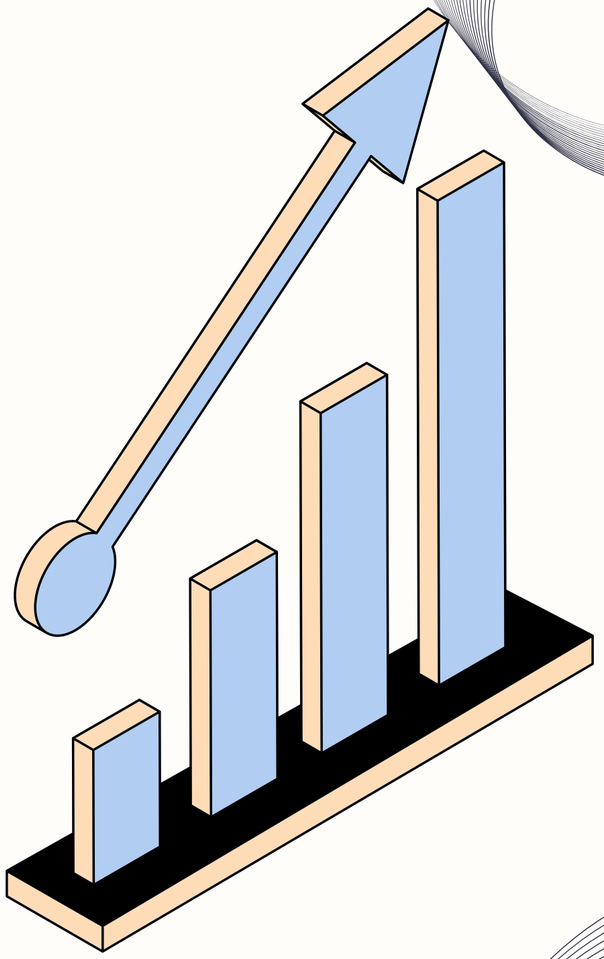
Scalability

Definition

Asking if the system grows in a particular way, what are the options for coping with that growth

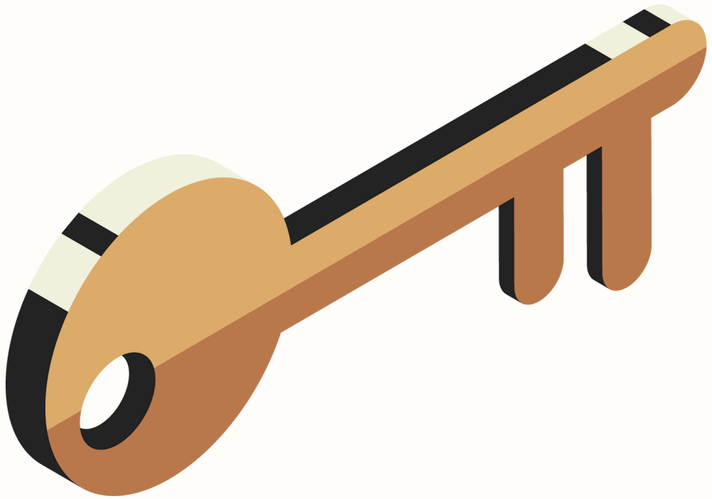
Why???

A system should be able to handle additional loads, such as an increasing number of users or data volume, without experiencing a decrease in performance



Scalability

Key Considerations



- Load
- Perfomance
- How to cope with load?

Scalability

Load

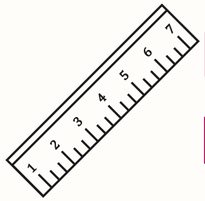
- Requests per second
- Read/Write ratio
- Active users
- Cache hit rate

Scalability

Perfomance

Latency

Response time



Perfomance measured by percentiles

Percentiles are used in SLA

Scalability

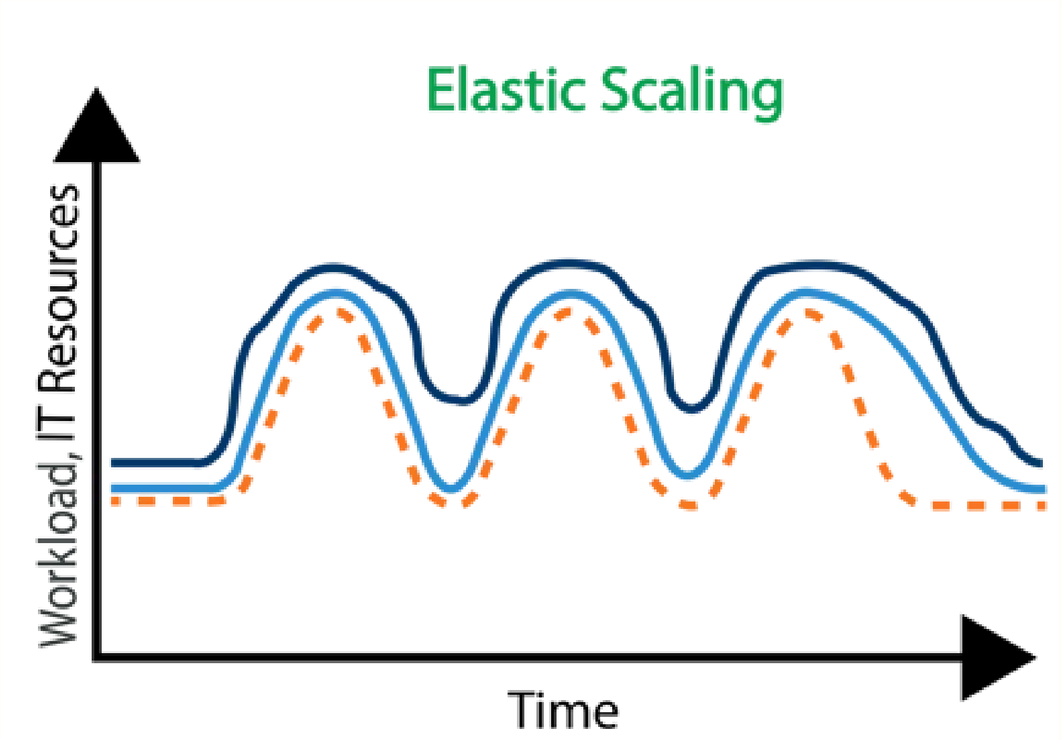
How to Cope with Load?

Scale Up

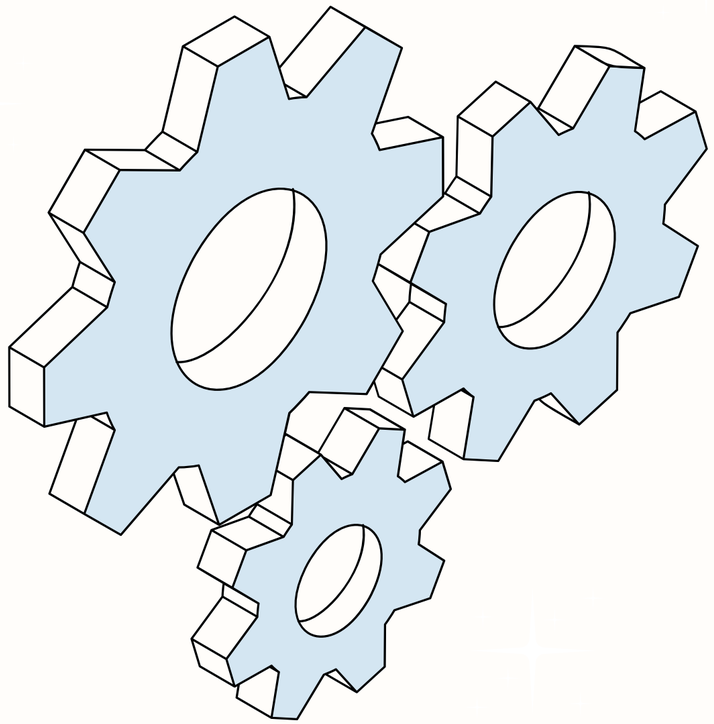
Scale Out

Scalability

How to Cope with Load?



Maintainability



Definition

Keeping the system easy to modify and adaptable for future changes or improvements

Why???

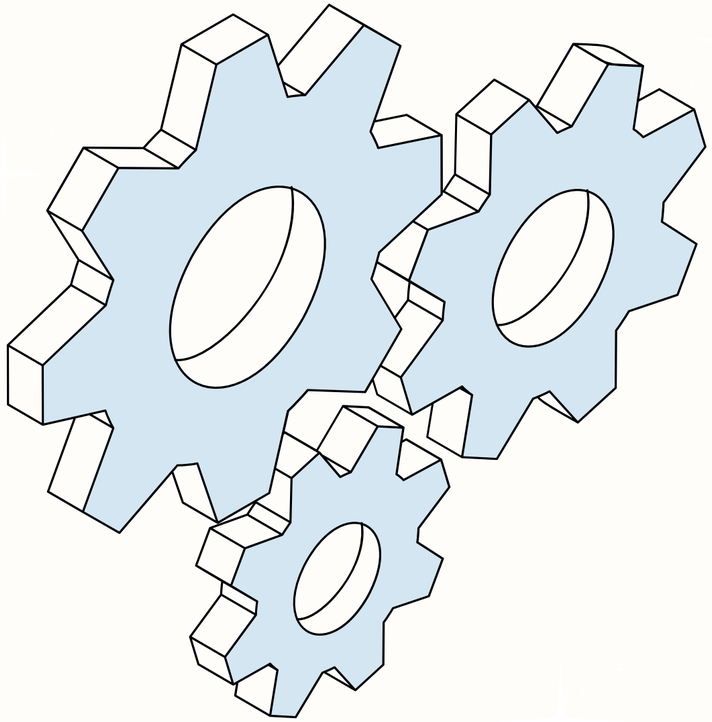
Eases the burden on engineers who work with the system by preventing difficulties in debugging, developing new features, and fixing bugs, which can lead to higher operational costs

Maintainability

Lack of Maintainability Example

- Initial State: An e-commerce app lacks clear documentation and organized code.
- Issue: Adding a seasonal discount feature is difficult due to overlapping functions and inconsistent variables.
- Impact: Slower development, more bugs, and longer debugging times.
- Result: Delayed feature release, lost revenue, and higher costs to fix the code.

Maintainability



- Operability
- Simplicity
- Evolvability

Operability

Measuring how easy it is for the IT team to operate and maintain the system on a daily basis

How?

- Monitoring system and quickly restoring
- Tracking system failure and degraded perfomance
- Keeping software up to date
- Keeping track how one system affect the other
- Establishing good deployment, configurations practices
- Avoiding dependency or individual machines
- Good documentation

Simplicity

What is simplicity?

- The simpler the design and code, the easier it is engineers to understand

How to reduce complexity

- Clean Code
- Modular Design
- Use descriptive and consistent names
- Refactoring
- Using design patterns
- Documentation & comments

Evolvability

Making it easy for engineers to make changes into the system in the failure

Why it is needed?

- Business Priorities change
- Users request new features
- New plaforms
- Architectural changes

How?